UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

"DISEÑO DE UN BANCO DE PRUEBAS PARA ESTUDIAR EL COMPORTAMIENTO DEL PROTOCOLO DE TRANSPORTE ESTP Y OTROS PROTOCOLOS TCP QUE EMPLEAN UN ALGORITMO AIMD"

TESIS PARA OPTAR EL GRADO DE MAGÍSTER EN

INGENIERÍA DE REDES DE COMUNICACIONES

JESÚS MARTÍN BRAVO SUCLUPE

PROFESOR GUÍA:

CLAUDIO IGNACIO ESTÉVEZ MONTERO

MIEMBROS DE LA COMISIÓN:

ALBERTO JOSUÉ CASTRO ROJAS

RODRIGO ANDRÉS ARENAS ANDRADE

SANTIAGO DE CHILE

2016

RESUMEN

Transmission Control Protocol (TCP) es un protocolo de transporte cuyo rendimiento depende del algoritmo de control de congestión empleado, el cual modifica el comportamiento de la ventana de congestión, la cual es la cantidad de paquetes que es posible enviar antes de ser reconocidos.

Al emplear TCP tradicional, al comenzar una transmisión, la ventana de congestión enviada tiene un crecimiento determinado por un "slow-start" (inicio lento), y continúa con una fase de evitación de congestión que emplea un algoritmo AIMD: incremento aditivo de 1 paquete por cada ventana reconocida y un decremento multiplicativo medio que reduce la ventana enviada a la mitad de su valor cuando se produce una congestión. Cuando una congestión es producida por "timeout" (tiempo de espera agotado), la ventana de congestión es reducida a 1, y luego se realiza un "slow-start" y continúa con la etapa evitación de la congestión.

General Additive Increase-Multiplicative Decrease (GAIMD) es una variante de TCP, que modifica el comportamiento del incremento aditivo en un valor de "α" paquetes por cada ventana enviada reconocida y un decremento "β" que mutliplica el valor de la ventana enviada, cuando se produce una congestión. GAIMD es amistoso cuando se establecen los valores de $\alpha=0.31$ y $\beta=0.875$, y desprecia "timeouts" producidos para el cálculo de su rendimiento, el cuál teóricamente es mayor que el rendimiento en TCP tradicional.

Ethernet Services Transport Protocol (ESTP) es un protocolo de transporte diseñado para transmisiones sobre redes Ethernet que modifica sólo el decremento multiplicativo (β) de TCP tradicional, sin embargo β tiene un valor variable, el cual se comporta de acuerdo a la cantidad de paquetes transmitidos entre dos pérdidas (δ). Para el cálculo de β, ESTP emplea una función con un componente exponencial cuya variable principal es δ.

Para la evaluación del rendimiento de ESTP y las otras variantes de TCP que emplean un algoritmo AIMD se requiere que cada protocolo sea implementado en el código fuente del sistema operativo y el empleo de herramientas de generación de tráfico como iPerf, herramientas de captura de información de la transmisión como lo son tshark y tcpprobe, utilidades para establecer reglas en el tráfico como netem e iptables, y de software que interprete la información capturada y grafique el comportamiento de la ventana de congestión y rendimiento de la transmisión.

Haciendo uso de las herramientas y software mencionados, se diseña y construye un banco de pruebas que permita evaluar el rendimiento de los protocolos de transporte, con el objetivo de contrastar y mejorar el comportamiento de los mismos.

DEDICATORIA

A mis padres y hermanos quienes son mis grandes amigos y me han brindado su apoyo en cada decisión importante en mi vida.

AGRADECIMIENTO

Primeramente, a Dios por los momentos de vida que me brinda, entre ellos el hacer posible la realización de una de mis metas, al culminar satisfactoriamente mi maestria.

A mis padres Martín Walter Jesús Bravo Llaque y Felícita Suclupe Chanamé por su apoyo incondicional en cada uno de los momentos significativos para mi persona y cada objetivo que me he planteado.

A mis hermanos Ángel y César, y de modo impersonal a cada uno de mis seres queridos, por su confianza y estímulo brindados en esta etapa personal.

Al Programa Nacional de Becas y Crédito Educativo de mi país, por haberme ofrecido la oportunidad de cursar mis estudios de posgrado en la Universidad de Chile.

A a mi profesor guía de tesis Claudio Estévez Montero, por su confianza brindada al aceptarme como uno de sus estudiantes, por haber compartido con mi persona sus conocimientos y experiencia profesional, y por cada uno de sus consejos que me han permitido una mejora constante durante el tiempo que he cursado la maestría.

