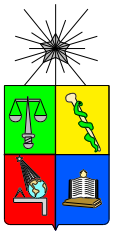

UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
DEPARTAMENTO DE INGENIERIA EN COMPUTACIÓN

ESTUDIO DEL DESEMPEÑO DE ESTP EN REDES DE ALTA LATENCIA QUE IMPLEMENTAN EL USO DE UN TCP PROXY

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO E INGENIERO CIVIL EN COMPUTACIÓN

ANDRÉS RICARDO ABUJATUM DUEÑAS

PROFESOR GUÍA:
CLAUDIO ESTÉVEZ MONTERO
PROFESOR GUÍA 2:
JAVIER BUSTOS JIMÉNEZ

MIEMBROS DE LA COMISIÓN
ANDRÉS MUÑOZ ÓRDENES
MARCOS DÍAZ QUEZADA

SANTIAGO DE CHILE
AGOSTO 2014

SANTIAGO DE CHILE 2013
RESUMEN DE LA MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRICISTA E INGENIERO CIVIL EN CIENCIAS DE LA COMPUTACIÓN.
POR: ANDRÉS ABUJATUM DUEÑAS.
FECHA: 23 DE DICIEMBRE DE 2013
PROFESOR GUÍA ELÉCTRICIDAD: CLAUDIO ESTÉVEZ MONTERO
PROFESOR GUÍA COMPUTACIÓN: JAVIER BUSTOS JIMÉNEZ

"ESTUDIO DEL DESEMPEÑO DE ESTP EN REDES DE ALTA LATENCIA QUE IMPLEMENTAN EL USO DE UN TCP PROXY"

En la actualidad las telecomunicaciones han tomado un rol preponderante en el diario vivir de las personas, esto por la gran cantidad de servicios que se ofrecen, como por ejemplo video streaming, redes sociales, llamadas Voice over IP (VoIP), etc. y que día a día van consumiendo mayores cantidades de ancho de banda. Como consecuencia de esto la transmisión de información ha ido aumentando su tamaño de manera acelerada. En el ámbito científico, principalmente la astronomía, se están recogiendo grandes cantidades de datos todas las noches, provenientes de distintos puntos de investigación como por ejemplo el Telescopio Digital Sky Survey (SDSS) reúne 200 GB de información cada noche o su sucesor el Telescopio para grandes rastreos sinópticos (LSST) que empezará a funcionar en el 2020, se estima que generara cerca de 30 TB por noche. El Gran Colisionador de Hadrones (LHC) produce cerca de 15 PB de información al año y estos son solo 3 lugares, pero hay más, en otras áreas y es por esto que hay un desafío a nivel de transmisión de datos.

El presente trabajo tiene como objetivo principal, utilizando el protocolo de transporte creado por el profesor Claudio Estévez llamado Protocolo de Transporrte de Servicios Ethernet (ESTP, *Ethernet Services Transport Protocol*), generar un modelo analítico del throughput de ESTP en redes de alta latencia que implementen uno o más servidores de proxy TCP, y demostrar que el protocolo ESTP tiene un rendimiento igual o superior a TCP en dichas redes, es decir, el throughput va aumentando, en particular para cuando se envían grandes volúmenes de datos. Esto porque las implementaciones actuales de TCP presentan problemas con las redes de alto delay en transmisiones de datos. Esto último se comprueba mediante una serie de experimentos.

Los resultados obtenidos son bastante alentadores y proponen a ESTP como una solución factible a las transmisiones de grandes volúmenes de datos en redes de alta latencia, esto porque logra superar a otros protocolos de control, como TCP Reno que forma parte de los orígenes de TCP y también CUBIC y BIC, que son las versiones más actuales y usadas de TCP. Estos resultados fueron presentados y aceptados en *IEEE International Conference on Comunications 2012*, en *IEEE LATINCOM 2013* y *IEEE CHILECON 2013*, actualmente existe una publicación que se encuentra en proceso de revisión en el IEEE *Journal on Computers Comunications 2013.*

Se deja como propuesto el poder implementar ESTP en un satélite y realizar pruebas en el espacio. Desarrollar un proxy TCP en redes inteligentes y así mejorar la velocidad de transmisión y poder implementar una mayor cantidad de servicios.

"A mis padres Ricardo y Carolina, y mi hermana Paula por su infinita paciencia, apoyo incondicional y sabias palabras"

Agradecimientos

En primer lugar a mi familia por todos los años de espera y sacrificio. Su comprensión y cariño incondicional ha sido muy importante para mi, para lograr todo esto.

En segundo lugar a los profesores de mi comisión, Claudio Estévez, Javier Bustos, Marcos Díaz y Andrés Muñoz por el tiempo dedicado y sus importantes indicaciones a este trabajo.

A mis amigos Alonso, Sebastián, Fernando, Alejandro y Jaime, quienes han estado junto a mi en toda esta etapa universitaria.

A Loreto y Javiera, por su alegría, paciencia y comprensión desde el primer año.

Al profesor Juan Álvarez por permitirme ser profesor auxiliar y descubrir que hacer clases es una de la grandes pasiones de mi vida.

Al equipo de Basketball por estar ahí siempre que los necesite, en especial, Milton, Oscar, Mariela y Gabriela.

Al CRI, grupo en el cual conocí a grandes personas y me permitió desarrollar habilidades más allá de las académicas.

A mis amigos, alumnos y compañeros que de alguna u otra forma formaron parte de mi vida universitaria, sin ustedes no habría disfrutado estos años como lo hice, gracias a todos ustedes.

Finalmente a Tomás Mosqueira y Natalia Jiménez por su gran y desinteresada ayuda en la elaboración de este documento.

Índice de contenido

1 Tabla de contenido

Índice de Figuras

Índice de Tablas

Capitulo 1: INTRODUCCIÓN

1.1 **Motivación**

Los ingresos de los servicios Ethernet de negocios continúan creciendo de forma constante. Se ha proyectado que para el 2013 este mercado tendrá un valor de $ 38,9 mil millones en todo el mundo[1]. A la vez que otros mercados de telecomunicaciones entran en recesión, los servicios Ethernet continúan siendo adquiridos, a gran velocidad, por empresas que buscan nuevas alternativas para apoyar velocidades de red más grandes en distintas tipos de aplicaciones metropolitanas, regionales, nacionales y globales. Solo en EEUU, la demanda de los puertos Ethernet de negocios de servicios se expandió a una tasa del 43% en el 2008[2].

Una cantidad importante del tráfico de internet, incluyendo acceso remoto (Telnet), correo electrónico (SMTP), transferencia de archivos (FTP) y WWW (HTTP) es transmitida mediante el protocolo de transporte TCP, siendo la velocidad de transmisión un factor de preocupación entre los investigadores. Así nació el concepto de Carrier Ethernet, que es el uso de tecnología de banda ancha para acceder a internet y comunicar redes locales de trabajo, académicas y gubernamentales a alta velocidad.

Esta tecnología dio pie a la creación de la Red de Carrier Ethernet (CEN, *Carrier Ethernet Networks*) que permite conectar grandes áreas de cobertura a través de enlaces de alto rendimiento como cables de fibra óptica, cobre, entre otros. Algunas ventajas que posee esta red son flexibilidad, estabilidad, bajos costos de operación, simpleza e interoperabilidad.

Las redes CEN poseen canales de banda ancha de bajo costo, como por ejemplo fibra mono modo, lásers de alta velocidad y técnicas de transmisión en paralelo, la velocidad de datos puede llegar al rango de los Terabytes por segundo [*Terabits per second*], haciendo posible para muchas compañías conectar sus redes, creando una red *backhaul* de carrier Ethernet de multi-dominio, que puede cubrir todo el mundo. Esta cobertura consiste de enlaces de 10 Gbps.

Pero estas redes también requieren de un protocolo que pueda identificar como se mueve el tráfico de paquetes sin la necesidad de saber explícitamente las direcciones de la fuente y el destino. El Protocolo de Control de Transporte (TCP, *Transport Control Protocol*) se presenta como el protocolo por defecto que se utiliza, porque evita la congestión, garantiza la entrega de paquetes y organiza la secuencia de datos, haciendo que el protocolo de mejor rendimiento de internet sea confiable, asumiendo, eso sí, el *trade-off* de que al ser más confiable también posea más retraso.

El dilema se presenta en las redes de alta latencia BDP (*Bandwith-delay product*) con la transmisión de archivos de gran tamaño. El protocolo TCP -el más usado en la actualidad-, tras una serie de pruebas presenta problemas de rendimiento debido a las maneras en que combate la congestión. Esto quiere decir que la ventana no crece a la velocidad que corresponde, al no poder soportar la información almacenada en los búfer, provocando una situación similar a un efecto embudo.

[1] Vertical Systems Group, "Worldwide Business Ethernet Services Market Rises to $38.9 Billion by 2013," Vertical Systems News, March 2009
[2] Vertical Systems Group, "Business Ethernet Expands 43% in 2008," Vertical Systems News, Feb 2009

Para mitigar esto, existe un algoritmo de disminución multiplicativa que se encarga de controlar el tamaño de la ventana de congestión. Esto quiere decir que puede transmitir una cantidad de paquetes a la vez, lo que también disminuye rápidamente el tamaño de cada pérdida.

Para mejorar estas limitaciones del TCP Tradicional sobre el BDP han sugerido varios protocolos como alternativas: HighSpeed TCP, Scalable TCP, XCP, BIC, CUBIC y ACP. Cada uno de éstos presenta mejoras distinguibles, pero el problema es que algunos son incompatibles o no son atractivos para las redes CEN.

Las principales razones de esto son:

1. Calidad de transmisión del operador: Aquí la transparencia es obligatoria lo que descarta todos los protocolos que son dependientes del router.

2. Directivas de tráfico: Se utilizan en la interfaz usuario/red (UNI), por lo que cambios drásticos en la ventana de congestión (Congestion Window, CWND) se pueden ver en formas de ráfaga. Esto descarta los protocolos con algoritmos de aumento agresivo y disminución agresiva de la ventana.

3. Se requiere equidad entre tráficos con el mismo nivel de prioridad, lo que resulta en preferir protocolos justos.

La capa de transporte utiliza y prefiere los puntos mencionados (1, 2 y 3), porque conoce estas razones, en particular el Committed Information Rate (CIR) y el Excess Information Rate (EIR) que aprovechan el flujo de tráfico para obtener *feedback*. El Protocolo de Transporte de Servicios Ethernet (ESTP, *Ethernet Service Transport Protocol*) está diseñado para atenuar los efectos del algoritmo de disminución, mediante el ajuste de forma dinámica de la tasa de transmisión, de acuerdo con la intensidad de la congestión estimada dentro de la red y que se ajusta en la capa de transporte de forma de obtener ventaja de las reglas (1, 2 y 3).

Debido a que TCP es un protocolo orientado a la conexión, y por lo tanto necesita de retroalimentación (ACK, *ACKnowledge*), las redes con alto tiempo de ida y vuelta (RTT, *Round Trip-Time*) retardan la retroalimentación y reducen el desempeño de forma inversamente proporcional a RTT (Tasa de transmisión proporcional a 1/RTT). Esta es la motivación principal para el estudio de protocolos de transporte más eficientes, como es el caso de ESTP.

En este trabajo se confirma que TCP presenta los problemas antes expuestos y se presenta a ESTP como una alternativa confiable para reemplazar a TCP en las redes BDP, para ello se utilizarán tres métodos de validación: Un modelo analítico, simulación y pruebas experimentales, en donde se comparará ESTP con otros protocolos de transporte orientados a la conexión de alto rendimiento, en particular TCP, en diversas configuraciones, incluyendo rendimiento y pruebas de congestión.

El resto del documento se estructura en: La motivación para la presente propuesta, luego los objetivos del proyecto y finalmente la metodología que se utilizó para su desarrollo.

La motivación de esta memoria es realizar un estudio del problema de TCP y analizar por qué sus implementaciones actuales bajan notablemente su rendimiento en redes de alta latencia al

enviar grandes volúmenes de datos, junto con proponer a ESTP como una solución real para las nuevas generaciones de redes CEN, tanto en redes en la tierra como red satelital. En la figura 1 se muestran ambos esquemas de adaptación de TCP, tanto en redes en la tierra como satelitales.

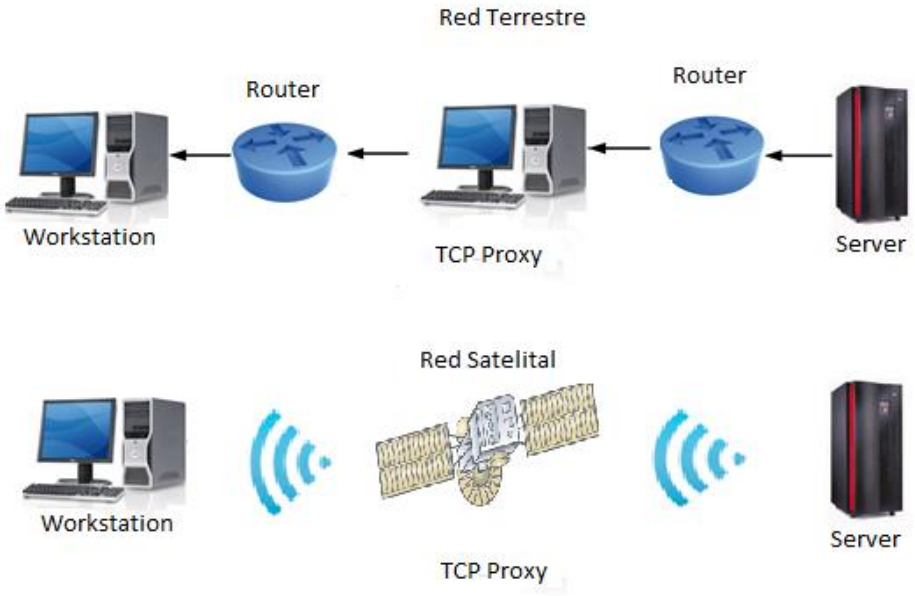

Figura 1: Esquemas de configuración de Workstation-Proxy-Servidor, en redes terrestre y satelital

1.2 Objetivos del Trabajo de Memoria

1.2.1 Objetivo General

Crear un modelo analítico del throughput de ESTP en redes de alta latencia que implementen uno o más servidores de TCP proxy, y demostrar que el protocolo ESTP tiene un rendimiento igual o superior a TCP en dichas redes al enviar grandes volúmenes de datos. Esto incluye también el diseño y la construcción del TCP Proxy con el que se trabajará, esto con el objetivo de añadir mejoras en torno al algoritmo con el que trabaja el proxy.

1.2.2 Objetivos Específicos

i. Estudio de trabajo previo TCP y ESTP.

ii. Derivación de fórmula de throughput para ESTP.

iii. Simulación del protocolo de ESTP con un nodo proxy.

iv. Demostrar de forma empírica que las actuales implementaciones de TCP bajan notablemente su rendimiento en redes de alta latencia al enviar grandes volúmenes de datos.

v. Encontrar la causa del problema en iv.

vi. Validación de ESTP como solución al problema.

1.3 Metodología

Para el desarrollo de este trabajo se seguirán los siguientes pasos:

- **Estudio del trabajo de Previo de TCP y ESTP:** Revisar las investigaciones previas acerca de TCP y ESTP para poder tener un respaldo teórico fuerte.

- **Estudio del software necesario para el desarrollo de ESTP y Simulaciones:** El software con que se trabajara incluye Matlab, Mathematica y OPNET, este último es una herramienta de modelación y simulación de redes.

- **Derivación de la fórmula de ESTP:** Trabajo teórico en que se deriva la fórmula para ESTP, en base a la ecuación inicial de TCP con parámetros libres.

- **Programación en OPNET de ESTP con un nodo TCP Proxy:** Programación de ESTP en C++ en el software OPNET, además de la programación del nodo TCP Proxy en C++.

- **Simulación de ESTP con un nodo TCP Proxy sobre OPNET:** Etapa de simulación del experimento para confirmar que ESTP funciona correctamente en la teoría.

- **Construcción de un Testbed para pruebas en laboratorio:** El testbed deberá ser una estructura capaz de simular el comportamiento de Internet.

- **Implementación de Protocolos en sistema Linux:** Se implementarán TCP Reno, BIC, CUBIC y ESTP en el testbed.

- **Colocar los "Probes" en el cliente, servidor y Proxy:** Definir, implementar de ser necesario e instalar las herramientas que permitan medir las pruebas de transmisión y congestión en el testbed.

- **Análisis de resultados:** Análisis de los resultados obtenidos en la simulación en OPNET y en el testbed, incluyendo una comparación entre ambas pruebas.

1.4 Estructura de la Memoria

En el próximo capítulo (capítulo 2) se discute una revisión bibliográfica, en la cual se revisan algunos conceptos básicos del problema para introducir al lector de una manera más completa.

En el capítulo 3 se hablará del trabajo realizado junto con algunas herramientas que ayudaron en las simulaciones, como OPNET y las herramientas de Linux. Aquí se revisarán las mediciones y simulaciones del protocolo de transferencia nuevo, en comparación con los protocolos TCP ya antes descritos. Además de explicar el código con que se implementó el protocolo.

En el capítulo 4 se analizan las simulaciones y diferentes pruebas realizadas para confirmar la tesis propuesta en el proyecto.

Y finalmente en el capítulo 5, se muestran las conclusiones del trabajo. También en este capítulo, se propone una manera de poder continuar este trabajo o qué cosas podrían ayudar a completarlo en el futuro.

Capitulo 2: CONCEPTOS BÁSICOS

2.1 Introducción

En este capítulo se hace una revisión de conceptos básicos mínimos para poder entender el problema que se aborda en este documento para poder contextualizar al lector. Adicionalmente se muestra el estado de Arte de protocolos de transmisión que se han desarrollado hasta la fecha.

2.2 Conceptos Básicos

2.2.1 Modelo OSI

La *Open System Interconnection* (OSI) definió un modelo de estructura (*framework*) de red para implementar protocolos en 7 capas. El control se transmite de una capa a la siguiente, comenzando por la capa Física o capa 1, esta se encuentra al fondo del modelo.

Tabla 1: Modelo OSI

7	Aplicación
6	Presentación
5	Sesión
4	Transporte
3	Red
2	Enlace de datos
1	Física

La capa en la cual se desarrolla este trabajo es la capa 4, de Transporte. Esta capa tiene como objetivo proveer una transferencia, transparente, de datos entre hosts, además es responsable de errores de recuperación de extremo a extremo (end-to-end recovery) y control de flujo, esto último mediante checksums y ACKnowledgements. Asegura que la transferencia de datos sea completa.

2.2.2 Protocolo de Control de Transmisión

La IEEE define[3] el Protocolo de Control de Transmisión (TCP) como uno de los principales protocolos en las redes TCP/IP. Mientras el protocolo IP solo trabaja con paquetes, TCP permite que 2 hosts puedan establecer una conexión y tengan un intercambio de flujos de datos, junto con garantizar que los paquetes son entregados en el mismo orden en que fueron enviados. TCP Reno es otro nombre para TCP clásico o TCP estándar, algunas de las funciones básicas son explicadas durante este punto.

[3] http://www.ieee.org/education_careers/education/standards/standards_glossary.html

A continuación en la tabla 1, se resumen las definiciones de las variables más importantes utilizadas en este trabajo.

Tabla 2: Tabla de acrónimos útiles

Parámetro	Definición
Ventana de Congestión (CWND)	Esta variable de TCP tiene como función limitar el tamaño de la información que se puede transmitir por TCP un emisor antes de recibir un ACKnowledgment (ACK).
Tamaño en Vuelo	Cantidad de información que ha sido enviado, pero no se ha recibido un ACK, es decir son los segmentos que están todavía "en tránsito", todavía dentro de la red.
Round Trip Time (RTT)	Es el tiempo que ha transcurrido desde la transmisión de un segmento y la recepción del ACK correspondiente.
Segmento	Cualquier información TCP/IP o paquete ACK (o ambos).
Máximo tamaño de un segmento del emisor (SMSS)	Tamaño del paquete más largo que un emisor puede transmitir. Depende del tipo de red usada y otros factores.
Máximo tamaño de un segmento del receptor (RMSS)	Tamaño del paquete más largo que un receptor puede recibir.
Slow Start Threshold (ssthresh)	El SS threshold es el punto de inflexión entre las fases Slow Start y Congestion Avoidance.

2.2.2.1 Número de secuencia y Paquetes de acuso de recibo

TCP utiliza un número de secuencia para identificar cada byte de información. El número de secuencia permite identificar el orden de los bytes enviados desde cualquier nodo, de tal forma que la información pueda ser reconstruida en orden, sin importar que haya sido fragmentada, desordenada o haya ocurrido una perdida. Cuando el receptor recibe un paquete con número de secuencia, este transmite un paquete de acuso de recibo (*ACKnowledgement pACKets*, ACK), estos son utilizados por TCP para certificar la llegada de paquetes SYN (paquetes sincronizados) cuando se establece una conexión, paquetes de información mientras una conexión se utiliza y paquetes FIN cuando se termina una conexión.

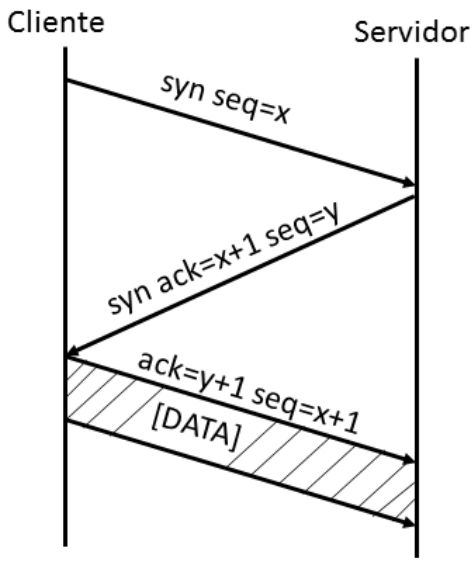

Figura 2: Cómo funcionan los números SEQ y ACK.

Para iniciar el proceso de conexión primero el cliente envía un paquete SYN de sincronización al servidor como parte de la negociación de 3 pasos (3-way handshake), el servidor una vez haya recibido el paquete SYN, transmite un paquete SYN/ACK y para finalizar el cliente responde a este paquete con un ACK, completando de esta manera el 3-way handshake, finalizando el proceso de establecimiento de la conexión.

Cuando se establece la conexión, cada lado genera un número al azar como su número inicial de secuencia (Initial sequence number). Debe ser al azar, puesto que si no, habría problemas de seguridad en caso de que alguien en internet pudiera adivinar el número de secuencia, pues se podrían introducir paquetes en el flujo TCP.

2.2.2.2 Slow Start

El algoritmo de Slow Start tiene por finalidad comprobar la disponibilidad de la red. Esto mediante el crecimiento gradual de la CWND, en vez de comenzar con un valor fijo bien alto que podría llegar a causar congestión. El valor inicial de la CWND debe ser menor o igual que 2 veces el SMSS bytes. El valor de la CWND aumenta hasta que sobrepase el valor del ssthreshold.