A mi profesor co-guía Alberto Castro Rojas, por su apoyo en la culminación de mi tesis, orientándome para obtener un mejor trabajo final.

Al ingeniero Rodrigo Arenas Andrade, por haber aceptado integrar la Comisión de evaluación de mi tesis.

Al claustro académico de profesores del MIRC, que han impartido su conocimiento y compartido sus experiencias con mi persona.

Finalmente, a mis compañeros de estudio con quienes he compartido además del aprendizaje en clase, momentos buenos de amistad, en especial a César Cárdenas Lino, Pablo Flores Aravena, Diego Gonzalez Betancourth, Daniel Orellana Guerrero y Johanna Ortega Briones.

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

1. ANTECEDENTES

El stack de protocolos más empleado sobre Internet en la actualidad es "Transmission Control Protocol/Internet Protocol" (TCP/IP). TCP proporciona un servicio de flujo de bytes confiable orientado a la conexión, lo cual significa que el duo transmisor-receptor debe establecer una conexión TCP poniéndose en contacto entre sí antes de que puedan intercambiar datos [1] [2] [3].

Tiempo posterior al despliegue de TCP, se empezó a experimentar congestión en el transporte, entonces, junto con ello surge el concepto de "ventana" como la colección de paquetes que han sido inyectados por el emisor, pero aún no han sido completamente reconocidos por el receptor [4] [1]. El control de la ventana de congestión en TCP propuesto inicialmente, es conocido como "TCP Tahoe" o simplemente TCP, por Jacobson en 1988, en el cual se exponen dos técnicas: "slow start" (inicio lento) y "congestion avoidance" (evitación de la congestión) [4] [5]. Luego han surgido diversas implementaciones como Reno, Reno Fast-retransmission, SACK, BIC, CUBIC, entre otras, con el objetivo de obtener un mayor rendimiento, y mejorar el uso del ancho de banda, siendo el problema de TCP que su algoritmo de control de congestión no permite flujos para obtener un ancho de banda completo disponible sobre enlaces de larga distancia.

Implementaciones que se destacan son General Incremento aditivo-decremento multiplicativo (GAIMD), que busca establecer un protocolo de transporte amistoso con un mayor rendimiento [6] y Protocolo de Transporte de Servicios Ethernet (ESTP), que establece un control de congestión el cual se comporta de acuerdo a la cantidad de datos enviados entre pérdidas [7]. El objetivo de las implementaciones de control de congestión es el de mejorar el rendimiento u otra característica del protocolo. Para ello es necesario el uso de herramientas que permitan desarrollar un banco de pruebas para realizar una correcta evaluación del rendimiento de los protocolos de transporte.

2. HIPÓTESIS

Utilizando una red de computadores con módulos configurables es posible estudiar con gran detalle el comportamiento de distintos protocolos de transporte, en particular ESTP, el de mayor interés en este estudio. El banco de prueba es capaz de obtener datos como ventana de congestión, velocidad de transmisión, pendiente de crecimiento, y otros parámetros de interés. Estos instrumentos ayudarán a diseñar nuevos protocolos de transporte en una forma más eficaz y expedita.

3. OBJETIVOS

Objetivo General: Diseñar un banco de pruebas que entregue información objetiva sobre el comportamiento de protocolos de transporte y comparar el desempeño de ESTP con otros protocolos.

Objetivos específicos:

a) Realizar un estudio teórico de los protocolos de transporte ESTP, GAIMD y TCP.
b) Implementar los protocolos de transporte ESTP, GAIMD y TCP en Linux.
c) Diseñar e implementar un banco de pruebas para la evaluación de protocolos de transporte.
d) Evaluar el comportamiento de ESTP, GAIMD y TCP haciendo uso del banco de pruebas.

4. METODOLOGIA

En el presente trabajo de investigación se emplea estudio teórico, experimentación y análisis.

a) Estudio teórico: Se realiza un entendimiento teórico del comportamiento de la ventana de congestión en los protocolos de transportes TCP (Protocolo de Control de Transporte), GAIMD (General Incremento aditivo – decremento multiplicativo) y ESTP (Protocolo de Transporte de Servicios Ethernet), de igual modo se estudia el rendimiento de los protocolos mencionados. Este estudio teórico se realiza en base a publicaciones anteriormente realizadas.

b) Experimentación: Se implementa un banco de pruebas con ordenadores con distribución libre de sistema operativo Linux, los cuales cuentan con el código fuente de TCP, y se implementan TCP, GAIMD y ESTP en el código fuente de estos ordenadores. El comportamiento de la ventana de congestión es capturado utilizando las herramientas iperf [8], tcpprobe [9], iptables [10] y tshark (wireshark) [11]. Luego se emplea el software Matlab, ejecutándose un programa para analizar el comportamiento de la ventana de congestión y el rendimiento de los protocolos de transporte.

c) Análisis: A partir de los valores obtenidos por iPerf y procesados por Matlab, se realiza un análisis en base a los gráficos obtenidos y estadística descriptiva.