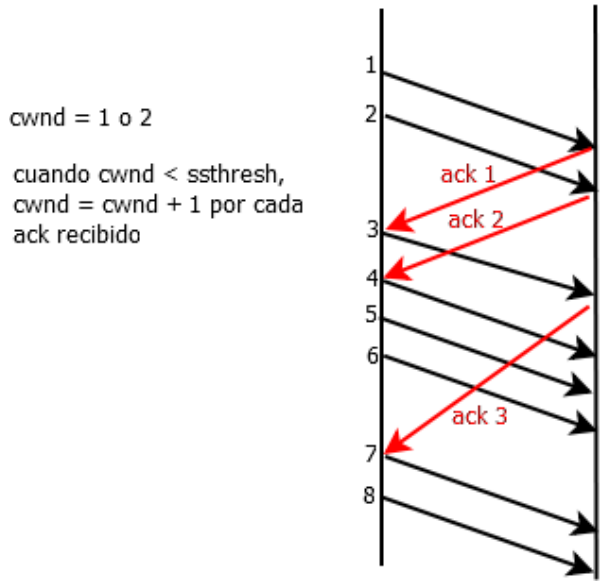

Figura 3: Funcionamiento del algoritmo Slow Start.

2.2.2.3 Fast Retransmit / Fast Recovery

Los algoritmos de Fast Retransmit y Fast Recovery son herramientas que posee TCP para recuperarse de manera adecuada y rápida de distintos tipos de pérdida, el primero realiza una retransmisión rápida del segmento perdido y a su vez el segundo no deja volver a realizar un Slow Start, esto para mantener un ritmo rápido de transmisión. A continuación se explica con mayor detalle cada uno de ellos.

Un receptor TCP debe enviar un ACK duplicado cuando un segmento fuera de orden es recibido. El propósito de este ACK es de informar al emisor que un segmento fue recibido fuera de orden y qué número de secuencia es esperado. Desde el punto de vista del emisor, un ACK duplicado puede significar un sin número de problemas en la red, entre ellos los más importantes se considera la caída de un segmento, lo que genera que todos los segmentos recibidos después del primero que se ha caído envíen un ACK duplicado hasta que la perdida es arreglada. Otro problema puede ser el reordenamiento de segmentos de información hechos por la red misma, lo que es usual en algunas redes. Finalmente algunos ACK duplicados por replicación de ACKs o segmentos de información por la red. Adicionalmente el receptor debe enviar inmediatamente un ACK en caso de que un paquete lleno el segmento buscado. Esto dará más tiempo para una buena recuperación.

El emisor TCP usa el algoritmo de retransmisión rápida para detectar y reparar perdidas, basados en los ACK duplicados, el cual utiliza la llegada de 3 ACK duplicados como una alerta de que un segmento se perdió. Después de recibir 3 ACK duplicados TCP realiza una transmisión del segmento perdido, sin esperar que el contador de retransmisión se acabe.

Después que haya ocurrido la retransmisión de lo que sería el segmento perdido, el algoritmo de “fast recovery” toma el mando de las trasmisiones de información nueva hasta que ningún ACK duplicado llegue. Esto anula el algoritmo de Slow Start, puesto que la razón de que se reciban ACK duplicados no solo es que se pierdan paquetes, sino que también los paquetes posiblemente está dejando la red. Junto con esto, el reloj de recepción de ACK se mantiene, y el

emisor puede seguir transmitiendo nuevos paquetes, eso sí, la transmisión debe continuar con una ventana de congestión reducida, ya que perdidas es una indicador de congestión.

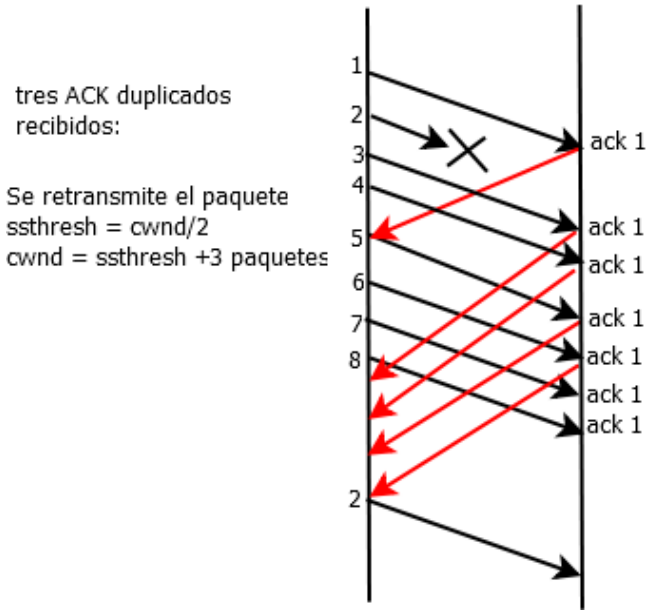

Figura 4: Funcionamiento de algoritmo Fast Restransmit y Fast Recovery.

2.2.2.4 Limitación del Throughput TCP

Throughput es la tasa promedio de la entrega de paquetes de información en un canal de comunicación. Esta información puede ser transmitida por un medio físico o lógico, o puede pasar a través de un nodo de la red.

Una aproximación del throughput teórico que corresponde a una transmisión TCP.

$$Throughput = \min(\, rwnd \,, \frac{MSS}{RTT\sqrt{\frac{2p}{3}}})$$

En donde rwnd es la ventana de congestión del receptor, MSS es el máximo tamaño de un segmento para el receptor y el emisor y p la perdida paquetes por unidad. En el caso en que no haya pérdidas, el segundo término tiende a infinito, por lo que el valor es de la ventana de congestión del receptor.

En redes de alta latencia la probabilidad de tener pérdidas es bastante alta, por lo que una aproximación de la fórmula del throughput vendría a ser:

$$Throughput = \frac{MSS}{RTT\sqrt{\frac{2p}{3}}}$$

Esto nos entrega una relación entre el throughput y el RTT de proporcionalidad inversa, la que ayudara en la búsqueda de aumentar el throughput si se disminuye el RTT con la inclusión de un nodo proxy.

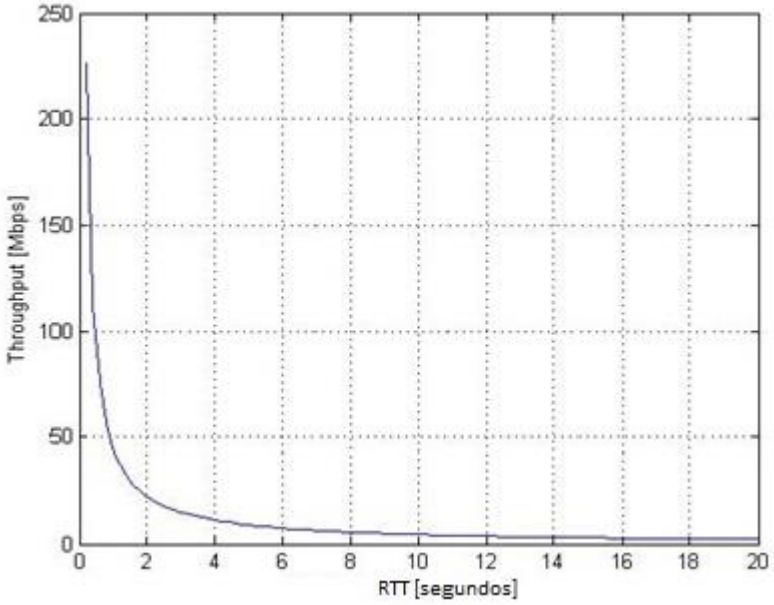

Figura 5: Relación de proporcionalidad inversa entre el Throughput y el RTT.

2.2.3 TCP BIC

El protocolo TCP BIC o Control de Congestión mediante Incremento Binario (*Binary Increase Congestion Control)*, nace como una respuesta a los problemas que presentaba TCP Reno, al momento de trabajar con redes de alta latencia, el mayor de ellos era la injusticia (*fairness problem)*, esto porque la tasa de crecimiento de la ventana se agrandaba cuando la ventana crecía, irónicamente, es la misma razón que los hace más escalables.

Para poder solucionar este problema, el protocolo BIC presenta un algoritmo para la ventana de congestión diferente a TCP Reno, el cual se enfoca en encontrar el máximo para la CWND y mantenerlo por un periodo de tiempo usando un algoritmo de búsqueda binario, como se muestra en la figura 7.

BIC está implementado y es usado en los kernel de Linux 2.6.18 y superior, hasta la versión 2.6.19.

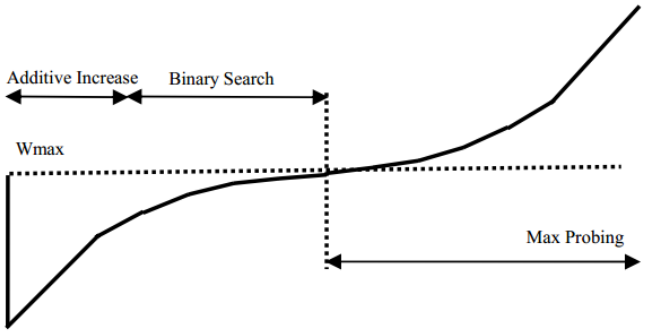

Figura 6: Función de crecimiento de TCP BIC

2.2.4 TCP CUBIC

El protocolo de control de congestión TCP CUBIC (*CUBIC*), nace con el mismo propósito de TCP BIC, solucionar los problemas en redes de alta latencia y como respuesta al protocolo TCP BIC, esto porque el algoritmo multiplicativo usado por BIC en el cálculo de la ventana de congestión era muy agresivo para TCP, sobre todo usando RTT bajos o en redes de baja velocidad.

El algoritmo usado para el crecimiento de la ventana de congestión tiene una forma cúbica, la que se puede apreciar en la figura 8, este es más lento que el usado por BIC que se incrementaba de forma binaria o logarítmica cercana al origen.

CUBIC al igual que BIC toman todas las ventajas de TCP y le añaden mejoras en lo que a justicia "fairness" con el RTT se refiere, escalabilidad, convergencia y amistad "friendliness" con TCP.

CUBIC se utiliza en las versiones 2.6.19 del Nucleo (*Kernel*) de Linux.

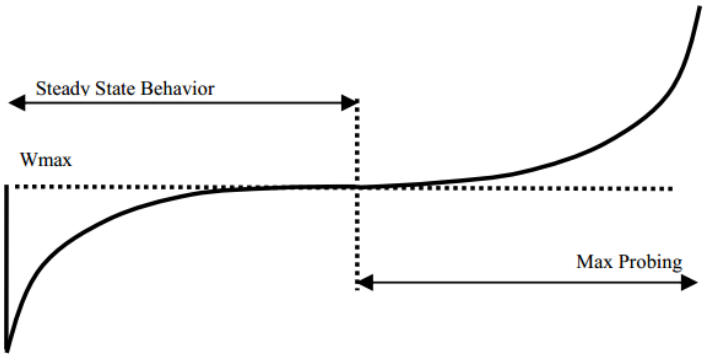

Figura 7: Función de crecimiento de TCP CUBIC

2.3 Trabajo Previo

2.3.1 Modelado de TCP

Actualmente el modelamiento de TCP tiene como principal factor de observación el algoritmo para prevenir la congestión (Congestion Avoidance Algorithm) y el impacto que tiene en el throughput, tomando en consideración la dependencia que genera la prevención de congestión sobre el comportamiento de los ACK. De cualquier forma que se infiera que el paquete se pierda, por ejemplo por ACK duplicado o Fast Retransmit; por que se acabe el tiempo, o por una ventana de congestión del receptor limitada en tamaño y/o el RTT promedio.

El trabajo de investigación incluye una sección de modelamiento de ESTP, donde se explicará con mayor detalle la obtención de cada fórmula publicada, que dicho sea de paso, proviene del mismo planteamiento con algunas diferencias producto que son algoritmos distintos.

El modelo TCP se concentra en la forma de evitar la congestión, donde el tamaño de la ventana de congestión es W y se incremente por $1/W$ cada vez que un ACK es recibido. Por el contrario la ventana disminuye cuando se detecta que un paquete se ha perdido, cuando disminuye dependerá de si la perdida es por ACK duplicado o se acaba el tiempo.

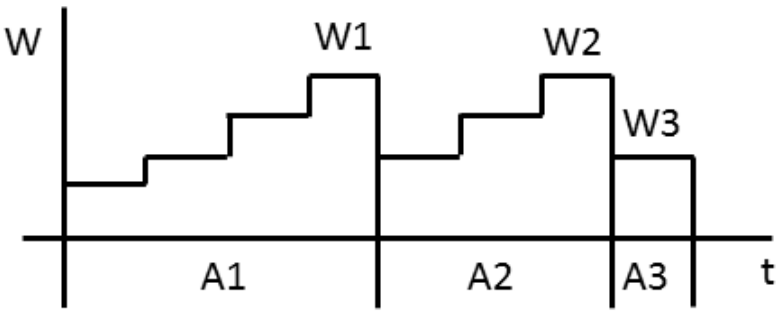

Figura 8: Variaciones del tamaño de la ventana de congestión en el tiempo, cuando solo hay triple ACK duplicados como pérdida.

El modelo para evitar congestión se divide por turnos. El primer turno comienza con la transmisión completa de W paquetes, donde W es el tamaño actual de la ventana de congestión de TCP. Una vez que todos los paquetes han sido transmitidos, no se transmiten más paquetes hasta que el primer ACK es recibido, la recepción del primer ACK marca el fin del primer turno y el comienzo del segundo. En el modelo presentado en [7] el tiempo de duración de un turno es igual al RTT y se asume independiente de la ventana de congestión. En el segundo turno se envían W' paquetes, esto pues la ventana de congestión se ha incrementado. Ahora, supongamos que son sólo b los paquetes que se reciben por un ACK. La mayoría de las implementaciones de TCP hacen que $b = 2$ si en el primer turno se enviaron W paquetes y se recibieron todos los ACK correspondientes, entonces se habrán recibido W/b . Como cada ACK aumenta la ventana de congestión en $1/W$ entonces la ventana de congestión al comienzo del segundo turno es $W' = W + 1/b$. Esto durante la evasión de congestión y sin pérdidas, con esto la ventana de congestión aumenta de forma lineal en el tiempo de la forma $1/b$ paquetes por RTT.

Ahora, el modelo se concentra en la relación entre los distintos tipos de pérdidas y sus consecuencias sobre el throughput de TCP. $T(p)$ es el throughput de TCP dependiendo de la probabilidad de perdida de paquete p. El cálculo de cada subsección será visto con mayor detalle en el capítulo 3 de este documento.

2.3.1.1 Perdidas exclusivas por triple ACK duplicados

Las pérdidas por triple ACK duplicados, ocurren cuando el emisor recibe 3 veces el mismo paquete ACK, lo que le indica al sistema que ha ocurrido una perdida. Esta taza afecta el throughput de la siguiente forma:

$$T(p) = \frac{1}{RTT}\sqrt{\frac{3}{2bp}} + o\left(1/\sqrt{p}\right) \qquad (1)$$

Donde RTT es el tiempo de ida y vuelta , b es el número máximo de paquetes confirmados (ACK recibidos) y p es la probabilidad de perdida por paquete. El termino de o chica se puede despreciar para valores grandes de p.

2.3.1.2 Perdidas por triple ACK duplicados y por tiempo de espera.

Cuando añadimos perdidas por paquetes que se demoran mucho en llegar a su destino o no pueden llegar por algún motivo, la formula (1) cambia a:

$$T(p) \approx \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 \cdot \min\left(1{,}3\sqrt{\frac{3bp}{8}})p(1+32p^2)\right)}$$

2.3.1.3 El impacto de una ventana limitada.

Finalmente si se tiene una ventana limitada en tamaño, digamos W_{max}, la fórmula para el throughput estaría dado por:

$$T(p) \approx \min\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 \cdot \min\left(1{,}3\sqrt{\frac{3bp}{8}}\right)p(1+32p^2)}\right)$$

2.3.2 TCP Proxy - Spoofing

Actualmente se trabaja con distintas formas de mejorar el rendimiento de TCP en redes de alta latencia, ya sea en redes físicas, *wireless* o satelitales. El mayor problema que presentan es que el protocolo TCP debe mantenerse en los extremos de la red sin ser modificado, es más, en algunos sistemas operativos los usuarios no pueden controlar con que variante de TCP utilizar en sus conexiones. Es por esto que se desarrollaron ideas para combatir los problemas de los Proxy tradicionales, que consiste en que el emisor envía un paquete, este pasa por un nodo proxy intermedio, el que se encarga de crear una copia de ACK y retransmitirla al emisor inmediatamente. Además crea una copia del paquete que acaba de llegar y la guarda en una cola (queue), esto en caso de que se llegase a perder el paquete entre el proxy y el receptor, para poder retransmitirlo. Luego transmite el paquete al receptor. El receptor recibe el paquete y envía el ACK original, que debe pasar por el nodo proxy, el que se encargara de destruirlo, para evitar problemas de múltiples retransmisiones. Esto queda visto en la figura 8.

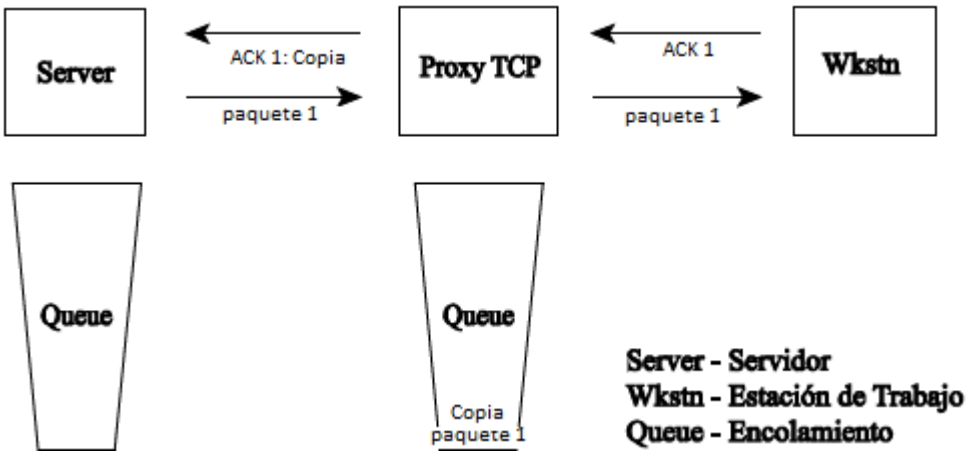

Figura 9: TCP Spoofing sobre un nodo Proxy.

TCP Spoofing, es una solución que ha tenido bastante aceptación en la comunidad científica para resolver el problema de disminución de rendimiento que afecta las redes de alto RTT. Ésta consiste en acelerar el crecimiento de la ventana de transmisión de TCP ocultando el retraso producido por la distancia entre nodos de la red, como bien se muestra en la figura 9, además de generar las copias de ACK en el nodo proxy, así reduciendo el RTT total a la distancia al nodo desde el emisor. El principal problema que este método posee es que la semántica extremo a extremo (end-to-end) no se cumple, creando problemas e ineficiencias en algunas aplicaciones que se basan en esta característica. Como se mencionó antes, el nodo proxy será encargado de la eliminación de los ACK originales para evitar problemas de ACK duplicados, así también en el caso de una pérdida es el nodo quién debe tener un sistema de colas (queues) donde tiene una copia de los paquetes enviados.

Los problemas que se acaban de mencionar deben ser manejados para que la red pueda seguir soportando conexiones TCP de buena manera. La solución a estos, están implementadas en el capítulo 3, puesto que es independiente del protocolo de transmisión que se utilice tanto como ESTP o TCP en cualquiera de sus versiones.

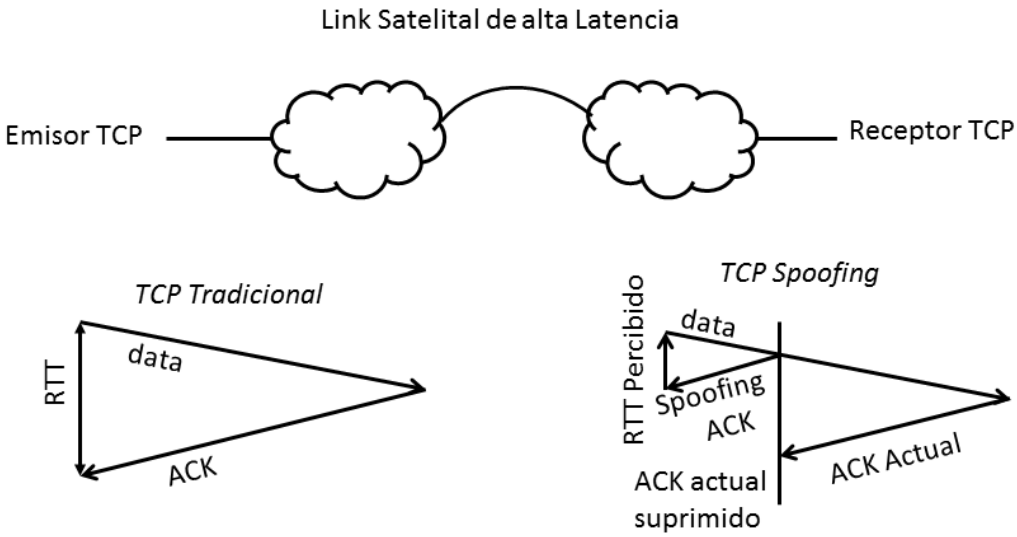

Figura 10: Aplicación sobre una red satelital de TCP Spoofing.

2.3.3 Ethernet-Services Transport Protocol (ESTP)

El protocolo de transporte de servicios Ethernet, nace como una medida para disminuir los efectos de los algoritmos de reducción que a su vez reduce los efectos de congestión ajustando dinámicamente el tamaño de la ventana de congestión (CWND) y por lo tanto la tasa de transmisión, usando como retroalimentación el nivel estimado de congestión de la red. Una de las características importantes de ESTP es que consigue esto usando solo información que se encuentra disponible en la capa de transporte, y no requiere nada adicional, haciendo que este proceso de retroalimentación sea transparente para la red.

ESTP se puede resumir en tres partes:

- Estimación del nivel de congestión.
- Función de mapeo.
- Algoritmo de control para evitar la congestión.

2.3.3.1 Mecanismo de estimación del nivel de congestión

Esta es una de las características más importantes de ESTP y no debe ser confundida con detección de congestión. El nivel de congestión está relacionado a la cantidad de paquetes que fueron transmitidos exitosamente entre dos pérdidas de paquetes. La cantidad instantánea de paquetes transmitidos entre dos pérdidas se utilizan como argumento de una función de mapeo exponencial, que permite determinar un factor de disminución multiplicativo menos agresivo. Esta función se define como $1/_{map(\alpha)}$. El propósito de elegir un perfil exponencial es de igualar la distribución de probabilidad mostrada con la distancia entre las dos pérdidas de paquete. Al tener la ventana de congestión dependiendo en el nivel de congestión de la red, el tamaño de la ventana de congestión es más maleable y se puede controlar de manera más eficiente.

α es la cantidad de paquetes que transmitidos correctamente entre las dos pérdidas de paquetes más uno, con esto el rango de α va de 1 a ∞, haciendo que los casos de borde sean dos pérdidas consecutivas y cuando no hay perdidas. Dado este rango $map(\alpha)$ varía en el rango de 2 a 1. Lo que implica que ESTP puede dividir la ventana de congestión por un factor no más grande que 2 y no más pequeño que 1.

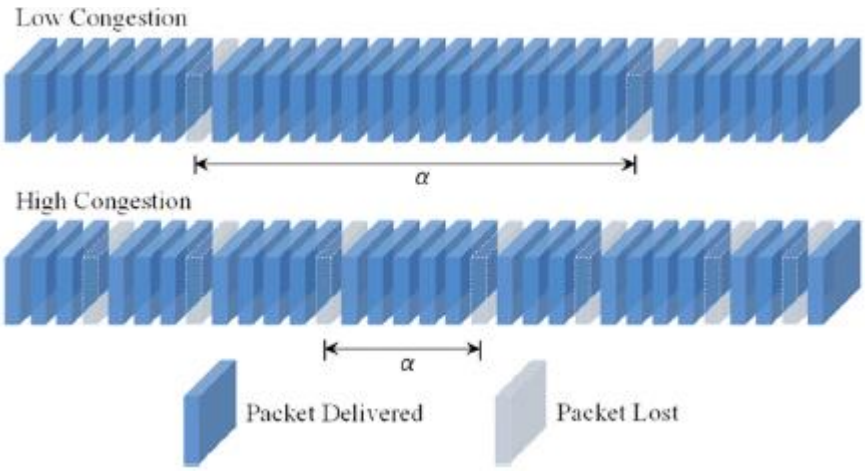

Figura 11: El valor de α es la cantidad de paquetes transmitidos entre dos pérdidas.

2.3.3.2 Función de Mapeo

La función de mapeo de ESTP se modela como la probabilidad de un evento de 2 casos, donde la pérdida de un paquete tiene una distribución de Bernoulli, donde la pérdida es un evento correcto. El número de eventos correctos en un periodo de tiempo fijo, se distribuye mediante una Poisson. Con estos datos podemos definir una función de probabilidad de una distribución exponencial como:

$$f(\alpha;\lambda) = \begin{cases} \frac{1}{\lambda} e^{-\frac{\alpha}{\lambda}} & \alpha \geq 0 \\ 0 & \alpha < 0 \end{cases}$$

Donde λ es la Esperanza de α.

Con esta función de distribución se construye la función de mapeo utilizando las condiciones de borde, haciendo tender α a 0 y α a∞.
Luego la función map() está dada por:

$$map(\alpha) = e^{-\frac{\alpha-1}{\tau}} + 1$$

Y la expresión para la ventana de congestión queda:

$$cwnd_{n+1} = cwnd_n \left(e^{-\frac{\alpha-1}{\tau}} + 1 \right)^{-1}$$

2.3.3.3 Control para evitar la congestión

ESTP ya muestra mejoras respecto a al rendimiento throughput con las técnicas anteriores y de manera independiente una de otra. Al combinar los métodos anteriores con los mecanismos del acuerdo del nivel de servicio, esta técnica aprovecha el bandwidth provisto por el subscriptor, consigiendo que el throughput se mantenga por arriba del CIR. Luego la $cwnd_{MIN} = RTT \cdot CIR$ y el $cwnd_{MAX} = RTT \cdot EIR$ el cota superior previene que se exceda del EIR y, por lo tanto, se generan pérdidas de paquete debido a políticas de QoS de tráfico. Luego la ventana de congestión queda definida por la siguiente expresión:

$$cwnd_{n+1} = min\left(cwnd_{MAX}, cwnd_n + \frac{1}{cwnd_n}\right)$$

Luego se define el TCP tradicional como un caso especial cuando $cwnd_{MAX} = \infty$. De esta forma se puede ver que ESTP logra mantener de buena manera la ventana de congestión dentro de sus límites y con rendimiento superior al TCP tradicional.

Capitulo 3:
TRABAJO REALIZADO

3.1 Introducción

En este capítulo se desarrollan los temas trabajados, estos son el desarrollo teórico del modelo throughput de ESTP, este desarrollo es bien teórico y va desde la elección del canal del modelo hasta la derivación de la misma fórmula. A continuación se detalla el modelo que mezcla ESTP con un TCP Proxy y el cambio respecto al modelo obtenido en el punto anterior. Posterior a eso se discute acerca de la elección del modelo y de su proceso de construcción y desarrollo, el que se dividió en dos, primero un modelo ideal sin perdidas y luego el modelo con pérdidas en distintos puntos del trayecto emisor-receptor. Finalmente se muestra la lógica de cómo se programó en lenguaje C los distintos estados que sirven para representar el proxy y así generar los datos de simulaciones cuyos resultados se verán en el Capitulo 4.

3.2 Desarrollo teórico del modelo throughput de ESTP

Para poder derivar una fórmula analítica del throughput de ESTP, es necesario definir el canal del modelo, que fue explicado anteriormente en la sección 2.3.3.2 como función de mapeo. Este modelo tiene dos ramas que son capaces de discriminar el throughput cuando el tráfico es igual o menor al CIR (siendo ruteado al camino pathp1) o cuando el tráfico excede el CIR (que es ruteado a través del camino pathp2, donde el subíndice indica la tasa de pérdida sujeta a cada camino). Este modelo del canal es el comienzo para modelar el throughput en estado estacionario de ESTP.

Debe dejarse en claro que la tasa de tráfico que está por debajo del CIR sea garantizada y las únicas pérdidas que pueden ocurrir en este escenario son las pérdidas intrínsecas del sistema. Como ESTP calcula la CWND_MIN para mantener el throughput por arriba del CIR, el throughput de estado estacionario es acotado inferiormente por el CIR. Usando la teoría de renovación planteada en [8][9] se puede obtener un modelo más exacto. Como las redes de Carrier Ethernet corren servicios Ethernet que proveen QoS, solo las pérdidas por triple ACK duplicado son consideradas. Luego el throughput de estado estacionario se define como:

$$T = \lim_{t\to\infty} \frac{N_t}{t}$$

N_t es el número de paquetes enviados por intervalos [0,t]. Si extendemos esta definición con la teoría de renovación, donde la evolución de la ventana de congestión es analizada sobre una base por ciclos, la expresión se convierte en:

$$T = \frac{E[Y]}{E[A]}$$

Y_i es el número de paquetes enviados en el ciclo i durante el tiempo A_i. El periodo de renovación es elegido como el periodo de un triple ACK duplicado (TDAP), es decir como el intervalo entre dos pérdidas de paquetes (ver figura 11). Esta es una opción conveniente, por qué la ventana de

congestión crece consistentemente en este intervalo. W_i está definido como la ventana de congestión al final del ciclo y la $CWND_{MIN}$ es la cota inferior del tamaño de la ventana de congestión. Para calcular el valor promedio de Y_i , es decir $E[Y]$, el número de paquetes transmitidos exitosamente es sumado al número de paquetes en vuelo al momento de la pérdida, resultando:

$$E[Y] = E[\psi - 1] + E[W]$$

$$E[Y] = E[\psi] + E[W] - 1$$

El final de los paquetes en vuelo simboliza el final del ciclo. Como detectar la pérdida toma un RTT completo, el tamaño de paquetes en vuelo es W_i, como se describe en la formula anterior y se muestra en la figura 13.

Se asume que una pérdida de paquete está modelada como una variable aleatoria, independiente y que se distribuye uniformemente. La probabilidad de que el paquete ψ-esimo se pierda, es igual a la probabilidad que $\psi - 1$ paquetes hayan sido transmitidos correctamente y que uno se pierda, así la probabilidad queda descrita como:

$$P[\psi = k] = (1-p)^{k-1}p, \;\; (k > 0) \cap (k \in \mathbb{Z})$$

Luego el promedio es:

$$E[\psi] = \sum_{k=1}^{\infty} (1-p)^{k-1} pk = \frac{1}{p}$$

Luego juntando estas dos ecuaciones se obtiene:

$$E[Y] = \frac{1}{p} + E[W] - 1 = E[W] + \frac{1-p}{p} \quad (\Pi)$$

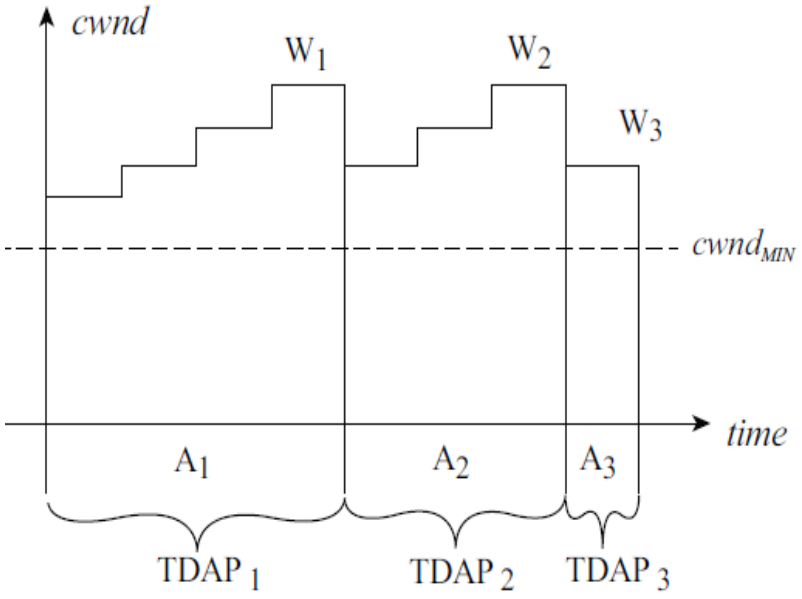

Figura 12: Ejemplo de un periodo de triple ACK duplicado.

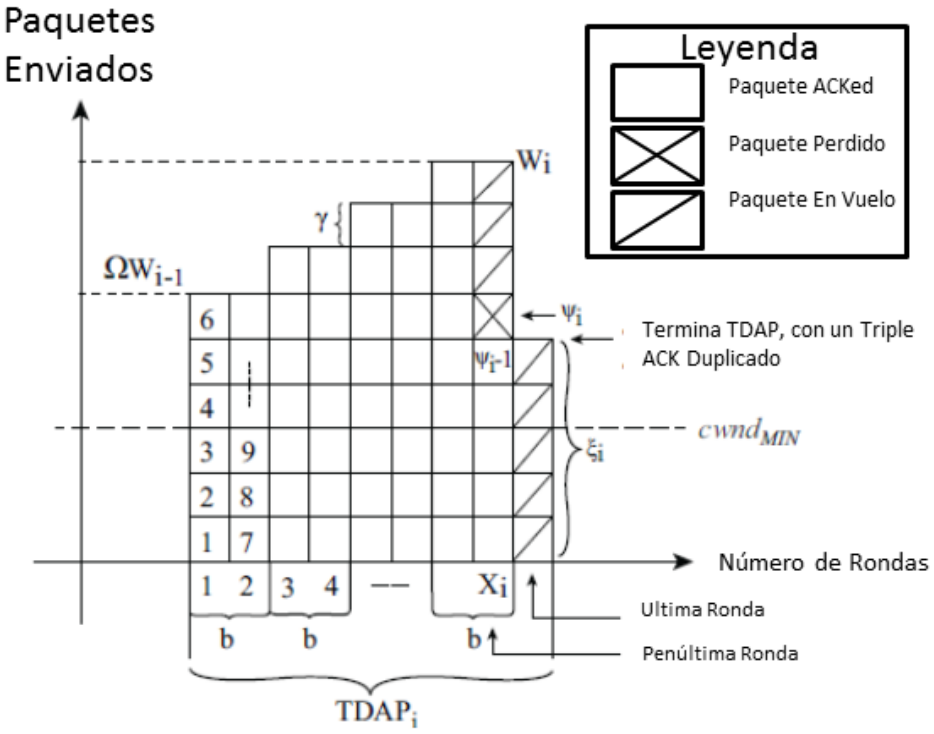

Figura 13: Paquetes enviados durante un TDAP.

Calcular el tiempo promedio de TDAP dado $E[X]$ es directo, asumiendo que el tiempo de ida y vuelta (RTT) es conocido. X_i es la ida y vuelta en donde la pérdida ocurre, por lo tanto $X_i + 1$ es el cantidad total de idas y vueltas en $TDAP_i$, por lo que:

$$E[A] = (E[X] + 1)RTT$$

La relación entre E[X] y E[W] procede del patrón de crecimiento de W_i dependiendo de X_i, que es:

$$W_i = (W_{i-1} + cwnd_{MIN})\Omega + \frac{\Upsilon}{b} X_i$$

De la ecuación anterior se obtiene:

$$E[X] = \frac{b}{\gamma}(E[W](1 + \Omega) - cwnd_{MIN}\Omega)$$

El número de paquetes Y_i transmitidos en $TDAP_i$, puede ser expresado como:

$$Y_i = \sum_{k=0}^{\frac{\gamma x_i}{b}-1} (\Omega W_{i-1} + k)\frac{b}{\Upsilon} + \xi_i$$

$$= \Omega W_{i-1} X_i + \frac{X_i}{2}\left(\frac{\gamma}{b} X_i - 1\right) + \xi_i$$

$$= \frac{X_i}{2}(2\Omega W_{i-1} + \frac{\gamma}{b} X_i - 1) + \xi_i$$

$$= \frac{X_i}{2}(\Omega W_{i-1} + W_i - \Omega CWND_{MIN} - 1) + \xi_i \quad (\Psi)$$

El término ξ_i es la cantidad de paquetes en la ida y vuelta X_{i+1} que permanecen en vuelo, tras la pérdida ocurrida en X_i. Además ξ_i depende de Ω, que a su vez depende de α. La distribución de α debe ser considerada en los cálculos. El resultado puede aproximarse asumiendo que ξ_i se distribuye como una normal entre W_i y $CWND_{MIN}$, por lo tanto $E[\xi] = (CWND_{MIN} + E[W])/2$. Si este resultado se combina con (Π) y (Ψ), se obtiene la siguiente expresión:

$$E[W] + \frac{1-p}{p} = \frac{E[X]}{2}(E[W](\Omega + 1) - \Omega CWND_{MIN} - 1) + \frac{1}{2}(CWND_{MIN} + E[W])$$

Utilizando esta última ecuación y la relación de E[X], se obtiene:

$$E[W] \cong \frac{\Omega CWND_{MIN}}{1-\Omega^2} + \sqrt{\left[\frac{\Omega^2 CWND_{MIN}}{1-\Omega^2}\right]^2 + \frac{2\gamma}{b(1-\Omega^2)p}}$$

Como la pérdida es mucho menor que 1, los términos de orden menor pueden ser despreciados para simplificar el problema. Obteniendo la igualdad de E[W] es importante, ya que ambos E[Y] y E[A] dependen de ella. Finalmente el throughput es calculado con $E[Y]/E[A]$.

$$T = \frac{E[Y]}{E[A]} \cong \frac{1}{RTT\left(-k + \sqrt{k^2 + \frac{2b\ p(1-\Omega)}{\gamma(1+\Omega)}}\right)}$$

$$k = \frac{cwnd_{MIN}\Omega^2 b\ p}{\gamma(1+\Omega)}$$

Esta expresión queda en términos del incremento aditivo γ y del decrecimiento multiplicativo Ω. RTT es el tiempo de ida y vuelta, b es el número máximo de paquetes de los cuales se recibió su ACK respectivo y p es la tasa de pérdida de paquetes. Esta expresión tiene por unidades [paquetes/segundos], siendo necesario multiplcarla por el tamaño máximo de un segmento (MSS, *Maximum Segment Size*) para convertirla en [bits/segundos], lo que determina el número de bits por paquetes. Esta expresión asume un valor estático de Ω, pero ESTP lo varía dinámicamente usando el feedbACK del nivel de congestión de la red. Para ser más precisos, Ω es reemplazado por Ω_α, el cual depende de α, definido como la distancia entre dos pérdidas de paquete.

$$\Omega_\alpha = \frac{1}{map(\alpha)} = \left(e^{-\frac{\alpha-1}{\tau}} + 1\right)^{-1}$$

Por lo tanto, el throughput de estado estacionario, que depende de α, se obtiene al resolver el sistema con las últimas dos ecuaciones, sustituyendo Ω por Ω_α.

$$T_\alpha = \frac{1}{RTT\left(-k + \sqrt{k^2 + \frac{2b\ p\left(1 - \left(e^{-\frac{\alpha-1}{\tau}} + 1\right)^{-1}\right)}{\gamma\left(1 + \left(e^{-\frac{\alpha-1}{\tau}} + 1\right)^{-1}\right)}}\right)}$$

$$k = \frac{cwnd_{MIN} b\ p}{\gamma\left(1 + \left(e^{-\frac{\alpha-1}{\tau}} + 1\right)^{-1}\right)\left(e^{-\frac{\alpha-1}{\tau}} + 1\right)^2}$$

Esto nos lleva a una expresión cerrada. Como α es una variable aleatoria con distribución exponencial, es necesario determinar la esperanza del T_α.

$$E[T_\alpha] = \int_0^\infty T_\alpha\, f(\alpha;\lambda) d\alpha$$

Dado que el throughput de ESTP está acotado superiormente por la Taza de Exceso de Información (EIR, *Excess Information Rate*) y como la $cwnd$ no puede superar a la $cwnd_{MAX}$, la expresión final es:

$$T_{ESTP} = min(EIR, E[T_\alpha])$$

La expresión final depende del valor de α. En ambos casos $\lambda = E[\alpha]$, si se utiliza el tiempo $E[\alpha] = E[A]$ y si se utilizan los paquetes $E[\alpha] = E[Y]$. Estos casos nos llevarán a throughputs similares, pero con algunas diferencias, como que los paquetes lleguen en ráfagas o tengan periodos donde no llegan paquetes, lo que hace que estas métricas no sean paralelas, es decir, la relación $(Y_{i+1} - Y_i)/(A_{i+1} - A_i)$ no sea igual para todos los ciclos.

3.2.1 Modelo del Throughput de ESTP con TCP Proxy

Al ser el proxy un punto intermedio cualquiera, entre emisor y receptor, existen 2 RTT, el primero entre emisor-proxy y el segundo el proxy-receptor, se debe considerar el máximo entre estos RTT para calcular la fórmula del throughput.

$$T = \frac{1}{Max(RTT_1, RTT_2)\left(-k + \sqrt{k^2 + \frac{2b\,p(1-\Omega)}{\gamma(1+\Omega)}}\right)}$$

$$k = \frac{cwnd_{MIN}\Omega^2 b\,p}{\gamma(1+\Omega)}$$

En esta ocasión RTT_1 es el tiempo de ida y vuelta entre emisor y Proxy, así RTT_2 es el tiempo de ida y vuelta entre Proxy y receptor. La variable b es la probabilidad de perdida por paquete, mientras que Ω es el factor de decrecimiento multiplicativo y γ el incremento aditivo.

3.3 Diseño de un Nodo Proxy TCP.

Durante el proceso de diseño de este nodo se analizaron distintas herramientas para que la programación, el análisis y las simulaciones se pudieran realizar de forma amigable.

La siguiente etapa consistió en elegir un modelo sobre el cual trabajar evaluando ventajas y desventajas frente a sus pares. Una vez definido, se procede a diferenciar el modelo sin pérdidas del que sí las posee, sobre este último se tuvo que trabajar de manera adicional para poder manejar dichas pérdidas al igual que lo hace TCP tradicional sin proxies, para esto se usó el principio de "dividir para reinar", por lo que primero la preocupación fue controlar las perdidas entre el servidor y el proxy, para luego solucionar el problema de una perdida entre proxy y cliente.

3.3.1 Selección del Modelo

Para poder programar el sistema, primero se necesita una representación sobre la cual basarse, esto para no ir a ciegas y perder el tiempo. Lo primero que se hizo fue seleccionar un modelo que se adaptase al experimento que se quiere llevar a cabo, para esto se necesitaba al menos un servidor y un cliente, los cuales pudiesen interactuar sin problemas enviando y recibiendo paquetes de distinto tamaño. Luego se coloca un proxy entre ellos, que sirva de "peaje" y que los

paquetes tengan que pasar por él. Así se determinó que el modelo a utilizar de Servidor/Proxy/Cliente con una relación de 1:1:1, era el modelo que mejor se adecuaba a los requerimientos de la investigación, por sobre modelos con multi servidor, multi cliente o multi proxy, siendo este último un caso redundante del seleccionado. En la figura 14, se aprecia el modelo.

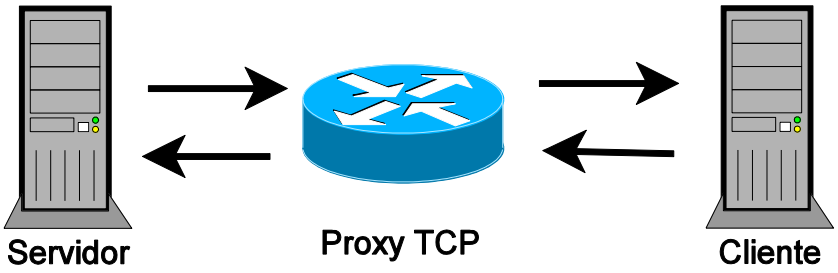

Figura 14: Modelo Servidor-Proxy TCP-Cliente.

Adicionalmente este modelo es sugerido para implementar el TCP spoofing como se explica en el punto 2.3.2, esto porque en el proxy se puede colocar una cola "queue" con la que es fácil controlar el envió y la recepción de paquetes. En la figura 15 y la figura 16 se puede ver una representación del envío de un paquete y como se ve afectado su RTT.