Se emplea R para contrastar supuestos teóricos que comparan el rendimiento de un protocolo con otro (Por ejemplo: contrastar si ESTP tiene un mejor rendimiento que TCP tradicional).

5. RESULTADOS ESPERADOS

Con el banco de pruebas diseñado debe ser posible realizar el correcto análisis, evaluación y validación de los protocolos de transporte implementados, mostrando gráficamente el comportamiento de la ventana de congestión y del rendimiento en el tiempo. También, debe permitir a investigaciones posteriores que empleen este banco de pruebas optimizar los protocolos, agilizando trabajos futuros. El proyecto elaborado sirve como guía para futuras implementaciones y mejoras.

CAPITULO I: PROTOCOLO DE CONTROL DE TRANSMISIÓN (TCP)

1. GENERALIDADES DE TCP

1.1. Definición de TCP

TCP es un protocolo de transporte cuyo propósito primario es proveer una conexión confiable y segura entre pares de procesos [2]. Éste es un protocolo de transporte para la suite de internet, usado por aquellas aplicaciones que necesitan un servicio de transporte confiable orientado a la conexión (Ejemplo: FTP, SMTP y Telnet) [12]. Dos conceptos base de TCP, dados en [2], [12] y [1] son: Número de secuencia y ACK. El primero es un identificador numérico que se envía por cada byte transmitido, y permite al receptor ordenar los segmentos recibidos. El segundo término deriva de "acknologement" (significa: reconocimiento), y es un mensaje de reconocimiento por parte del receptor, el cual confirma la recepción de la información al transmisor. Si el ACK no es recibido por el transmisor durante un tiempo de expiración ("timeout"), la información es retransmitida.

La operación básica de TCP se da en las siguientes áreas descritas en [2]: Transferencia de datos, confiabilidad, control de flujo, multiplexación, conexión, precedencia y seguridad (Ver Tabla 1).

Característica de TCP	Descripción
Transferencia de datos	TCP transfiere flujos de octetos de bits (bytes) y los agrupa en segmentos para su transmisión a través de una red o sistema de internet.
Confiabilidad	TCP debe recuperarse de información que es dañada, perdida, duplicada o entregada fuera de orden por el sistema de comunicación (Capa de red de modelo OSI o Internet). Esta acción se da gracias al establecimiento de un número de secuencia por cada byte transmitido y al ACK dado por el receptor.
Control de flujo	El receptor indica la cantidad de datos máxima que puede ser enviada por el transmisor, al retornar un valor de "ventana" con cada ACK. La ventana es el número de bytes que se puede transmitir antes de que reciba un futuro permiso.
Multiplexación	Un host puede usar varias comunicaciones TCP simultáneamente, empleando un número de puerto por cada conexión, el cual concatena con una dirección de capa de red. Esta concatenación se denomina "socket".
Conexión	Una conexión TCP es especificada por un par de "sockets", y es desplegado en un sistema de red no confiable, entre hosts no confiables, por esta razón usa un mecanismo "handshake" con números de secuencia basados en temporizador para evitar una inicialización errónea de la conexión
Precedencia y seguridad	Los usuarios de TCP indicarán el nivel de seguridad y precedencia en la comunicación.

Tabla 1. Características de TCP [2] [1].

TCP muestra relación con otros protocolos, tal y como se muestra en la Figura 1.

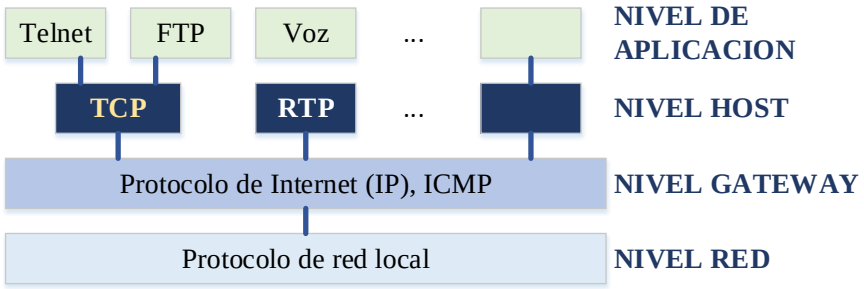


Figura 1. Relación de TCP con otros protocolos [2].