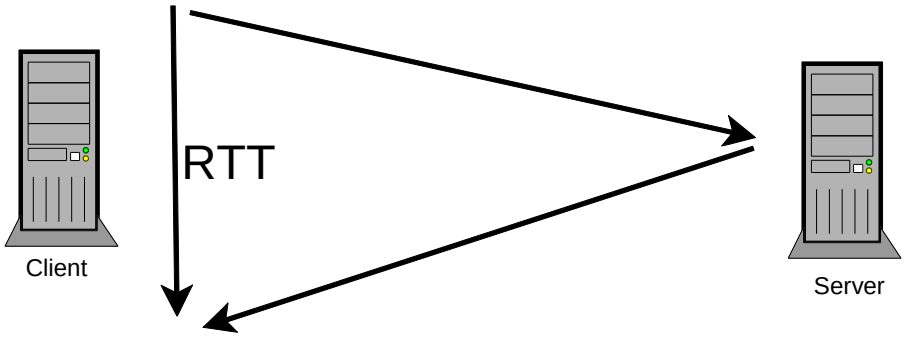

Figura 15: Modelo Servidor/Cliente sin Proxy

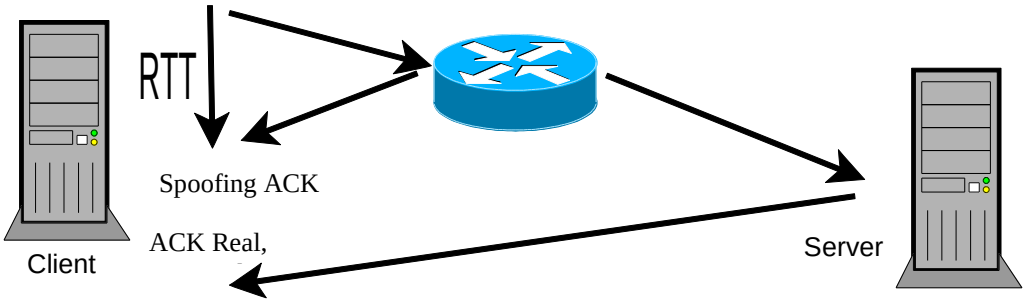

Figura 16: Modelo Servidor/Cliente con Proxy

Como se puede ver en la figura 16, se envía el paquete de información del servidor al cliente, este debe pasar por el proxy, y una vez sea recibido el paquete de información, generara una copia del paquete ACK (*Spoofing* ACK), que crea el servidor cuando recibe información. El ACK original será suprimido, por el mismo proxy, por lo que es fundamental que pase por el proxy, ya que en caso contrario pueden ocurrir situaciones donde se recibe 2 veces el mismo ACK por parte del cliente.

3.3.2 Modelo sin Pérdida

Como se vio en el punto 3.3.1 el primer modelo en el que se trabajó asume que no hay pérdidas, por lo que presenta un funcionamiento normal que consiste en que el servidor envía un paquete con información en dirección al cliente y al pasar por el proxy, crea una copia del paquete con la información y la guarda en la cabeza de una cola de paquetes. Además, hace una copia del paquete ACK de dicho paquete de información y la transmite de vuelta al servidor. Al mismo tiempo transmite el paquete original con información al cliente, disminuyendo el tiempo de ida y vuelta (RTT, *Round Trip Time*) de los paquetes. Una vez que el paquete llega al cliente, envía un paquete ACK correspondiente al paquete con información que acaba de recibir. Este paquete ACK es interceptado por el proxy y chequea si el número de secuencia (SEQ) del paquete que está en la cabeza de la cola es igual al número ACK, se borra la copia almacenada en la cola Q y se destruye el paquete ACK que se acababa de recibir. Esto se puede apreciar de forma más clara en la figura 17.

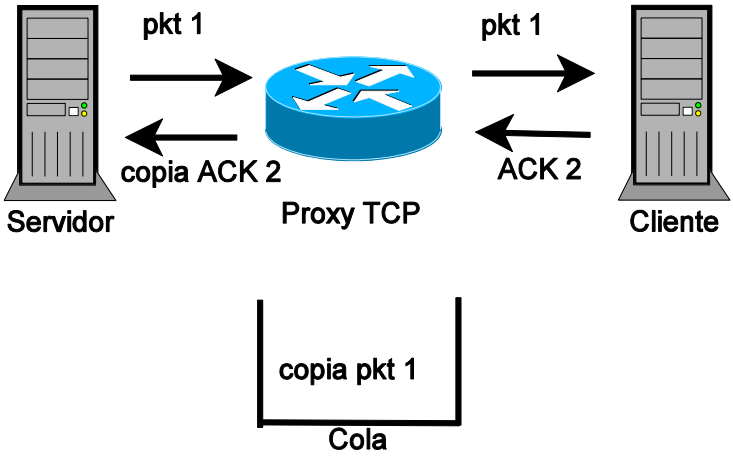

Figura 17: Modelo sin perdidas

3.3.3 Modelo con pérdidas

Presenta la misma estructura que el modelo sin pérdidas, dando el espacio para que ocurran perdidas, estas pueden ser en el camino entre servidor y proxy, entre el proxy y el cliente y del cliente al proxy. El caso entre proxy y servidor puede ocurrir, pero no necesita un trato especial, ya que el proxy pasado un tiempo volverá a enviar el paquete ACK perdido y se mantendrá esperando al siguiente paquete de información enviado por el servidor.

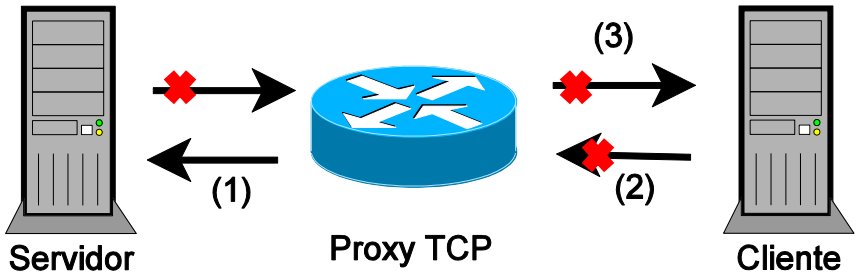

Figura 18: Modelo con perdida.

.En la figura 18, el punto (1) representa la uBICación donde puede ocurrir la perdida entre Servidor y Proxy. (2) la uBICación donde puede ocurrir la pérdida entre Cliente y Proxy. Finalmente el punto (3) representa la perdida entre Proxy y Cliente.

3.4 **Programación del Proxy TCP.**

La programación se llevó a cabo en lenguaje C, una vez definido el modelo, se programó por etapas, estos se encuentran detallados a continuación:

Primero se programó una perdida entre el servidor y el proxy, para ver el compartimiento de TCP frente a una perdida. Esto resulta importante, debido a que en los pasos siguientes se debía programar una copia exacta del paquete ACK que enviaba el cliente como respuesta de recepción. Para esto se rescataron las siguientes variables desde el paquete enviado:

Tabla 3: Variables importantes de un paquete ACK.

Variable	Utilidad
time	Tiempo en que se envió el paquete.
src_port	Puerto de procedencia del paquete.
dest_port	Puerto de destino del paquete.
seq_num	Número de identificación secuencial.
ACK_num	Número de acuso de recibo.
rcv_win	Tamaño de la ventana de recepción.
urgent_pointer	Variable no utilizada
data_len	Tamaño de la información.
flags	Identificador del paquete.

Posterior a esto, se pasó a crear un "relay", el cual solo se preocuparía de copiar el paquete y crear un archivo que comparara todas las variables.

3.4.1 **Algoritmo Leaky Bucket.**

Una vez completado esto, se procedió a mejorar el "relay", utilizando el algoritmo de "Leaky Bucket" o algoritmo de contador dinámico. Este algoritmo[4] consiste en una cola finita. Cuando un paquete llega, si hay espacio para otro paquete en la cola, este es añadido a la cola, sino se descarta. La modificación que se le hizo para este trabajo, fue darle un tamaño infinito a la cola, para mayor simpleza al momento de trabajar y mayor eficacia en la detección de errores.

[4] Definición de Andrew S. Tanenbaum en su libro "Computer Networks".

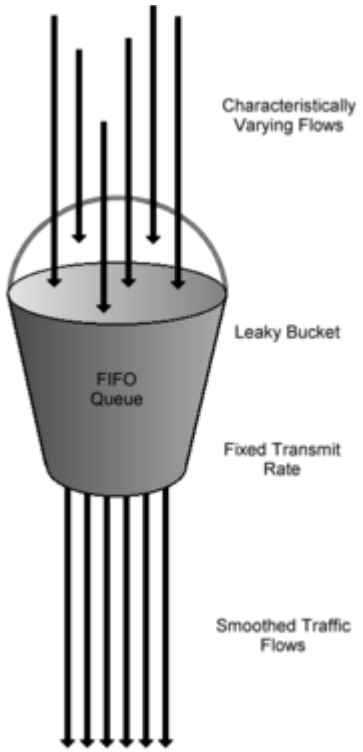

Figura 19: Leaky Bucket utilizado en el modelo.

A continuación se añadió la funcionalidad para copiar el paquete ACK, en el proxy. El algoritmo implementado para devolución de paquetes fue "*leaky bucket*" con cola infinita. Y como se puede observar en la figura 19, se guarda una copia del paquete de información y se reenvía el mismo a una taza de un paquete a la vez. Las copias de los paquetes de información son borradas cuando el paquete ACK enviado por el cliente es recibido por el proxy. La condición para saber si es el paquete correcto o no, es mediante su número SEQ el cual se compara con el número ACK y de ser iguales se borra el paquete de información alojado en la cola.
Una vez funcionando el modelo sin perdidas, se procede a avanzar al con pérdidas. La primera pérdida se coloca entre el servidor y el proxy. Esto fue decidido así, porque al solucionar esta situación primero se asegura que los paquetes dentro de la cola se encuentren en orden, lo que ayuda al correcto funcionamiento en caso de haber una perdida entre el proxy y el cliente, ya que solo hay que retransmitir la cabeza de la cola. En la figura 20, se muestra gráficamente como fue programado el relay con perdida, la decisión de modularizar la perdida se tomó para poder moverla en cualquier parte del modelo.

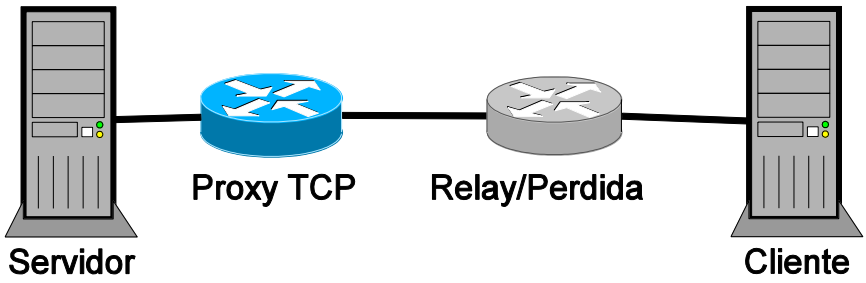

Figura 20: Módulos de Proxy TCP y Relay separados gráficamente.

El manejo detallado de las perdidas entre servidor-proxy y proxy-servidor a continuación en los puntos 3.4.2 y 3.4.3.

3.4.2 Manejo de pérdidas servidor/proxy

Para poder controlar las pérdidas entre el servidor y el proxy se creó una cola adicional Q_d, donde se almacenaban las copias de paquetes, una vez fuera detectada la primera pérdida. Luego de transmitir que había una pérdida, se chequea que la cabeza de la cola no esté vacía, de no estarlo, se revisa si el número de secuencia es el que se está buscando. En caso de serlo, se borra de la cabeza de la cola Q_d y se añade al final de la cola Q. En caso que el elemento en la cabeza de la cola Q_d sea el inmediatamente siguiente, se repite el proceso descrito. Este algoritmo sirve para dejar todos los elementos de la cola Q ordenados, lo que permitió un manejo más fácil en caso de las perdidas en 3.4.3.

Como se puede apreciar en la figura 21, el paquete cinco se ha perdido, por lo que al ser detectada su pérdida, los paquetes del seis en adelante son almacenados en la cola Q_d. Hay que destacar que el jitter no juega un rol importante en éste proceso, debido a la construcción de la misma, ya que se busca el paquete faltante en la cola Q en orden desde la cabeza al fin de la cola Q_d, En caso de no estar en la cabeza la cola es vaciada y se pide una retransmisión del paquete faltante al servidor y así el orden en que lleguen, no es relevante. Como se mencionó anteriormente se mantiene el orden de paquetes en la cola Q.

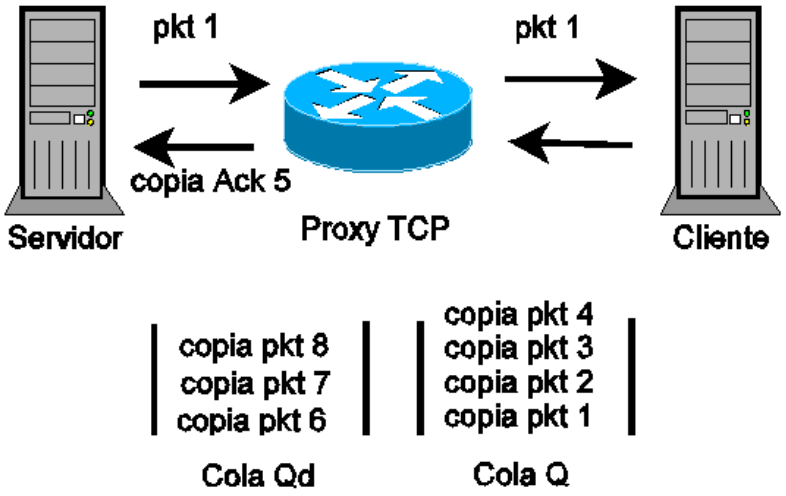

Figura 21: Modelo con pérdida entre servidor y proxy

3.4.3 Manejo de pérdidas cliente/proxy

Aquí se debe distinguir entre las perdidas ocurridas en la dirección proxy-cliente y cliente-proxy. En caso de una perdida en el primer tramo, se utiliza un algoritmo similar al que usa TCP para las perdidas, el cual consiste que al detectar la perdida el cliente enviara paquetes ACK's señalando la perdida, los que serán interceptados por el Proxy y este al recibir 3 ACK's repetidos

reenviara el paquete que este encolado en la cabeza de la cola Q. Para el segundo caso también se construyó una cola extra Q_f, que servirá para pasar las copias de paquetes desde Q a Q_f y esperen en orden hasta que la pérdida sea restablecida. Una vez que llega un paquete ACK, se verifica hasta que paquete de la cola Q su número es mayor, para proceder a eliminar los menores, esto acorde al protocolo TCP.

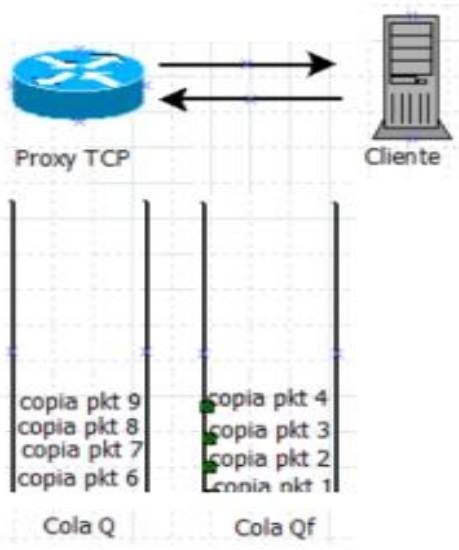

Figura 22: Modelo con pérdida entre cliente y proxy

Para solo dejar la perdida como entrada al cliente, se conectó las salidas y entradas del servidor al proxy, este va conectado al relay y este último va conectado a la entrada del cliente. La salida del cliente se conecta directo al proxy y así solo hay pérdidas en un solo sentido. La figura 24 muestra de mejor manera dichas conexiones. Hay que mencionar que esto solo es posible debido a la modularidad con que fue programado el modelo.

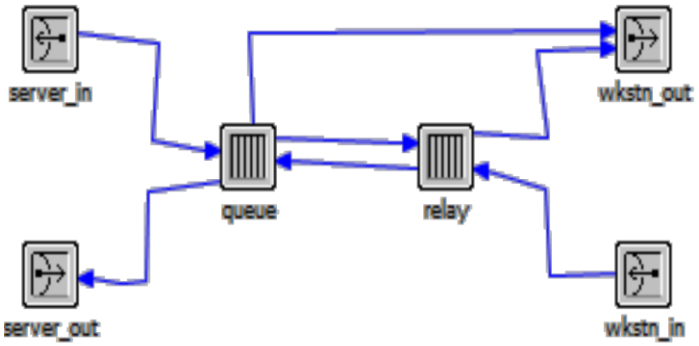

Figura 23: Conexiones Proxy-Relay

Cada uno de los módulos representados en la figura 23, tienen un desglose más profundo. El modulo Proxy, está representado por 4 estados, siendo INIT el estado inicial, donde se inicializan

las variables, se comienza a tomar el tiempo, entre otras cosas. Una vez andando, pasamos al estado IDLE que es el estado donde se espera a que llegue un paquete, una vez se cumpla la condición (PKT_IN o llegada de un paquete) se pasa al estado PKT_INFO, el cual genera una copia del paquete de información para posteriormente guardarla en la cola correspondiente Q, Q_d, Q_f y crea una copia del paquete ACK. Envía el paquete de información original al cliente y envía el ACK artificial al Servidor. Una vez hecho esto vuelve al estado IDLE. Para salir del estado IDLE se cumple la condición CLOSE, que cierra el proceso.

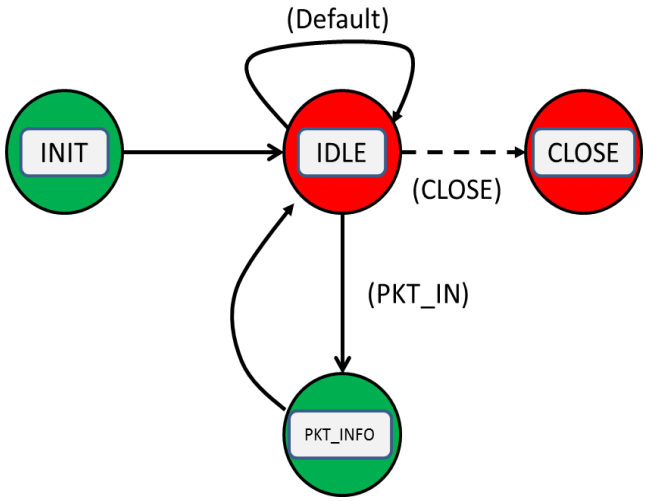

Figura 24: Estados del módulo proxy.

El módulo Relay también posee distintos estados, al igual que el modulo proxy, el estado INIT inicializa el proceso, luego se pasa al estado IDLE, donde se ejecuta la perdida, esta puede ser una sola, como también una serie de perdidas distribuidas a elección, ya sea normal, exponencial, etc.

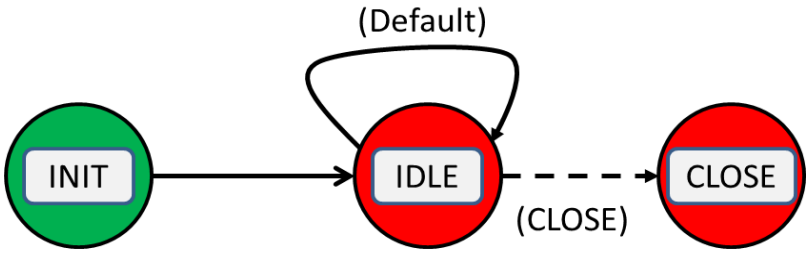

Figura 25: Estados del módulo Relay.

3.5 Trabajo como Ingeniero Eléctrico e Ingeniero en Computación.

El trabajo realizado consistió en la creación de un modelo analítico del *throughput* del protocolo de transferencia ESTP. Este modelo fue utilizado para demostrar que dicho protocolo tiene un rendimiento superior a TCP, en redes de alta latencia que implementen uno o más servidores de TCP Proxy. Esto incluyó el diseño y la programación del Proxy TCP utilizado en la demostración, haciéndole mejoras al algoritmo original con el que trabajan este tipo de *Proxies*.

Al ser un trabajo extenso y del área de telecomunicaciones es complejo poder diferenciar el trabajo especifico realizado por cada área de la ingeniería. A grandes rasgos se puede establecer que el conocimiento en computación es utilizado en una gran parte del diseño y la programación del proyecto, mientras que el criterio desarrollado como ingeniero eléctrico ayuda en la generación de las reglas y definiciones para el diseño y la programación.

En particular, el desarrollo teórico de ESTP es conocimiento técnico asociado en gran parte al área de electricidad, explicado en su totalidad en este capitulo. Por otro lado, el diseño del modelo de un Nodo Proxy es mixto, ya que se utilizan conocimientos comunes de ambas ramas de la ingeniería. Este problema se dividió en distintas etapas para ayudar a la programación del mismo, modelo sin perdidas, con perdidas entre emisor-proxy, perdidas proxy-receptor, etc; esto debido a que la programación también se pudo modular al momento de programar la forma de lidiar con los distintos tipos de pérdidas.

El diseño y la programación del nodo proxy utiliza reglas de telecomunicaciones, pero en su mayoría es un trabajo de computación. Se utilizaron diagramas de estado para modelar el sistema y posteriormente programarlo en C. El algoritmo *leaky bucket* fue sugerido por el profesor Claudio Estévez, sin embargo las modificaciones realizadas corresponden a conocimientos de ingeniería en computación, dichas modificaciones fueron el resultado de una investigación acerca del algoritmo de *leaky buket* y su relación con *proxies* TCP, donde la decisión de aumentar su capacidad fue tomada gracias a la necesidad de poder enviar una mayor cantidad de paquetes por el proxy y no solo uno cada vez, pues de esta forma, el *throughput* se veía notoriamente disminuido. Con este cambio se logra mantener el *throughput* de envío en los valores correspondientes. Adicionalmente se investigo acerca del manejo de Colas para poder trabajar de manera satisfactoria con las posibles perdidas de paquetes, las decisiones tomadas en la programación de éstas se encuentran desarrolladas en este capitulo.

Por último, las simulaciones, sobre las cuales se ahondara en el capitulo 4, representan trabajo en conjunto de ambas áreas ya que los programas utilizados (como Matlab) se usan a nivel global por diferentes áreas. Existen diversos programas de simulación de redes en el mercado, incluso algunos son software libre, pero son muy rígidos en su trabajo con otros protocolos no incluidos en sus librerías, como es el caso de ESTP. Por lo que se decidió utilizar Matlab, ya que proporciona una mayor flexibilidad para simular los comportamientos de ESTP y los datos de envío del nodo proxy. Los resultados y las conclusiones se pueden ver en los capítulos siguientes.

Capitulo 4:
RESULTADOS Y DISCUSIÓN DE PRUEBAS

4.1 Introducción

En este capítulo se analizan los distintos experimentos realizados a los protocolo ESTP, BIC, CUBIC y TCP RENO, todo esto a través de diferentes pruebas realizadas, tanto en redes de computadores reales, como en simulaciones computacionales.

Las primeras pruebas realizadas fueron para detectar el problema que tiene TCP en redes de alta latencia y descartar el fenómeno llamado "bufferbloat", esta situación se genera en redes que utilizan paquetes de información, donde el exceso de almacenamiento en buffers causa jitter y alta latencia. Es por esto que es importante poder confirmar que para las versiones actuales de TCP, como reno, BIC o CUBIC sufren de problemas de alta latencia, sin ser necesariamente el "bufferbloat", ya que si lo fuera ESTP no podría solucionarlo.

Las siguientes pruebas fueron simulaciones realizadas en programas numéricos como Matlab y Mathematica 7.0, Estas pruebas verifican que la programación del proxy TCP sea correcta. Además se prueba para distintas cantidades de proxy enlazado si sigue aumentando la velocidad de transmisión y finalmente se hace una última prueba con el proxy en la cual se determina la mejor uBICación para colocar el proxy entre servidor y cliente.

Las últimas pruebas se realizaron con equipos reales en donde se enfrentó ESTP contra TCP Reno, CUBIC y BIC y poder probar que ESTP es una mejor opción al momento de equipar nuestro nodo proxy. Ya que es más justo y más veloz que los demás protocolos actuales.

4.2 Pruebas sobre alta latencia

En esta sección se trata de comprobar que TCP tiene un problema con las redes de alta latencia, y más importante aún que la versión utilizada hoy en día, CUBIC se desempeña de mejor manera que la clásica TCP Reno, pero sufre de igual manera este problema.

El primer escenario nos muestra una transmisión de 1 GB de información a una velocidad de 1 GBPS con 100 ms de delay. En ella se ve la ventana de congestión tanto de Reno como de CUBIC comparadas.

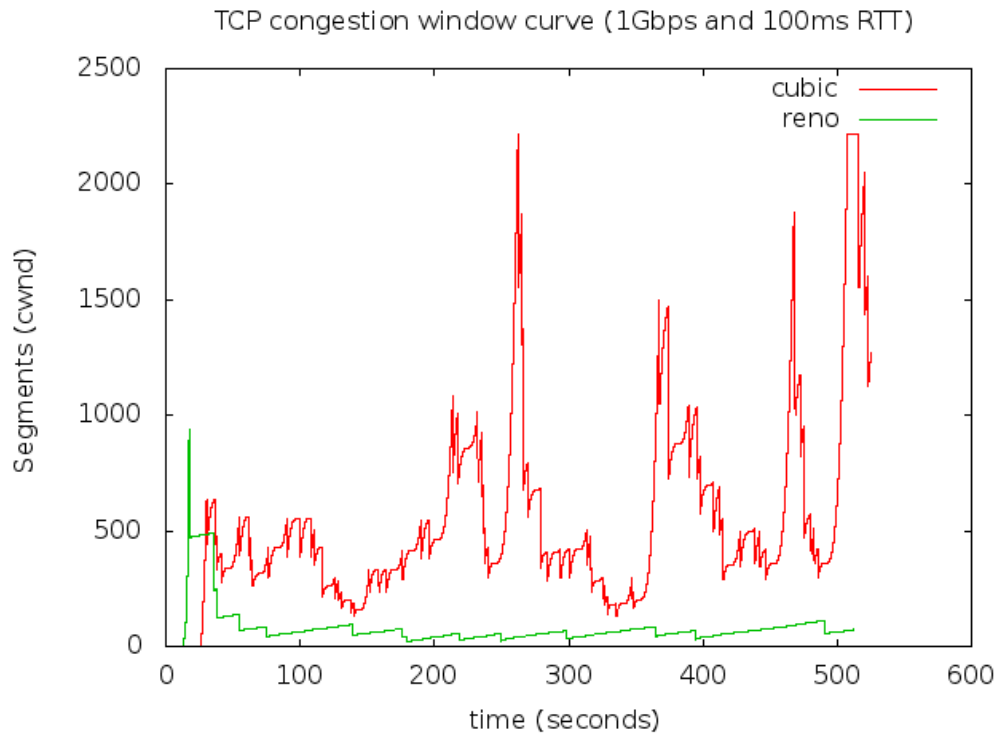

Figura 26: Comportamiento de la ventana de congestión de TCP Reno y CUBIC.

En la figura 26, se puede apreciar claramente que la ventana de congestión de CUBIC crece de forma más elevada incluso cuando hay un delay de tipo mediano.

El segundo escenario consiste en la transmisión de 10 Gb de información utilizando TCP Reno, con una pérdida de un 0.01%, un delay iguales a 5ms y con 50 ms.

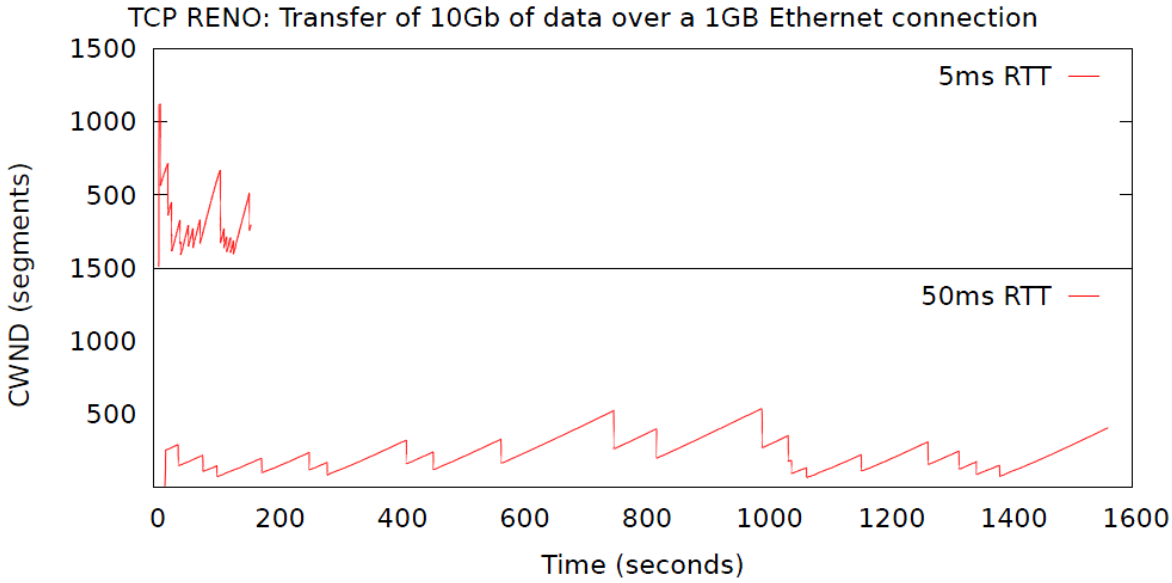

Figura 27: CWND TCP Reno 10 GB de data, a 0.01% de perdida, delay de 5 ms y 50 ms.

Como se puede ver en la figura 27, la ventana de congestión de Reno no logra recuperarse de la primera perdida y así el tamaño de la ventana no logra alcanzar su máximo de tamaño. Esto nos indica que además Reno posee un sistema de recuperación lenta ya que tarde alrededor de 1500 ms en completar la transmisión con 50 ms de delay y 200 ms la de 5 ms.

La última parte de estos experimentos nos muestra la misma transmisión anterior, realizada con el protocolo CUBIC, las diferencias quedan a la vista en la figura 28.

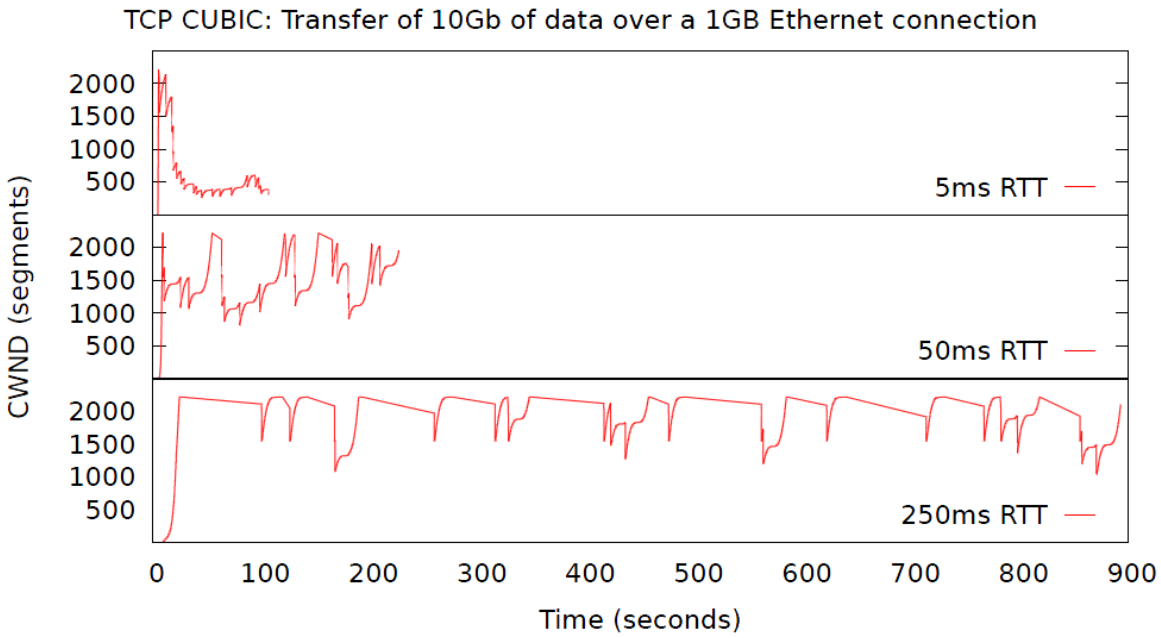

Figura 28: CWND TCP Reno 10 GB de data, a 0.01% de perdida, delay de 5 ms y 50 ms.

En esta oportunidad si se coloca la transmisión con 250 ms de delay, ya que usando Reno, se salía de la escala de tiempo razonable. Lo primero que se puede notar del gráfico es que la ventana de congestión logra reponerse de la primera perdida y las siguientes que tiene, llegando al máximo en 50 ms y 250 ms de delay, en los 5ms no lo consigue, porque se tardó muy poco tiempo en transmitir, por lo que no es necesario hacer el esfuerzo. Otro detalle a destacar es que con 250 ms de delay lograr completarlo en 900 ms, casi la mitad de lo que le toma a Reno trabajar con 50ms. Así como se vio en la teoría en los Capítulos 2 y 3 CUBIC posee una recuperación más rápida que las versiones anteriores de TCP, salvo BIC.

Como se pudo prever no existe bufferbloat en el problema de TCP Reno y su relación con las redes de alta latencia, esto porque al desarrollar experimentos en un ambiente ideal, presenta las mismas falencias que en redes abiertas. Una explicación para ello es que su control de la ventana de congestión es deficiente o muy lento y por eso CUBIC la supera en todo momento.

4.3 **Simulaciones Computacionales**

Las simulaciones computacionales realizadas en este punto sirven como “benchmark” para determinar si el proxy TCP fue programado correctamente para lograr esto se hizo un programa

en Matlab, el cual tomaba los datos guardados en un archivo de texto, el que era generado al momento de correr el programa. Dicho programa muestra el movimiento de los paquetes en las transmisiones entre servidor, cliente y proxy, además de mostrar su movimiento en las colas. El programa valida el software si al finalizar no quedan más paquetes "volando" en transmisiones o copias de paquetes en las colas.

A continuación se muestra la tabla con la configuración de TCP para poder llevar a cabo estas simulaciones, estas condiciones fueron idénticas en cada simulación realizada.

Tabla 4: Valores de Configuración de TCP.

Parametros de TCP	Valor
Versión/tipo	No especificada
Tamaño máximo de segmento (bytes)	Auto-asignado
Buffer de recepción (bytes)	4194304
Mecanismo para retraso de ACK	Segmentos/basado en reloj
Delay máximo de un ACK (sec)	0.200
Número máximo de segmentos ACK	1
Contador inicial de Slow Start (SS)	2
Retransmisión rápida	Habilitada
Barrera de ACK duplicado	3
Recuperación Rápida	TCP Reno
Escalamiento de Ventana	Habilitado
ACK Selectivo	Habilitado
Algoritmo de Nagle	Deshabilitado
Algoritmo de Karn	Habilitado
Barreras de Re-transmisión	Basado en intentos
RTO inicial (sec)	1
Mínimo RTO (sec)	0.001
Máximo RTO (sec)	64
Ganancia de RTT	0.125
Ganancia de desviación	0.25
Coeficiente de desviación de RTT	4
Granularidad del tiempo (sec)	0.0001
Persistencia de tiempo acabado (sec)	1
Aceleración	No habilitado

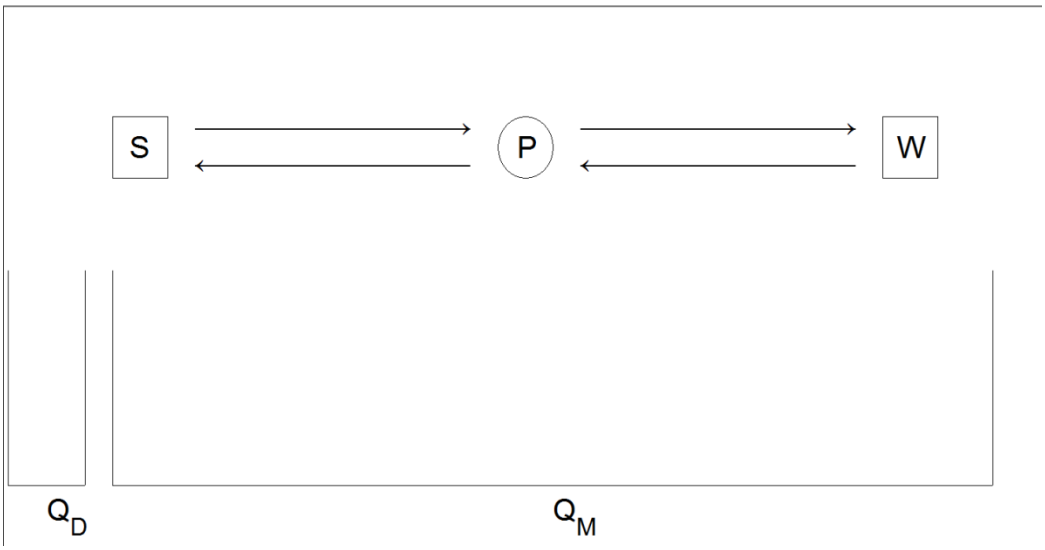

Figura 29: Interfaz Gráfica generada en Matlab que permite corroborar la programación del proxy.

Esta interfaz es dinámica, cuando se corre, comienzan a verse los paquetes de información con sus respectivos números SEQ (secuenciales) y sus números ACK. En la cola Qm se van guardando las copias de los paquetes de información, mientras que la cola Qd comienza a funcionar una vez que se detecta la perdida entre Servidor y Proxy. Para probar el caso en que haya pérdidas entre el proxy y el cliente o viceversa, solo hay que cambiar el parámetro "cola" en el código en Matlab.

Adicionalmente se desarrollaron algunas simulaciones para comprobar que efectivamente el uso de nodos proxy generaba una mejora en el rendimiento del throughput en comparación con el tiempo de ida y vuelta (RTT).

El primer experimento consistió en una conexión TCP típica con 200 ms de RTT con una cantidad variable, pero lineal de nodos. Se puede apreciar que hay un aumento del throughput de manera lineal. Este resultado se puede ver en la figura 30.

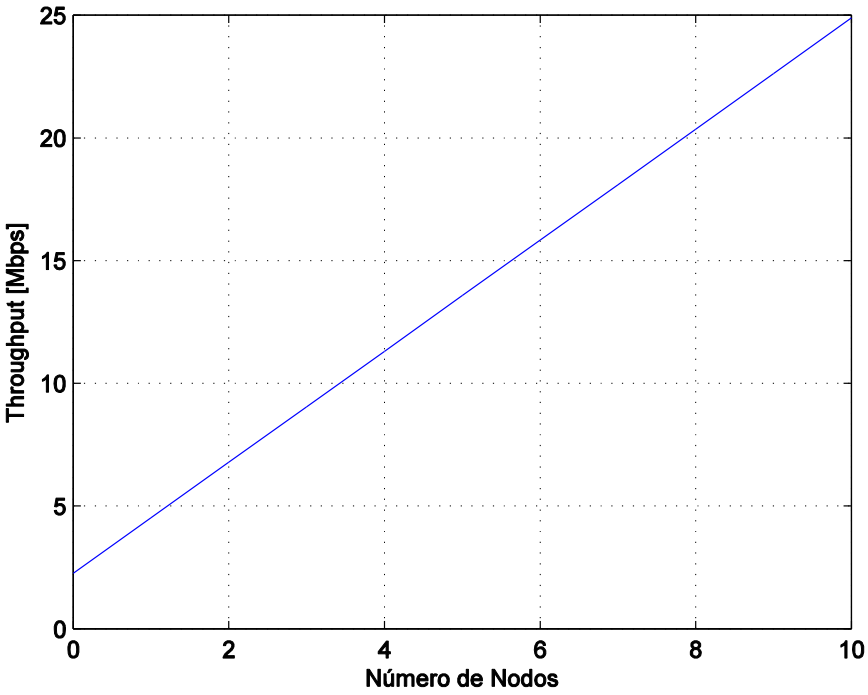

Figura 30: Simulación throughput vs Número de Nodos.

El segundo experimento consistió en determinar la mejor uBICación para colocar el Proxy TCP entre el servidor y el cliente, para ello se realizó una transmisión TCP en la cual se mantuvo el RTT constante en 200 ms, es decir el RTT del servidor al proxy más el RTT del proxy al cliente deben sumar 200 ms y con una perdida fija de 1%. En la figura 31 se puede apreciar que la mejor uBICación es en el punto medio entre ambos nodos.

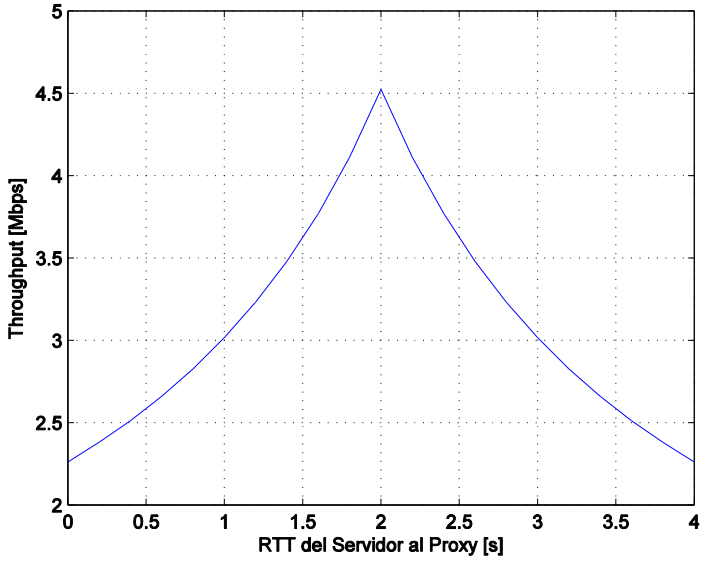

Figura 31: Segunda simulación del servidor al proxy.

4.4 **Experimentos sobre maquinas reales.**

En el este experimento, se compara TCP Reno, CUBIC y BIC contra ESTP y se mide su justicia y velocidad, como bien se dijo antes, BIC posee un algoritmo más rápido de control de su ventana de congestión, generando un problema de amistad con TCP Reno, mientras que CUBIC era más lento, pero más amigable. Estas pruebas muestran como ESTP supera a ambos siendo BIC el más cercano a él.

Antes de realizar las pruebas se configuraron los siguientes detalles de las máquinas donde se trabajó, todas poseían la distro de Linux, Ubuntu 12.04 porque era la que permitía de mejor manera generar los datos y gráficos. En la siguiente tabla se pueden apreciar los valores utilizados para configurar la parte FTP del modelo, los de mayor relevancia son, el tiempo entre solicitudes, el tamaño del archivo con el que se trabajara, esto para las simulaciones explicadas en el capítulo 4, otro parámetro importante es el tipo de servicio, el cual se suele utilizar en distintos tipo de trabajos de investigación.

Tabla 5: Datos de Configuración del Modelo

Atributo	Valor
Get/Total	100%
Tiempo entre solicitudes (segundos)	Constante (36000)
Tamaño del archivo (bytes)	Constante (2000000)
Nombre del Servidor Simbólico	FTP Server
Tipo de servicio	Mejor Esfuerzo (0)
Parámetros RSVP	Ninguno

Para poder llevar a cabo estas pruebas, se crearon distintos tipos de archivos que se compilaron en el núcleo (*kernel*) de Linux. Esto para que los reconociera y pudieran ser utilizados para las pruebas. Siempre que se utilizó ESTP, se compilo el núcleo (*kernel)* con el archivo modificado de ESTP, el cual no presento problemas.

En este experimento se realizaron tres corridas de datos, donde se calculaba el throughput para distintos tipos de delay de 10 ms a 100 ms yendo de 10 ms en 10 ms. Los resultados se pueden ver en el siguiente cuadro.

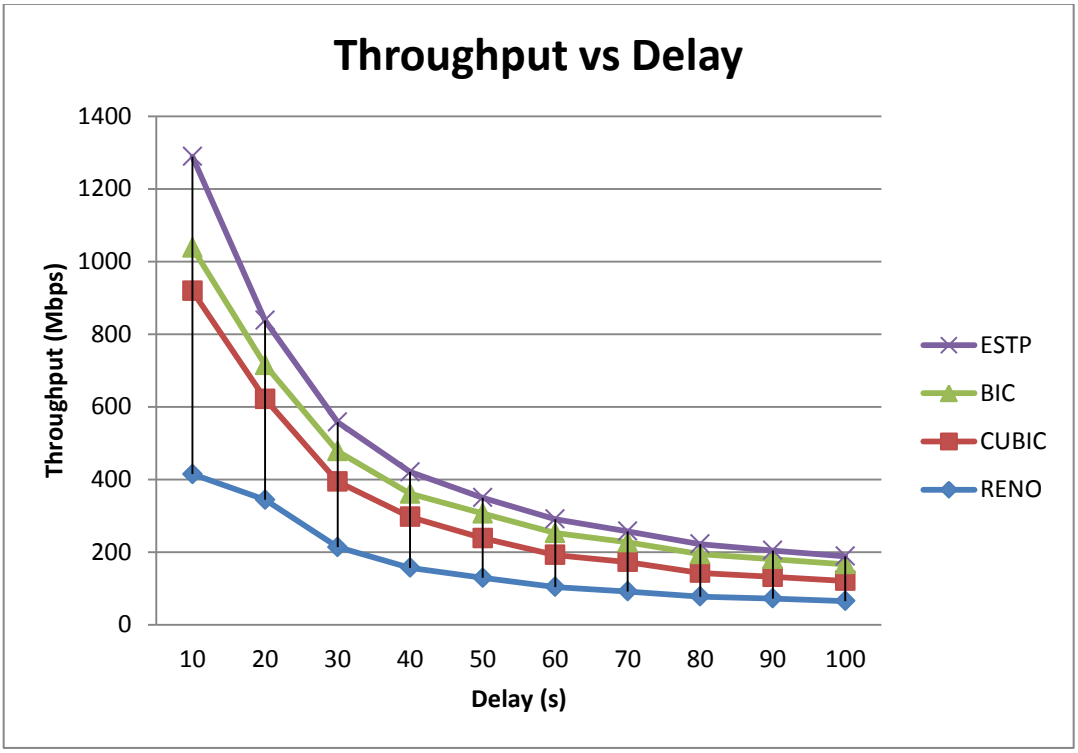

Figura 32: Gráfico de comparación, ESTP, BIC, CUBIC y TCP Reno.

La tabla con los datos obtenidos a continuación.

Tabla 6: Valores de experimentos en máquinas reales.

Protocolo	delay	Throughput 1	Throughput 2	Throughput 3	Promedio
CUBIC	100	54,4	54,2	58,7	55,7666667
CUBIC	90	57,6	56,7	64,7	59,6666667
CUBIC	80	71,8	60,7	63	65,1666667
CUBIC	70	86,1	74,3	82,6	81
CUBIC	60	89,1	88,7	85,6	87,8
CUBIC	50	105	112	110	109
CUBIC	40	140	141	139	140
CUBIC	30	177	176	187	180
CUBIC	20	272	280	280	277,333333
CUBIC	10	509	508	497	504,666667
Estp	100	19,8	23	23	21,9333333
Estp	90	20	23,4	28	23,8
Estp	80	27,3	28,2	28,4	27,9666667
Estp	70	33,5	27,9	30,1	30,5
Estp	60	38,9	38,9	37,5	38,4333333
Estp	50	42,8	44	43,9	43,5666667
Estp	40	55,7	61,9	62,7	60,1
Estp	30	82,4	81,3	76,7	80,1333333
Estp	20	129	122	118	123
Estp	10	254	243	257	251,333333

BIC	100	43,2	44,3	49,2	45,5666667
BIC	90	48,2	49,2	48,5	48,6333333
BIC	80	50,4	53,4	50,5	51,4333333
BIC	70	48,7	56,2	57,9	54,2666667
BIC	60	69,9	56,7	55	60,5333333
BIC	50	69,1	66,1	68,8	68
BIC	40	60,3	66,4	65,1	63,9333333
BIC	30	83,6	89,9	78,2	83,9
BIC	20	99,1	90,7	90,6	93,4666667
BIC	10	104	128	124	118,666667
Reno	100	72,5	61,2	61	64,9
Reno	90	72,9	73,6	69,7	72,0666667
Reno	80	74,9	81,4	76,2	77,5
Reno	70	93,6	91	89,3	91,3
Reno	60	105	99,9	107	103,966667
Reno	50	134	124	129	129
Reno	40	153	161	156	156,666667
Reno	30	217	216	208	213,666667
Reno	20	338	350	344	344
Reno	10	414	413	416	414,333333

Como se puede ver se llevaron a cabo 3 simulaciones por delay y por protocolo, el gráfico muestra el promedio de estas tres valores. Como se predijo ESTP muestra un mayor throughput en promedio, luego lo sigue BIC, luego CUBIC y finalmente TCP Reno.

Capitulo 5: CONCLUSIONES Y TRABAJO FUTURO

En este capítulo se presentan las conclusiones de los experimentos y simulaciones explicados en el capítulo 4.

5.1 Conclusiones

Se cumplió el objetivo general de este trabajo, el cual era crear un modelo analítico del throughput de ESTP en redes de alta latencia que implementen uno o más servidores de proxy TCP, además se mostró que el protocolo ESTP tiene un rendimiento igual o superior a TCP en dichas redes al enviar grandes volúmenes de datos. Todo esto quedó demostrado en las pruebas sobre máquinas reales y en las simulaciones computacionales.

El trabajo realizado cumplió de gran manera con los objetivos específicos, se investigo acerca de TCP y ESTP encontrando distintas versiones de TCP sobretodo que sirvieron para comparar contra ESTP, se llegó a una fórmula concreta para ESTP lo que da una base sólida al trabajo teórico y respalda al trabajo práctico. Junto con esto se mostró en las pruebas de alta latencia que las implementaciones de TCP bajan notablemente su rendimiento en redes con alto delay al transmitir grandes volúmenes de información, aunque para la última versión CUBIC, el problema se maneja de una mejor forma. Siendo ESTP la solución natural para esto, ya que posee una velocidad sobre la ventana de congestión mejor que BIC y su justicia y amistad con TCP Reno son mejores.

Cabe destacar que el desarrollo de la formula teórica de ESTP se hizo en paralelo con la programación del nodo proxy y por lo mismo, no se pudo tener una implementación de un proxy ESTP. Aunque como se mencionó en el capítulo 3, esto no afecta los resultados obtenidos en el capítulo 4 debido a que el algoritmo utilizado para resolver el problema es independiente del protocolo de control. En este caso, se utilizaría una variación en los parámetros de configuración, utilizando como método de recuperación rápida el algoritmo usado por ESTP y no por TCP Reno como se especificó en la tabla 4.

También hay que destacar las simulaciones computacionales y sus resultados, ya que en ellas se muestra como el número de proxy unidos aumenta de forma lineal el throughput de transmisión, aunque para cantidades grandes de nodos, se va a llegar a un punto donde la velocidad no seguirá aumentando y se mantendrá constante por más proxies que se añadan al sistema. También en las mismas pruebas se menciona que el mejor punto para colocar el proxy es al medio entre el servidor y el cliente o entre proxy y proxy, esto porque al ser inversamente proporcional al máximo de los RTT, y al ser RTT2 = 1-RTT/(RTT1+RTT2), se busca que haya la menor diferencia entre ellos y por lo tanto son iguales, igual a la mitad. La definición correcta es que el mejor punto para colocar un proxy entre dos nodos del sistema es al medio de ellos, donde sus RTT sean similares.

5.2 **Trabajo Futuro**

El paso siguiente en este trabajo es poder controlar o ajustar el buffer de envío del proxy, como se ve en la tabla 4, el valor que se coloca para esto es de 4 MB, que es muy grande, pero existen factores de corrección y en la programación misma del proxy, se propone obtener el valor del buffer cuando se envía en primer paquete ACK y a partir de ahí comenzar a construir un modelo que permita anteponerse a situación de riesgo como por ejemplo que el buffer este completo, las colas también y haya una perdida, lo que impediría que se recibiera con éxito el paquete retransmitido por el servidor o el mismo proxy.

Otro propuesto es el de implementar el proxy con ESTP y posteriormente poder pasarlo a un satélite y poder realizar las pruebas reales, quizás primero con globos aerostáticos y luego el espacio. Esto serviría para probar el desempeño de ESTP en redes de alta latencia como los son las comunicaciones satelitales.

Otro propuesto sobre el cual se podría trabajar sería relacionar el Proxy TCP con redes inteligentes, primero hay que lograr implantar el proxy sobre alguna unidad de funcionamiento de la red inteligente, se sugiere hacerlo sobre una Unidad de Regeneración BPL.

Capitulo 6:
BIBLIOGRAFÍA

1. C. Estevez, S. Angulo, G. Ellinas, and G.-K. Chang, “Ethernet-Services Transport Protocol for Carrier Ethernet Networks,” IEEE GLOBECOM 2011, Houston, TX, December 2011.

2. S. Floyd, HighSpeed TCP for Large Congestion Windows, IETF RFC 3649, Dec. 2003.

3. Estevez, Claudio, Sergio Angulo, Andres Abujatum, Georgios Ellinas, Cheng Liu, and Gee-Kung Chang. "A Carrier-Ethernet oriented transport protocol with a novel congestion control and QoS integration: Analytical, simulated and experimental validation." In *Communications (ICC), 2012 IEEE International Conference on*, pp. 2673-2678. IEEE, 2012.

4. Caini, C., Firrincieli, R., Marchese, M., de Cola, T., Luglio, M., Roseti, C., et al. (2006). Transport layer protocols and architectures for satellite networks. International Journal Of Satellite Communications And Networking, 25, 1-26. Retrieved March 9, 2009 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.73.7647&rep=rep1&type=pdf

5. Zhang, Grace. Modeling TCP Reno with RED. Communication Networks Laboratory, Julio 2003.

6. RFC 793, Transmission Control Protocol, DARPA INTERNET PROGRAM Protocol Specification, Septiembre 1981

7. Padhye, J., Firoin, V., Towsley, D., Kurose, J., “Modeling TCP Throughput: A Simple Model and its Empirical Validation” , Amherst, MA 01003 USA.

8. Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling tcp throughput: A simple model and its empirical validation. In ACM SIGCOMM Computer Communication Review, volume 28, pages 303{314. ACM, 1998.

9. Yang Richard Yang and Simon S Lam. General aimd congestion control. In Network Protocols, 2000. Proceedings. 2000 International Conference on, pages 187{198. IEEE, 2000.

10. Postel, Jon. "RFC 793." *Internet Engineering Task Force, SRI International*(1981).

11. Xu Lisong, Harfoush Khaled, Rhee Injong. “Binary Increase Congestion Control for Fast, Long Distance Networks”

12. I. Rhee, X. Lisong, “CUBIC: A New TCP-Friendly High-Speed TCP Variant”

13. Sloan Digital Scan Survey, Sky Server, http://skyserver.sdss.org

14. A. Abujatum, J. Maldonado, J. Bustos, “TCP Performance Improvement with the Inclusion of TCP Proxy Nodes”. LATINCOM Diciembre 2013.

15. A. Abujatum, F. Salas, C. Estevez, “Implementación de un Nodo Proxy en las unidades de regeneración de las líneas de Transmisión de Banda Ancha en Redes Inteligentes”, ChileCon Septiembre 2013.

ANEXO A: Codigo del Programa

```
/* Process model C form file: proxy.pr.c */

/* This variable carries the header into the object file */

const char proxy_pr_c [] = "MIL_3_Tfile_Hdr_ 145A 30A modeler 7 5213D2CF 5213D2CF 1
OWL-3 Lab 0 0 none none 0 0 none 0 0 0 0 0 0 0 0 1e80 8
";

#include <string.h>

#include <opnet.h>







/* Header Block */



#include <tcp_seg_sup.h>

#include <ip_dgram_sup.h>



#define PKT_IN (op_intrpt_type () == OPC_INTRPT_STRM)

#define CLOSE op_intrpt_type() == OPC_INTRPT_ENDSIM

#define WKSTN 0

#define SERVER 1

#define DIRECT 2



#define QM 0

#define QD 1
```

```
#define QF 2



#define PIN 0

#define POUT 1



#define TAIL 0

#define HEAD 1



/* End of Header Block */


#if !defined (VOSD_NO_FIN)

#undef BIN

#undef BOUT

#define BIN FIN_LOCAL_FIELD(_op_last_line_passed) = __LINE__ - _op_block_origin;

#define BOUT BIN

#define BINIT FIN_LOCAL_FIELD(_op_last_line_passed) = 0; _op_block_origin = __LINE__;

#else

#define BINIT

#endif /* #if !defined (VOSD_NO_FIN) */
```

```
/* State variable definitions */

typedef struct

        {

        /* Internal state tracking for FSM */

        FSM_SYS_STATE

        /* State Variables */

        Packet *                pkt_ip                  ;   /* extraer
informaciÛn del paquete */

        int                     port                    ;   /* Puerto de entrada */

        Packet *                pkt_tcp                 ;   /* Sirva para
extraer el paquete TCP */

        TcpT_Seg_Fields *       tcp_fd_ptr              ;   /* InformaciÛn
del Header del paquete TCP */

        FILE *                  pFile_wkstn_IP          ;   /* puntero al
archivo */

                                                        /* Cliente IP      */

        IpT_Dgram_Fields *      ip_fd_ptr               ;   /* informaciÛn
del header del paquete IP */

        IpT_Dgram_Fields *      ip_copy_fd_ptr          ;   /*
informaciÛn del header del paquete IP copia */

        Packet *                pkt_ip_fake_ack         ;

        Packet *                pkt_tcp_fake_ack        ;

        int                     on_off                  ;

        InetT_Address           tmp                     ;   /* variable
auxiliar para hacer el cambio src a dest. */
```

```
        FILE *                  pFile_proxy_IP                  ;       /* Proxy IP */

        FILE *                  pFile_wkstn_TCP                 ;       /* Cliente TCP
*/

        FILE *                  pFile_proxy_TCP                 ;       /* Proxy TCP
*/

        TcpT_Seg_Fields *               tcp_copy_fd_ptr                 ;       /*
Informacion del HEADER de TCP - Copy */

        int                     tmp_tcp                 ; /* temp para copiar
TCP */

        int                     tmp_ip                  ;   /* temp para IP */

        Packet *                trash                   ;       /* basura - data
field de la copia */

        Packet *                pkt_queue_out                   ;       /* copia
de paquete en el queue que extraemos */

        Packet *                pkt_queue_in                    ;/* Copia de
paquete que queda en el queue */

        int                     ack_wkstn                ; /* ack del wkstn ->
proxy */

        int                     seq_qout                ; /* seq del head del
queue */

        Packet *                pkt_queue_out_tcp               ;       /* pckt
TCP del queue out (borrar del queue) */

        TcpT_Seg_Fields *               queue_out_tcp_fd_ptr                ;       /*
puntero al campo del pckt queue_tcp_out (borrar queue) */

        int                     ack_proxy                ; /* ack que envia el
proxy al server. */

        int                     seq_server              ; /* seq que envia el
server. */

        int                     semaforo                ;

        int                     ack_ptr                 ;
```

```
	Packet *			pkt_queue_out_find		;	/*
Paquete que sirve para obtener el HEAD de QD en la funciÛn FIND */

	int			solution		;

	int			ack_proxy_old		;

	FILE *			matlab_anim		;

	FILE *			matlab_queue		;

	FILE *			pFile_close		;

	Packet *			pkt_queue_retransmit		;	/*	Pkt
que se retransmite en el proxy cuando el wkstn un duplicate ack */

	int			len_qout		;	/* Lago del pkt en el
HEAD del QUEUE */

	int			cont_ack		;	/* contador de ack's
para el single duplicate ack */

	int			old_ack		;	/* ultimo ack recibido
por el proxy desde el wkstn */

	int			loss_on		;	/* Switch de la perdida
wkstn->proxy */

	Packet *			pkt_fwd		;	/* pkt que se
envia de la queue QF al recuperarse de la perdida. */

	Packet *			pkt_fwd_tcp		;

	TcpT_Seg_Fields *		fwd_tcp_fd_ptr		;

	int			seq_qfwd		;

	} proxy_state;



#define pkt_ip			op_sv_ptr->pkt_ip

#define port			op_sv_ptr->port

#define pkt_tcp			op_sv_ptr->pkt_tcp

#define tcp_fd_ptr		op_sv_ptr->tcp_fd_ptr
```

```
#define pFile_wkstn_IP          op_sv_ptr->pFile_wkstn_IP
#define ip_fd_ptr               op_sv_ptr->ip_fd_ptr
#define ip_copy_fd_ptr          op_sv_ptr->ip_copy_fd_ptr
#define pkt_ip_fake_ack         op_sv_ptr->pkt_ip_fake_ack
#define pkt_tcp_fake_ack        op_sv_ptr->pkt_tcp_fake_ack
#define on_off                  op_sv_ptr->on_off
#define tmp                     op_sv_ptr->tmp
#define pFile_proxy_IP          op_sv_ptr->pFile_proxy_IP
#define pFile_wkstn_TCP         op_sv_ptr->pFile_wkstn_TCP
#define pFile_proxy_TCP         op_sv_ptr->pFile_proxy_TCP
#define tcp_copy_fd_ptr         op_sv_ptr->tcp_copy_fd_ptr
#define tmp_tcp                 op_sv_ptr->tmp_tcp
#define tmp_ip                  op_sv_ptr->tmp_ip
#define trash                   op_sv_ptr->trash
#define pkt_queue_out           op_sv_ptr->pkt_queue_out
#define pkt_queue_in            op_sv_ptr->pkt_queue_in
#define ack_wkstn               op_sv_ptr->ack_wkstn
#define seq_qout                op_sv_ptr->seq_qout
#define pkt_queue_out_tcp       op_sv_ptr->pkt_queue_out_tcp
#define queue_out_tcp_fd_ptr    op_sv_ptr->queue_out_tcp_fd_ptr
#define ack_proxy               op_sv_ptr->ack_proxy
#define seq_server              op_sv_ptr->seq_server
#define semaforo                op_sv_ptr->semaforo
#define ack_ptr                 op_sv_ptr->ack_ptr
```

```
#define pkt_queue_out_find              op_sv_ptr->pkt_queue_out_find
#define solution                        op_sv_ptr->solution
#define ack_proxy_old                   op_sv_ptr->ack_proxy_old
#define matlab_anim                     op_sv_ptr->matlab_anim
#define matlab_queue                    op_sv_ptr->matlab_queue
#define pFile_close                     op_sv_ptr->pFile_close
#define pkt_queue_retransmit            op_sv_ptr->pkt_queue_retransmit
#define len_qout                        op_sv_ptr->len_qout
#define cont_ack                        op_sv_ptr->cont_ack
#define old_ack                         op_sv_ptr->old_ack
#define loss_on                         op_sv_ptr->loss_on
#define pkt_fwd                         op_sv_ptr->pkt_fwd
#define pkt_fwd_tcp                     op_sv_ptr->pkt_fwd_tcp
#define fwd_tcp_fd_ptr                  op_sv_ptr->fwd_tcp_fd_ptr
#define seq_qfwd                        op_sv_ptr->seq_qfwd


/* These macro definitions will define a local variable called      */
/* "op_sv_ptr" in each function containing a FIN statement.         */
/* This variable points to the state variable data structure,   */
/* and can be used from a C debugger to display their values.       */
#undef FIN_PREAMBLE_DEC
#undef FIN_PREAMBLE_CODE
#define FIN_PREAMBLE_DEC      proxy_state *op_sv_ptr;
#define FIN_PREAMBLE_CODE   \
```

```
		op_sv_ptr = ((proxy_state *)(OP_SIM_CONTEXT_PTR->_op_mod_state_ptr));



/* Function Block */


#if !defined (VOSD_NO_FIN)
enum { _op_block_origin = __LINE__ + 2};
#endif


static void print_flag (FILE* pFile, unsigned int flag_var)
	{


	int fr = flag_var;


	FIN (print_flag (pFile, flag_var));


	if (fr == 0){
			fprintf (pFile, "NONE\n");
	}else{


		if(fr >= 0x80 ){
			fr=fr-0x80;
			fprintf (pFile, "CWR ");
		}
```

```
if(fr >= 0x40 ){
        fr=fr-0x40;
        fprintf (pFile, "ECE ");
}
if(fr >= 0x20 ){
        fr=fr-0x20;
        fprintf (pFile, "URG ");
}
if(fr >= 0x10 ){
        fr=fr-0x10;
        fprintf (pFile, "ACK ");
}
if(fr >= 0x08 ){
        fr=fr-0x08;
        fprintf (pFile, "PSH ");
}
if(fr >= 0x04 ){
        fr=fr-0x04;
        fprintf (pFile, "RST ");
}
if(fr >= 0x02 ){
        fr=fr-0x02;
        fprintf (pFile, "SYN ");
}
```

```
                if(fr >= 0x01 ){
                        fr=fr-0x01;
                        fprintf (pFile, "FIN ");
                }


        }


        FOUT;
        }


//imprime en pantalla


static void print_flag_2 ( unsigned int flag_var)
        {

        int fr = flag_var;

        FIN (print_flag_2 (flag_var));

        if (fr == 0){
                        printf ( "NONE\n");
```

```
}else{

        if(fr >= 0x80 ){
                fr=fr-0x80;
                printf ( "CWR ");
        }
        if(fr >= 0x40 ){
                fr=fr-0x40;
                printf ("ECE ");
        }
        if(fr >= 0x20 ){
                fr=fr-0x20;
                printf ("URG ");
        }
        if(fr >= 0x10 ){
                fr=fr-0x10;
                printf ( "ACK ");
        }
        if(fr >= 0x08 ){
                fr=fr-0x08;
                printf ( "PSH ");
        }
        if(fr >= 0x04 ){
                fr=fr-0x04;
```

```
                printf ( "RST ");
        }
        if(fr >= 0x02 ){
                fr=fr-0x02;
                printf ( "SYN ");
        }
        if(fr >= 0x01 ){
                fr=fr-0x01;
                printf ( "FIN ");
        }


    }


    FOUT;
    }


static int check_FIN_flag ( unsigned int flag_var)
    {

    int fr = flag_var;
```

```
FIN (check_FIN_flag (flag_var));



        if(fr >= 0x80 ){
                fr=fr-0x80;
        }
        if(fr >= 0x40 ){
                fr=fr-0x40;
        }
        if(fr >= 0x20 ){
                fr=fr-0x20;
        }
        if(fr >= 0x10 ){
                fr=fr-0x10;
        }
        if(fr >= 0x08 ){
                fr=fr-0x08;
        }
        if(fr >= 0x04 ){
                fr=fr-0x04;
        }
        if(fr >= 0x02 ){
                fr=fr-0x02;
        }
```

```
		if(fr >= 0x01 ){
			fr=fr-0x01;
			FRET(1);
		}




	FRET(0);
	}



static int extract_ack_value (Packet * ip_ptr)
	{

	int ack_value;
	Packet * tcp_ptr;
	TcpT_Seg_Fields * tcp_field;

	FIN (extract_ack_flag (ip_ptr))

		op_pk_nfd_get (ip_ptr, "data", &tcp_ptr);
		op_pk_nfd_access (tcp_ptr, "fields", &tcp_field);
```

```
		ack_value = tcp_field->ack_num;
		op_pk_nfd_set (ip_ptr, "data", tcp_ptr);


	FRET(ack_value);
	}



static int extract_seq_value (Packet * ip_ptr)
	{

	int seq_value;
	Packet * tcp_ptr;
	TcpT_Seg_Fields * tcp_field;

	FIN (extract_seq_flag (ip_ptr))

		op_pk_nfd_get (ip_ptr, "data", &tcp_ptr);
		op_pk_nfd_access (tcp_ptr, "fields", &tcp_field);
		seq_value = tcp_field->seq_num;
		op_pk_nfd_set (ip_ptr, "data", tcp_ptr);

	FRET(seq_value);
	}
```

```
static void matlab_packet_travel(Packet * ip_ptr, int dir, int puerto)
	{

	int seq_value;
	int ack_value;
	Packet * tcp_ptr;
	TcpT_Seg_Fields * tcp_field;

	FIN (matlab_packet_travel(ip_ptr))

	op_pk_nfd_get (ip_ptr, "data", &tcp_ptr);
	op_pk_nfd_access (tcp_ptr, "fields", &tcp_field);
	seq_value = tcp_field->seq_num;
	ack_value = tcp_field->ack_num;

	fprintf (matlab_anim, "%4.15f\t%d\t%d\t%d\t%d\n", op_sim_time(), tcp_field->seq_num, tcp_field->ack_num, puerto, dir);

	op_pk_nfd_set (ip_ptr, "data", tcp_ptr);

	FOUT;

	}
```

```
static void matlab_packet_queue(Packet * ip_ptr, int cola,int accion,int pos )

        {


        int seq_value;

        int ack_value;

        Packet * tcp_ptr;

        TcpT_Seg_Fields * tcp_field;


        FIN (matlab_packet_queue(ip_ptr))


        op_pk_nfd_get (ip_ptr, "data", &tcp_ptr);

        op_pk_nfd_access (tcp_ptr, "fields", &tcp_field);

        seq_value = tcp_field->seq_num;

        ack_value = tcp_field->ack_num;


        fprintf (matlab_queue, "%4.15f\t%d\t%d\t%d\t%d\t%d\t\n", op_sim_time(), tcp_field->seq_num, tcp_field->ack_num, cola, accion, pos);


        op_pk_nfd_set (ip_ptr, "data", tcp_ptr);


        FOUT;


        }
```

```
static int find(void)
	{

	int seq_value;
	Packet * tcp_ptr;
	TcpT_Seg_Fields * tcp_field;

	FIN (find())

		if (op_subq_empty (QD)==OPC_FALSE){

			/***********Bloque para obtener el SEQ de la HEAD del queue (Aumentar el SEQ)*******/

			pkt_queue_out_find = op_subq_pk_remove (QD, OPC_QPOS_HEAD); //op_subq_pk_remove( id_QUEUE, posiciÛn en el QUEUE);
			op_pk_nfd_get (pkt_queue_out_find, "data", &tcp_ptr);
			op_pk_nfd_access (tcp_ptr, "fields", &tcp_field);
			seq_value = tcp_field->seq_num;		// SEQ DEL PAQUETE EN EL QUEUE
			op_pk_nfd_set (pkt_queue_out_find, "data", tcp_ptr);
			matlab_packet_queue(pkt_queue_out_find, QD,0,HEAD);
			/**********************************FIN BLOQUE**********************/
```

```
                    if(seq_value == ack_proxy){ //Si es el pkt que busco lo encolo


                            op_subq_pk_insert (QM, pkt_queue_out_find,
OPC_QPOS_TAIL);// Inserto Pkt en Queue principal

                            matlab_packet_queue(pkt_queue_out_find, QM,1,HEAD);

                            ack_proxy_old = ack_proxy;

                            ack_proxy = seq_value + tcp_field->data_len;

                            tcp_copy_fd_ptr->ack_num   = ack_proxy;


                    }else{ //si no es, retorno 0


                            //matlab_packet_queue(pkt_queue_out_find, QD);

                            op_subq_pk_insert (QD, pkt_queue_out_find,
OPC_QPOS_HEAD);// Re-Inserto Pkt en Qd

                            matlab_packet_queue(pkt_queue_out_find, QD,1,HEAD);

                            FRET(0);

                    }


            }else{ // Si esta vacìo QD, retorna -1

                    FRET(-1);

            }


            FRET(1); // Si todo funciono bien retorna 1

    }
```

```
/* End of Function Block */


/* Undefine optional tracing in FIN/FOUT/FRET */
/* The FSM has its own tracing code and the other */
/* functions should not have any tracing.                    */
#undef FIN_TRACING
#define FIN_TRACING

#undef FOUTRET_TRACING
#define FOUTRET_TRACING

#if defined (__cplusplus)
extern "C" {
#endif
        void proxy (OP_SIM_CONTEXT_ARG_OPT);
        VosT_Obtype _op_proxy_init (int * init_block_ptr);
        void _op_proxy_diag (OP_SIM_CONTEXT_ARG_OPT);
        void _op_proxy_terminate (OP_SIM_CONTEXT_ARG_OPT);
        VosT_Address _op_proxy_alloc (VosT_Obtype, int);
        void _op_proxy_svar (void *, const char *, void **);
```

```
#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif




/* Process model interrupt handling procedure */



void
proxy (OP_SIM_CONTEXT_ARG_OPT)
	{
#if !defined (VOSD_NO_FIN)
	int _op_block_origin = 0;
#endif
	FIN_MT (proxy ());

		{
		/* Temporary Variables */
		int o_len;
		int f_len;
		int o_size;
```

```
int temp;
/* End of Temporary Variables */



FSM_ENTER ("proxy")


FSM_BLOCK_SWITCH
{
/*---------------------------------------------------------*/
/** state (init) enter executives **/
FSM_STATE_ENTER_FORCED_NOLABEL (0, "init", "proxy [init enter execs]")
FSM_PROFILE_SECTION_IN ("proxy [init enter execs]", state0_enter_exec)
{
pFile_wkstn_IP = fopen ("C:\\Satellite_Project\\wkstn_IP.txt","w");
pFile_wkstn_TCP = fopen ("C:\\Satellite_Project\\wkstn_TCP.txt","w");
pFile_proxy_IP = fopen ("C:\\Satellite_Project\\proxy_IP.txt","w");
pFile_proxy_TCP = fopen ("C:\\Satellite_Project\\proxy_TCP.txt","w");
matlab_anim = fopen ("C:\\Satellite_Project\\matlab_sat_anim.txt","w");
matlab_queue = fopen ("C:\\Satellite_Project\\matlab_queue.txt","w");
```

```
                    //pFile_pkt = fopen ("C:\\Satellite_Project\\pkt.txt","w");


                    on_off = 0;

                    semaforo =1;

                    old_ack=-1;

                    loss_on = 0;

                    }

                    FSM_PROFILE_SECTION_OUT (state0_enter_exec)


            /** state (init) exit executives **/

            FSM_STATE_EXIT_FORCED (0, "init", "proxy [init exit execs]")




            /** state (init) transition processing **/

            FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "init",
"idle", "tr_0", "proxy [init -> idle : default / ]")

                    /*---------------------------------------------------------*/




            /** state (idle) enter executives **/

            FSM_STATE_ENTER_UNFORCED (1, "idle", state1_enter_exec,
"proxy [idle enter execs]")



            /** blocking after enter executives of unforced state. **/
```

```
FSM_EXIT (3,"proxy")



/** state (idle) exit executives **/

FSM_STATE_EXIT_UNFORCED (1, "idle", "proxy [idle exit execs]")



/** state (idle) transition processing **/

FSM_PROFILE_SECTION_IN ("proxy [idle trans conditions]", state1_trans_conds)

FSM_INIT_COND (PKT_IN)

FSM_TEST_COND (CLOSE)

FSM_DFLT_COND

FSM_TEST_LOGIC ("idle")

FSM_PROFILE_SECTION_OUT (state1_trans_conds)


FSM_TRANSIT_SWITCH

	{

	FSM_CASE_TRANSIT (0, 2, state2_enter_exec, ;, "PKT_IN", "", "idle", "pkt_info", "tr_2", "proxy [idle -> pkt_info : PKT_IN / ]")

	FSM_CASE_TRANSIT (1, 3, state3_enter_exec, ;, "CLOSE", "", "idle", "close", "tr_5", "proxy [idle -> close : CLOSE / ]")

	FSM_CASE_TRANSIT (2, 1, state1_enter_exec, ;, "default", "", "idle", "idle", "tr_4", "proxy [idle -> idle : default / ]")

	}

	/*---------------------------------------------------------*/
```

```
/** state (pkt_info) enter executives **/

FSM_STATE_ENTER_FORCED (2, "pkt_info", state2_enter_exec, "proxy [pkt_info enter execs]")

FSM_PROFILE_SECTION_IN ("proxy [pkt_info enter execs]", state2_enter_exec)

{

port = op_intrpt_strm ();

pkt_ip = op_pk_get (port);


matlab_packet_travel(pkt_ip, PIN, port);


//printf("Entrada al proxy");

//op_subq_print (0);


temp = extract_seq_value(pkt_ip);

//printf("ack_value_test = %d\n", temp);


op_pk_nfd_get (pkt_ip, "data", &pkt_tcp);


op_pk_nfd_access (pkt_ip, "fields", &ip_fd_ptr);

op_pk_nfd_access (pkt_tcp, "fields", &tcp_fd_ptr);
```

```
                    if(!on_off && port==WKSTN ){

                            ack_proxy = tcp_fd_ptr->ack_num;  //para que no haya error en el triple-duplicate ack.

                        }


                    pkt_tcp_fake_ack = op_pk_copy (pkt_tcp);


                    op_pk_nfd_access (pkt_tcp_fake_ack, "fields", &tcp_copy_fd_ptr);



                    op_pk_nfd_set (pkt_ip, "data", pkt_tcp);



                        // SE IMPRIME EN TCP

                        /*  printf ( "time: %4.15f, src_port: %d\tdest_port: %d\tseq_num: %u\tack_num: %u\trcv_win: %u urgent_pointer: %d data_len: %d\tflags:",op_sim_time(), tcp_fd_ptr->src_port,tcp_fd_ptr->dest_port,tcp_fd_ptr->seq_num,tcp_fd_ptr->ack_num,tcp_fd_ptr->rcv_win,tcp_fd_ptr->urgent_pointer,tcp_fd_ptr->data_len);

                            print_flag_2(tcp_fd_ptr->flags);

                            printf ( "\tlocal_key: %X remote_key: %X\n",tcp_fd_ptr->local_key,tcp_fd_ptr->remote_key);*/
```

```
                    //SE TERMINA DE IMPRIMIR EN TCP



                if(check_FIN_flag(tcp_fd_ptr->flags)){


                    on_off=0;


                }


                if(port){

                    //printf ("Server->Wkstn\n");


                    /*      fprintf (pFile, "Server->Wkstn: ");

                    // SE IMPRIME EN TCP

                            fprintf (pFile, "src_port: %d\tdest_port: %d\tseq_num: %u\tack_num: %u\trcv_win: %u urgent_pointer: %d data_len: %d\tflags:",tcp_fd_ptr->src_port,tcp_fd_ptr->dest_port,tcp_fd_ptr->seq_num,tcp_fd_ptr->ack_num,tcp_fd_ptr->rcv_win,tcp_fd_ptr->urgent_pointer,tcp_fd_ptr->data_len);

                            print_flag(tcp_fd_ptr->flags);

                            fprintf (pFile, "\tlocal_key: %X remote_key: %X\n",tcp_fd_ptr->local_key,tcp_fd_ptr->remote_key);

                    // SE TERMINA DE IMPRIMIR EN TCP

                    */

                    o_len  = ip_fd_ptr->orig_len;

                    f_len  = ip_fd_ptr->frag_len;

                    o_size = ip_fd_ptr->original_size;
```

```
                                if (o_len == 1480 && on_off == 0){ // primer packet con
informacion

                                        on_off = 1;

                        //              fprintf (pFile, "Triple-duplicate ACK over\n");

                                }



                                // SE IMPRIME EN IP

                        //              fprintf(pFile,"orig_len: %d\tident: %d\t frag_len:
%d\t ttl: %u\tsrc_addr: %x\tdest_addr: %x\tprotocol: %d\t tos: %d\toffset: %d\ttraffic_class:
%s\tsrc_internal_addr: %x\tdest_internal_addr: %x\tcompression_method: %d\toriginal_size:
%d\toptions_field_set: %d\ttunnel_pkt_at_src: %d\tdecompression_delay: %f\tnext_addr:
%x\ttunnel_end_addr: %x\tnet_addr_tunneled: %x\ttunnel_ptr: %d\ttunnel_start_time:
%0.0f\tvpn_stamp_time: %f\ticmp_type: %x\tencap_count: %d\tipv6_extension_hdr_info_ptr:
%d\tsrc_num_hops_stat_hndl_ptr %d \n",o_len,ip_fd_ptr->ident,f_len,ip_fd_ptr->ttl,ip_fd_ptr-
>src_addr.address,ip_fd_ptr->dest_addr.address,ip_fd_ptr->protocol,ip_fd_ptr->tos,ip_fd_ptr-
>offset,ip_fd_ptr->traffic_class,ip_fd_ptr->src_internal_addr,ip_fd_ptr-
>dest_internal_addr,ip_fd_ptr->compression_method,o_size,ip_fd_ptr-
>options_field_set,ip_fd_ptr->tunnel_pkt_at_src,ip_fd_ptr->decompression_delay,ip_fd_ptr-
>next_addr.address,ip_fd_ptr->tunnel_end_addr.address,ip_fd_ptr-
>net_addr_tunneled.address,ip_fd_ptr->tunnel_ptr,ip_fd_ptr->tunnel_start_time,ip_fd_ptr-
>vpn_stamp_time,ip_fd_ptr->icmp_type,ip_fd_ptr->encap_count,ip_fd_ptr-
>ipv6_extension_hdr_info_ptr,ip_fd_ptr->src_num_hops_stat_hndl_ptr);




                                //      printf("ttl :%f %f %x %x %x\n",ip_fd_ptr-
>tunnel_start_time,NaN);



                                /*********************************/

                                /****CODIGO DE COPIA DE PAQUETE****/

                                /*********************************/
```

```
                    //printf("on_off : %d\n",on_off);//debug



                    if(on_off){

        /*************************************************
                    **********Copia Paquete IP**********
        *************************************************/


                    //printf("Dentro del IF(on_off)\n");//debug
                    pkt_ip_fake_ack = op_pk_copy (pkt_ip);
                    op_pk_nfd_access (pkt_ip_fake_ack, "fields", &ip_copy_fd_ptr);

                    ip_copy_fd_ptr->ident = ip_copy_fd_ptr->ident + 1;
                //  ip_copy_fd_ptr->ttl = ip_copy_fd_ptr->ttl;
                //  ip_copy_fd_ptr->src_addr = ip_copy_fd_ptr->dest_addr.address

                //  printf("ident: %d -> ",ip_fd_ptr->ident);
```

```
                        //      printf("%d\n",ip_copy_fd_ptr->ident);




                                        tmp
                = ip_copy_fd_ptr->src_addr;

                                        ip_copy_fd_ptr->src_addr
        = ip_copy_fd_ptr->dest_addr;

                                        ip_copy_fd_ptr->dest_addr
        = tmp;

                                        tmp_ip
                = ip_copy_fd_ptr->src_internal_addr;

                                        ip_copy_fd_ptr->src_internal_addr       =
ip_copy_fd_ptr->dest_internal_addr;

                                        ip_copy_fd_ptr->dest_internal_addr      =
tmp_ip;

                                        ip_copy_fd_ptr->next_addr
        = ip_copy_fd_ptr->dest_addr;

                        //              ip_copy_fd_ptr->connection_class        =
conn_class;

                        //              ip_copy_fd_ptr->orig_len
        = data_len;

                        //              ip_copy_fd_ptr->frag_len
        = data_len;

                        //              ip_copy_fd_ptr->ttl
= ttl;

                        //              ip_copy_fd_ptr->protocol
        = protocol_type;

                        //              ip_copy_fd_ptr->tos
        = type_of_service;
```

```
                    //          ip_copy_fd_ptr->compression_method          =
IpC_No_Compression;




                    //          ip_dgram_fields_set                (pkt_ip_fake_ack,
ip_copy_fd_ptr);

                    //          op_pk_nfd_access      (pkt_ip_fake_ack,      "fields",
&ip_copy_fd_ptr);

                    //          printf("src addr: %d -> ",ip_fd_ptr->src_addr);

                    //          printf("%d\n",ip_copy_fd_ptr->src_addr);


                    //          printf("dst addr: %d -> ",ip_fd_ptr->dest_addr);

                    //          printf("%d\n",ip_copy_fd_ptr->dest_addr);


                    //    printf("%d\n",(pkt_ip_fake_ack->fields)->(ip_fd_ptr->ident));

                    //          matlab_packet_travel(pkt_ip_fake_ack, POUT);

                    //          op_pk_send (pkt_ip_fake_ack, 1); // server_out

                    //          printf("packet bounced at time %f, on_off = %d\n",
op_sim_time(), on_off);

                              //op_pk_destroy(pkt_ip_fake_ack);



        /*****************************************************

                              **********FIN      Copia      Paquete
IP**********

        *****************************************************/
```

```
        /*************************************************
                                        **********Copia                    Paquete
TCP**********
        *************************************************/


                                        //op_pk_nfd_access    (pkt_tcp_fake_ack,    "fields",
&tcp_copy_fd_ptr);

                                        seq_server = tcp_copy_fd_ptr->seq_num;

                                        tmp_tcp
= tcp_copy_fd_ptr->src_port;
                                        tcp_copy_fd_ptr->src_port           =
tcp_copy_fd_ptr->dest_port;
                                        tcp_copy_fd_ptr->dest_port          = tmp_tcp;
                                        tmp_tcp
        = tcp_copy_fd_ptr->seq_num;
                                        tcp_copy_fd_ptr->seq_num            =
tcp_copy_fd_ptr->ack_num;
                                        ack_proxy_old                 = ack_proxy;

                                        if(seq_server == ack_proxy){
                                            tcp_copy_fd_ptr->ack_num  =
```

```
tmp_tcp+tcp_copy_fd_ptr->data_len;

                                                ack_proxy                       =
tcp_copy_fd_ptr->ack_num;



                                                //printf("IF
SEQ_NUM:\t%d\tACK_PROXY\t%d\n\n", seq_server, ack_proxy_old);



                                        }else{

                                                tcp_copy_fd_ptr->ack_num  = ack_proxy;

                                                //printf("ELSE
SEQ_NUM:\t%d\tACK_PROXY\t%d\n\n", seq_server, ack_proxy_old);

                                        }



                        //              tcp_copy_fd_ptr->rcv_win        =
tcp_copy_fd_ptr->rcv_win;

                        //              tcp_copy_fd_ptr->urgent_pointer =
tcp_copy_fd_ptr->urgent_pointer;

                                        tcp_copy_fd_ptr->data_len        = 0;

                                        tcp_copy_fd_ptr->flags                  = 0x10;

                        //              tcp_copy_fd_ptr->local_key      =
tcp_copy_fd_ptr->local_key;

                        //              tcp_copy_fd_ptr->remote_key             =
tcp_copy_fd_ptr->remote_key;



                                                        /***** version 2 *****/

                        /*

                                        tcp_copy_fd_ptr->src_port       =    tcp_fd_ptr-
>dest_port;
```

```
                                        tcp_copy_fd_ptr->dest_port          =    tcp_fd_ptr-
>src_port;

                                        tcp_copy_fd_ptr->seq_num            =    tcp_fd_ptr-
>ack_num;

                                        tcp_copy_fd_ptr->ack_num            =    tcp_fd_ptr-
>seq_num + tcp_copy_fd_ptr->data_len;

                              //        tcp_copy_fd_ptr->rcv_win            =
tcp_copy_fd_ptr->rcv_win;

                              //        tcp_copy_fd_ptr->urgent_pointer     =
tcp_copy_fd_ptr->urgent_pointer;

                                        tcp_copy_fd_ptr->data_len           = 0;

                                        tcp_copy_fd_ptr->flags                    = 0x10;

                              //        tcp_copy_fd_ptr->local_key          =
tcp_copy_fd_ptr->local_key;

                              //        tcp_copy_fd_ptr->remote_key               =
tcp_copy_fd_ptr->remote_key;

                              */



                              //        ip_dgram_fields_set                 (pkt_ip_fake_ack,
ip_copy_fd_ptr);

                              //        op_pk_nfd_access      (pkt_ip_fake_ack,     "fields",
&ip_copy_fd_ptr);









        /**************************************************if          (op_subq_empty
(QM)==OPC_FALSE){
```

```
                                        **********FIN Copia Paquete TCP**********

        **************************************************/



                                        //printf("NF data size: %d\n", op_pk_nfd_size (pkt_tcp_fake_ack, "data")/8);
                                        if (op_pk_nfd_size (pkt_tcp_fake_ack, "data") > 0){
                                                //op_pk_print(pkt_tcp_fake_ack);
                                                op_pk_nfd_get_pkt (pkt_tcp_fake_ack, "data", &trash);
                                                op_pk_destroy(trash);
                                                //op_pk_print(pkt_tcp_fake_ack);
                                                //printf("NF data size removed: %d\t, orig-len: %d\t, frag-len: %d\n", op_pk_nfd_size (pkt_tcp_fake_ack, "data")/8,ip_copy_fd_ptr->orig_len,ip_copy_fd_ptr->frag_len);
                                        }



                                        ip_copy_fd_ptr->orig_len
        = op_pk_total_size_get (pkt_tcp_fake_ack)/8;
                                        ip_copy_fd_ptr->frag_len
        = op_pk_total_size_get (pkt_tcp_fake_ack)/8;



                                        //printf("NF data size removed: %d\t, orig-len: %d\t, frag-len: %d\n", op_pk_nfd_size (pkt_tcp_fake_ack, "data")/8,ip_copy_fd_ptr->orig_len,ip_copy_fd_ptr->frag_len);
```

```
                                        // SE IMPRIME EN TCP



                                        fprintf (pFile_proxy_TCP, "src_port: %d\tdest_port: %d\tseq_num: %u\tack_num: %u\trcv_win: %u urgent_pointer: %d data_len: %d\tflags:",tcp_copy_fd_ptr->src_port,tcp_copy_fd_ptr->dest_port,tcp_copy_fd_ptr->seq_num,tcp_copy_fd_ptr->ack_num,tcp_copy_fd_ptr->rcv_win,tcp_copy_fd_ptr->urgent_pointer,tcp_copy_fd_ptr->data_len);

                                        print_flag(pFile_proxy_TCP, tcp_copy_fd_ptr->flags);

                                        fprintf (pFile_proxy_TCP, "\tlocal_key: %X remote_key: %X\n",tcp_copy_fd_ptr->local_key,tcp_copy_fd_ptr->remote_key);



                                        // SE TERMINA DE IMPRIMIR EN TCP



                                        o_len = ip_copy_fd_ptr->orig_len;

                                        f_len = ip_copy_fd_ptr->frag_len;

                                        o_size =ip_copy_fd_ptr->original_size;



                                        // SE IMPRIME EN IP

                                        fprintf(pFile_proxy_IP,"orig_len: %d\tident: %d\t frag_len: %d\t ttl: %u\tsrc_addr: %x\tdest_addr: %x\tprotocol: %d\t tos: %d\toffset: %d\ttraffic_class: %s\tsrc_internal_addr: %x\tdest_internal_addr: %x\tcompression_method: %d\toriginal_size: %d\toptions_field_set: %d\ttunnel_pkt_at_src: %d\tdecompression_delay: %f\tnext_addr: %x\ttunnel_end_addr: %x\tnet_addr_tunneled: %x\ttunnel_ptr: %d\ttunnel_start_time: %0.0f\tvpn_stamp_time: %f\ticmp_type: %x\tencap_count: %d\tipv6_extension_hdr_info_ptr: %d\tsrc_num_hops_stat_hndl_ptr %d
```

```
\n",o_len,ip_copy_fd_ptr->ident,f_len,ip_copy_fd_ptr->ttl,ip_copy_fd_ptr->src_addr.address,ip_copy_fd_ptr->dest_addr.address,ip_copy_fd_ptr->protocol,ip_copy_fd_ptr->tos,ip_copy_fd_ptr->offset,ip_copy_fd_ptr->traffic_class,ip_copy_fd_ptr->src_internal_addr,ip_copy_fd_ptr->dest_internal_addr,ip_copy_fd_ptr->compression_method,o_size,ip_copy_fd_ptr->options_field_set,ip_copy_fd_ptr->tunnel_pkt_at_src,ip_copy_fd_ptr->decompression_delay,ip_copy_fd_ptr->next_addr.address,ip_copy_fd_ptr->tunnel_end_addr.address,ip_copy_fd_ptr->net_addr_tunneled.address,ip_copy_fd_ptr->tunnel_ptr,ip_copy_fd_ptr->tunnel_start_time,ip_copy_fd_ptr->vpn_stamp_time,ip_copy_fd_ptr->icmp_type,ip_copy_fd_ptr->encap_count,ip_copy_fd_ptr->ipv6_extension_hdr_info_ptr,ip_copy_fd_ptr->src_num_hops_stat_hndl_ptr);

                                        // FIN PRINT IP









                                        if (seq_server != ack_proxy_old && semaforo){

                                        printf("Lost packet?\tseq_server= %d\tack_proxy= %d \n", seq_server, ack_proxy_old);

                                        semaforo=0;



                                        }else if(seq_server == ack_proxy_old && !semaforo){

                                                printf("llego el packet?\tseq_server= %d\tack_proxy= %d \n", seq_server, ack_proxy_old);

                                                semaforo=1;



                                        }
```

```
                                        //
**************************Seccion I de la PIZARRA*******************
                                        //printf("Inserto pkt en la queue\n");
                                        //op_pk_print (pkt_ip);

                                        pkt_queue_in = op_pk_copy (pkt_ip);

                                        if(semaforo){

                                                matlab_packet_queue(pkt_queue_in,
QM,1,TAIL);
                                                op_subq_pk_insert (QM, pkt_queue_in,
OPC_QPOS_TAIL);// Inserto Pkt en Queue principal
                                        //      printf("QM:%d SEQ_Server:%d
ACK_Proxy:%d\n", temp, seq_server, ack_proxy_old);
                                        }else{
                                                matlab_packet_queue(pkt_queue_in,
QD,1,TAIL);
                                                op_subq_pk_insert (QD, pkt_queue_in,
OPC_QPOS_TAIL);// Inserto Pkt en Queue DelayeD
                                        //      printf("\t\tQD:%d SEQ_Server:%d
ACK_Proxy:%d\n", temp, seq_server, ack_proxy_old);

                                        }
                                                //op_subq_print (0);
                                        //if (op_subq_empty (QD)==OPC_FALSE){
```

```
//      op_subq_print (QD);
//}
/******FIN COPIA DE PAQUETE******/

//B˙squeda, actualizaciÛn de ACK_proxy y de Qd y Qm

//printf("ANTES DEL WHILE (TRUE) \n");
while(1){

//      printf("ENTRO AL WHILE (TRUE) \n");
        solution = find();


        if(solution!=1){

                break;
        }


}
//printf("SALIO DEL WHILE (TRUE) \n");


//COPIAR LOS DATOS DE TCP MODIFICADOS
```

DENTRO DE LA COPIA

op_pk_nfd_set (pkt_ip_fake_ack, "data", pkt_tcp_fake_ack);

matlab_packet_travel(pkt_ip_fake_ack, POUT, SERVER);

op_pk_send(pkt_ip_fake_ack, SERVER);

//FIN DE LA COPIA DE DATOS

}

if(loss_on){

op_subq_pk_insert (QF, pkt_ip, OPC_QPOS_TAIL);// Inserto Pkt en Queue Forward

}else{

matlab_packet_travel(pkt_ip, POUT, WKSTN);

printf ("time: %4.15f, src_port: %d\tdest_port: %d\tseq_num: %u\tack_num: %u\trcv_win: %u urgent_pointer: %d data_len: %d\tflags:",op_sim_time(), tcp_fd_ptr->src_port,tcp_fd_ptr->dest_port,tcp_fd_ptr->seq_num,tcp_fd_ptr->ack_num,tcp_fd_ptr->rcv_win,tcp_fd_ptr->urgent_pointer,tcp_fd_ptr->data_len);

```
                                        print_flag_2(tcp_fd_ptr->flags);

                                        printf ( "\tlocal_key: %X remote_key: %X\n",tcp_fd_ptr->local_key,tcp_fd_ptr->remote_key);

                                        op_pk_send (pkt_ip, WKSTN);

                                    }




                    //          **********Seccion II de la PIZARRA**********


                    }else{

                        //printf ("Wkstn->Server\n");

                    //      fprintf (pFile, "Wkstn->Server: ");



                        // SE IMPRIME EN TCP

                                fprintf (pFile_wkstn_TCP, "src_port: %d\tdest_port: %d\tseq_num: %u\tack_num: %u\trcv_win: %u urgent_pointer: %d data_len: %d\tflags:",tcp_fd_ptr->src_port,tcp_fd_ptr->dest_port,tcp_fd_ptr->seq_num,tcp_fd_ptr->ack_num,tcp_fd_ptr->rcv_win,tcp_fd_ptr->urgent_pointer,tcp_fd_ptr->data_len);

                                print_flag(pFile_wkstn_TCP, tcp_fd_ptr->flags);

                                fprintf (pFile_wkstn_TCP, "\tlocal_key: %X remote_key: %X\n",tcp_fd_ptr->local_key,tcp_fd_ptr->remote_key);

                        // SE  TERMINA DE IMPRIMIR EN TCP



                        o_len = ip_fd_ptr->orig_len;

                        f_len = ip_fd_ptr->frag_len;

                        o_size = ip_fd_ptr->original_size;
```

```
// SE IMPRIME EN IP

fprintf(pFile_wkstn_IP,"orig_len: %d\tident: %d\t frag_len: %d\t ttl: %u\tsrc_addr: %x\tdest_addr: %x\tprotocol: %d\t tos: %d\toffset: %d\ttraffic_class: %s\tsrc_internal_addr: %x\tdest_internal_addr: %x\tcompression_method: %d\toriginal_size: %d\toptions_field_set: %d\ttunnel_pkt_at_src: %d\tdecompression_delay: %f\tnext_addr: %x\ttunnel_end_addr: %x\tnet_addr_tunneled: %x\ttunnel_ptr: %d\ttunnel_start_time: %0.0f\tvpn_stamp_time: %f\ticmp_type: %x\tencap_count: %d\tipv6_extension_hdr_info_ptr: %d\tsrc_num_hops_stat_hndl_ptr %d \n",o_len,ip_fd_ptr->ident,f_len,ip_fd_ptr->ttl,ip_fd_ptr->src_addr.address,ip_fd_ptr->dest_addr.address,ip_fd_ptr->protocol,ip_fd_ptr->tos,ip_fd_ptr->offset,ip_fd_ptr->traffic_class,ip_fd_ptr->src_internal_addr,ip_fd_ptr->dest_internal_addr,ip_fd_ptr->compression_method,o_size,ip_fd_ptr->options_field_set,ip_fd_ptr->tunnel_pkt_at_src,ip_fd_ptr->decompression_delay,ip_fd_ptr->next_addr.address,ip_fd_ptr->tunnel_end_addr.address,ip_fd_ptr->net_addr_tunneled.address,ip_fd_ptr->tunnel_ptr,ip_fd_ptr->tunnel_start_time,ip_fd_ptr->vpn_stamp_time,ip_fd_ptr->icmp_type,ip_fd_ptr->encap_count,ip_fd_ptr->ipv6_extension_hdr_info_ptr,ip_fd_ptr->src_num_hops_stat_hndl_ptr);

// SE TERMINA DE IMPRIMIR EN IP



//printf("ttl :%f %f %x %x %x\n",ip_fd_ptr->tunnel_start_time,NaN);

//op_pk_print (pkt_ip);



if (on_off){


// **********Seccion I de la PIZARRA**********

//op_subq_print (0);

if (op_subq_empty (QM)==OPC_FALSE){
```

```
                                                    //printf("ENTRO AL IF para sacar pkt de la queue\n");
                                                    //op_subq_print (QM);



                                                    /***********Bloque para obtener el SEQ de la cabeza del queue (Aumentar el SEQ)*******/


                                                    pkt_queue_out = op_subq_pk_remove (QM, OPC_QPOS_HEAD); //op_subq_pk_remove( id_QUEUE, posiciÛn en el QUEUE);
                                                    op_pk_nfd_get (pkt_queue_out, "data", &pkt_queue_out_tcp);
                                                    op_pk_nfd_access (pkt_queue_out_tcp, "fields", &queue_out_tcp_fd_ptr);
                                                    seq_qout = queue_out_tcp_fd_ptr->seq_num;  // SEQ DEL PAQUETE EN EL QUEUE
                                                    len_qout = queue_out_tcp_fd_ptr->data_len; // SEQ DEL PAQUETE EN EL QUEUE
                                                    op_pk_nfd_set (pkt_queue_out, "data", pkt_queue_out_tcp);
                                                    matlab_packet_queue(pkt_queue_out, QM,0,HEAD);

        /**********************************FIN BLOQUE**********************/


                                                    ack_wkstn = tcp_fd_ptr->ack_num; // ACK DEL PAQUETE QUE LLEGO

        printf("**************************************\nACK recibido: %d\n*********************************************\n", ack_wkstn);
```

```
                                        if(ack_wkstn==old_ack){


                                                cont_ack++;

                                                if(cont_ack==1){
                //SINGLE DUPLICATE ACK

                                                        pkt_queue_retransmit =
op_pk_copy (pkt_queue_out);


        op_pk_send(pkt_queue_retransmit,DIRECT);                        //ESTE    HAY
QUE TRANSMITIRLO SEND();

                                                        loss_on=1;




        printf("**************************************\n\n\n\n\nACK                recibido:
%d\nPaquete                        Retransmitido\nSEQ                enviado:
%d\n\n\n\n\n**********************************************\n", ack_wkstn, seq_qout);



                                                }

                                        }else{


                                                if(loss_on){

                                                        loss_on=0;

                                                        while(          (op_subq_empty
(QF)==OPC_FALSE)){




                                                                pkt_fwd         =
op_subq_pk_remove (QF, OPC_QPOS_HEAD); //op_subq_pk_remove( id_QUEUE, posiciÛn
en el QUEUE);

                                                                op_pk_nfd_get
```

```
(pkt_fwd, "data", &pkt_fwd_tcp);

                                                                    op_pk_nfd_access
(pkt_fwd_tcp, "fields", &fwd_tcp_fd_ptr);

                                                                    seq_qfwd          =
fwd_tcp_fd_ptr->seq_num;          // SEQ DEL PAQUETE EN EL QUEUE

                                                                    op_pk_nfd_set
(pkt_fwd, "data", pkt_fwd_tcp);

                                                                    printf("\nSeq_fwd:
%d\n",seq_qfwd);


        op_pk_send(pkt_fwd,WKSTN);                          //ESTE      HAY      QUE
TRANSMITIRLO SEND();

                                                                    }

                                                            }

                                                            old_ack=ack_wkstn;

                                                            cont_ack=0;

                                                    }


                                            while(ack_wkstn  >=  (seq_qout+len_qout)  &&
(op_subq_empty (QM)==OPC_FALSE)){




                                                            /*if    (ack_wkstn    ==    seq_qout){
//CONDICION IGUALDAD AL COMIENZO.




                                                                    pkt_queue_retransmit        =
op_pk_copy (pkt_queue_out);
```

```
        op_pk_send(pkt_queue_retransmit,WKSTN);                    //ESTE HAY
QUE TRANSMITIRLO SEND();

                                                        break;

                                                }*/


                                                op_pk_destroy(pkt_queue_out);


                                                /***********Bloque para obtener el
SEQ de la cabeza del queue (Aumentar el SEQ)*******/

                                                pkt_queue_out                =
op_subq_pk_remove (QM, OPC_QPOS_HEAD); //op_subq_pk_remove( id_QUEUE, posiciÛn
en el QUEUE);

                                                op_pk_nfd_get        (pkt_queue_out,
"data", &pkt_queue_out_tcp);

                                                op_pk_nfd_access
(pkt_queue_out_tcp, "fields", &queue_out_tcp_fd_ptr);

                                                seq_qout = queue_out_tcp_fd_ptr-
>seq_num;          // SEQ DEL PAQUETE EN EL QUEUE

                                                op_pk_nfd_set        (pkt_queue_out,
"data", pkt_queue_out_tcp);

                                                matlab_packet_queue(pkt_queue_out,
QM,0,HEAD);

        /**********************************FIN BLOQUE**********************/




                                        }

                                        op_subq_pk_insert (QM,    pkt_queue_out,
OPC_QPOS_HEAD);// Re-Inserto Pkt en QM
```

```
                                        matlab_packet_queue(pkt_queue_out,
QM,1,HEAD);



                        }
                        op_pk_destroy(pkt_ip);



                }else{  // else del if(on_off)

                //      op_pk_print(pkt_ip);

                        matlab_packet_travel(pkt_ip, POUT, SERVER);
                        op_pk_send(pkt_ip, SERVER);

                }


        }

        }
        FSM_PROFILE_SECTION_OUT (state2_enter_exec)
```

```
/** state (pkt_info) exit executives **/

FSM_STATE_EXIT_FORCED (2, "pkt_info", "proxy [pkt_info exit execs]")




/** state (pkt_info) transition processing **/

FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "pkt_info", "idle", "tr_3", "proxy [pkt_info -> idle : default / ]")

/*--------------------------------------------------------*/




/** state (close) enter executives **/

FSM_STATE_ENTER_UNFORCED (3, "close", state3_enter_exec, "proxy [close enter execs]")

FSM_PROFILE_SECTION_IN ("proxy [close enter execs]", state3_enter_exec)

{

fclose (pFile_wkstn_IP);

fclose (pFile_wkstn_TCP);

fclose (pFile_proxy_IP);

fclose (pFile_proxy_TCP);

fclose (matlab_anim);

}

FSM_PROFILE_SECTION_OUT (state3_enter_exec)
```

```
                /** blocking after enter executives of unforced state. **/
                FSM_EXIT (7,"proxy")



                /** state (close) exit executives **/
                FSM_STATE_EXIT_UNFORCED (3, "close", "proxy [close exit execs]")



                /** state (close) transition processing **/
                FSM_TRANSIT_MISSING ("close")
                        /*---------------------------------------------------------*/




                }



        FSM_EXIT (0,"proxy")
        }
}
```

```
void
_op_proxy_diag (OP_SIM_CONTEXT_ARG_OPT)
	{
	/* No Diagnostic Block */
	}




void
_op_proxy_terminate (OP_SIM_CONTEXT_ARG_OPT)
	{

	FIN_MT (_op_proxy_terminate ())



	/* No Termination Block */

	Vos_Poolmem_Dealloc (op_sv_ptr);

	FOUT
	}
```

```
/* Undefine shortcuts to state variables to avoid */
/* syntax error in direct access to fields of */
/* local variable prs_ptr in _op_proxy_svar function. */
#undef pkt_ip
#undef port
#undef pkt_tcp
#undef tcp_fd_ptr
#undef pFile_wkstn_IP
#undef ip_fd_ptr
#undef ip_copy_fd_ptr
#undef pkt_ip_fake_ack
#undef pkt_tcp_fake_ack
#undef on_off
#undef tmp
#undef pFile_proxy_IP
#undef pFile_wkstn_TCP
#undef pFile_proxy_TCP
#undef tcp_copy_fd_ptr
#undef tmp_tcp
#undef tmp_ip
#undef trash
#undef pkt_queue_out
#undef pkt_queue_in
#undef ack_wkstn
```

```
#undef seq_qout

#undef pkt_queue_out_tcp

#undef queue_out_tcp_fd_ptr

#undef ack_proxy

#undef seq_server

#undef semaforo

#undef ack_ptr

#undef pkt_queue_out_find

#undef solution

#undef ack_proxy_old

#undef matlab_anim

#undef matlab_queue

#undef pFile_close

#undef pkt_queue_retransmit

#undef len_qout

#undef cont_ack

#undef old_ack

#undef loss_on

#undef pkt_fwd

#undef pkt_fwd_tcp

#undef fwd_tcp_fd_ptr

#undef seq_qfwd


#undef FIN_PREAMBLE_DEC
```

```
#undef FIN_PREAMBLE_CODE


#define FIN_PREAMBLE_DEC

#define FIN_PREAMBLE_CODE


VosT_Obtype

_op_proxy_init (int * init_block_ptr)

        {

        VosT_Obtype obtype = OPC_NIL;

        FIN_MT (_op_proxy_init (init_block_ptr))


        obtype = Vos_Define_Object_Prstate ("proc state vars (proxy)",

                sizeof (proxy_state));

        *init_block_ptr = 0;


        FRET (obtype)

        }


VosT_Address

_op_proxy_alloc (VosT_Obtype obtype, int init_block)

        {

#if !defined (VOSD_NO_FIN)

        int _op_block_origin = 0;

#endif
```

```
	proxy_state * ptr;
	FIN_MT (_op_proxy_alloc (obtype))

	ptr = (proxy_state *)Vos_Alloc_Object (obtype);
	if (ptr != OPC_NIL)
		{
		ptr->_op_current_block = init_block;
#if defined (OPD_ALLOW_ODB)
		ptr->_op_current_state = "proxy [init enter execs]";
#endif
		}
	FRET ((VosT_Address)ptr)
	}




void
_op_proxy_svar (void * gen_ptr, const char * var_name, void ** var_p_ptr)
	{
	proxy_state		*prs_ptr;

	FIN_MT (_op_proxy_svar (gen_ptr, var_name, var_p_ptr))

	if (var_name == OPC_NIL)
```

```
        {
        *var_p_ptr = (void *)OPC_NIL;
        FOUT
        }
prs_ptr = (proxy_state *)gen_ptr;


if (strcmp ("pkt_ip" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pkt_ip);
        FOUT
        }
if (strcmp ("port" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->port);
        FOUT
        }
if (strcmp ("pkt_tcp" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pkt_tcp);
        FOUT
        }
if (strcmp ("tcp_fd_ptr" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->tcp_fd_ptr);
```

```
        FOUT
        }
    if (strcmp ("pFile_wkstn_IP" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pFile_wkstn_IP);
        FOUT
        }
    if (strcmp ("ip_fd_ptr" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->ip_fd_ptr);
        FOUT
        }
    if (strcmp ("ip_copy_fd_ptr" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->ip_copy_fd_ptr);
        FOUT
        }
    if (strcmp ("pkt_ip_fake_ack" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pkt_ip_fake_ack);
        FOUT
        }
    if (strcmp ("pkt_tcp_fake_ack" , var_name) == 0)
        {
```

```
            *var_p_ptr = (void *) (&prs_ptr->pkt_tcp_fake_ack);
            FOUT
            }
    if (strcmp ("on_off" , var_name) == 0)
            {
            *var_p_ptr = (void *) (&prs_ptr->on_off);
            FOUT
            }
    if (strcmp ("tmp" , var_name) == 0)
            {
            *var_p_ptr = (void *) (&prs_ptr->tmp);
            FOUT
            }
    if (strcmp ("pFile_proxy_IP" , var_name) == 0)
            {
            *var_p_ptr = (void *) (&prs_ptr->pFile_proxy_IP);
            FOUT
            }
    if (strcmp ("pFile_wkstn_TCP" , var_name) == 0)
            {
            *var_p_ptr = (void *) (&prs_ptr->pFile_wkstn_TCP);
            FOUT
            }
    if (strcmp ("pFile_proxy_TCP" , var_name) == 0)
```

```
        {
        *var_p_ptr = (void *) (&prs_ptr->pFile_proxy_TCP);
        FOUT
        }
    if (strcmp ("tcp_copy_fd_ptr" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->tcp_copy_fd_ptr);
        FOUT
        }
    if (strcmp ("tmp_tcp" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->tmp_tcp);
        FOUT
        }
    if (strcmp ("tmp_ip" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->tmp_ip);
        FOUT
        }
    if (strcmp ("trash" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->trash);
        FOUT
        }
```

```
if (strcmp ("pkt_queue_out" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pkt_queue_out);
        FOUT
        }
if (strcmp ("pkt_queue_in" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pkt_queue_in);
        FOUT
        }
if (strcmp ("ack_wkstn" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->ack_wkstn);
        FOUT
        }
if (strcmp ("seq_qout" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->seq_qout);
        FOUT
        }
if (strcmp ("pkt_queue_out_tcp" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pkt_queue_out_tcp);
        FOUT
```

```
        }
    if (strcmp ("queue_out_tcp_fd_ptr" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->queue_out_tcp_fd_ptr);
        FOUT
        }
    if (strcmp ("ack_proxy" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->ack_proxy);
        FOUT
        }
    if (strcmp ("seq_server" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->seq_server);
        FOUT
        }
    if (strcmp ("semaforo" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->semaforo);
        FOUT
        }
    if (strcmp ("ack_ptr" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->ack_ptr);
```

```
        FOUT
        }
    if (strcmp ("pkt_queue_out_find" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pkt_queue_out_find);
        FOUT
        }
    if (strcmp ("solution" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->solution);
        FOUT
        }
    if (strcmp ("ack_proxy_old" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->ack_proxy_old);
        FOUT
        }
    if (strcmp ("matlab_anim" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->matlab_anim);
        FOUT
        }
    if (strcmp ("matlab_queue" , var_name) == 0)
        {
```

```
        *var_p_ptr = (void *) (&prs_ptr->matlab_queue);
        FOUT
        }
    if (strcmp ("pFile_close" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pFile_close);
        FOUT
        }
    if (strcmp ("pkt_queue_retransmit" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pkt_queue_retransmit);
        FOUT
        }
    if (strcmp ("len_qout" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->len_qout);
        FOUT
        }
    if (strcmp ("cont_ack" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->cont_ack);
        FOUT
        }
    if (strcmp ("old_ack" , var_name) == 0)
```

```
        {
        *var_p_ptr = (void *) (&prs_ptr->old_ack);
        FOUT
        }
    if (strcmp ("loss_on" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->loss_on);
        FOUT
        }
    if (strcmp ("pkt_fwd" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pkt_fwd);
        FOUT
        }
    if (strcmp ("pkt_fwd_tcp" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pkt_fwd_tcp);
        FOUT
        }
    if (strcmp ("fwd_tcp_fd_ptr" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->fwd_tcp_fd_ptr);
        FOUT
        }
```

```
if (strcmp ("seq_qfwd" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->seq_qfwd);
        FOUT
        }
*var_p_ptr = (void *)OPC_NIL;

FOUT
}
```