1.2. Modo básico de operación de TCP

Se describe en la Tabla 2:

Acción	Capa de modelo OSI
La información a ser transmitida se almacena en búferes y se agrupa en segmentos. Se llama al módulo de internet para transmitir cada segmento al destino TCP.	Transporte
El módulo de internet empaqueta los segmentos TCP en datagramas y los encamina al módulo de internet del destino (o Gateway intermedio).	Red
Para transmitir el datagrama a través de una red local, el datagrama debe ser envuelto en un paquete de red local, conocido como "trama".	Enlace
En un "gateway" entre redes, el datagrama de internet es "desenvuelto" desde la trama y examinado para determinar a cual red el datagrama de internet debe viajar (siguiente salto). El datagrama de internet es entonces "envuelto" en una nueva trama, compatible con la siguiente red y encaminada al siguiente "gateway" o destino final.	Red Enlace
Un "gateway" es permitido para romper un datagrama de internet en fragmentos de datagrama de internet más pequeños si es necesario. El formato de cada fragmento de datagrama permite que los fragmentos sean reensamblados en el módulo de internet destino.	Red
Los conmutadores de paquetes ("switches") realizan empaquetamiento (en tramas), fragmentación, u otra operación para lograr entregar la información a su destino.	Enlace
Un módulo de internet destino desenvuelve el segmento desde el datagrama y pasa este al destino TCP.	Red
El receptor TCP localiza los datos del segmento en el búfer del receptor y notifica al usuario receptor.	Transporte

Tabla 2. Modo de operación de TCP [2] [13].

El modelo mostrado en la Tabla 2 es simple y oculta muchos detalles, entre ellos: La información de control que incluye TCP (Número de secuencia, ACK, ventana, etc.), así como las utilidades de un datagrama de capa de red.

1.3. Cabecera TCP

Los dos primeros campos de la cabecera TCP identifican los puertos de origen y destino, cada uno con una longitud de 16 bits. Otros campos importantes son: el número de secuencia que identifica el primer byte del flujo de datos enviado, número de reconocimiento (ACK) que identifica el siguiente número de secuencia del segmento que se espera recibir del otro extremo y la ventana que es la cantidad de bytes que el extremo esta dispuesto a recibir. La estructura y cada uno de los campos de esta cabecera se caracterizan en [2] y son descritos en el Anexo A.

1.4. Funcionamiento de TCP

Una conexión TCP se define como un cuarteto, que consiste en dos direcciones IP y dos números de puerto. Es decir, existen dos extremos, donde cada extremo se identifica por un par (dirección IP, número de puerto) [1], cada par es también denominado socket [2]. Esta conexión normalmente pasa por tres fases: establecimiento, transferencia de datos (cuando la conexión ya está establecida) y el cierre de la conexión [1]. Conceptos importantes como número de secuencia y acuse de recibo son importantes en el funcionamiento de una conexión TCP.

1.4.1. Número de secuencia y acuse de recibo (ACK).

El número de secuencia y acuse de recibo, este último conocido también como "número de reconocimiento", son campos de la cabecera de TCP que se encuentran inter-relacionados. Un número de secuencia identifica a un byte, y cada byte integra un segmento, mientras que un acuse de recibo es un número representado como "SEG.ACK", que identifica que el número de secuencia del último byte recibido fue "SEG.ACK-1". El número de secuencia de una cabecera se identifica como "SEQ". No es común que el número ACK que envía un receptor incremente en una unidad en comparación al ACK anterior enviado por el mismo, debido a que este incrementará en el número de bytes que contiene el segmento reconocido.

Para un mejor entendimiento, se emplea la siguiente nomenclatura, definida en [2]:
SND.UNA = el menor número de secuencia sin acuse de recibo.
SND.NXT = número de secuencia del próximo envío.
SEG.ACK = acuse de recibo procedente del receptor (próximo número de secuencia esperado).
SEG.SEQ = primer número de secuencia de un segmento.
SEG.LEN = el número de bytes ocupados por los datos en el segmento (incluyendo SYN y FIN).
SEG.SEQ+SEG.LEN-1 = último número de secuencia de un segmento.

En condiciones de operación normales, la inter-relación del número de secuencia y acuse de recibo se muestra en la figura 2.

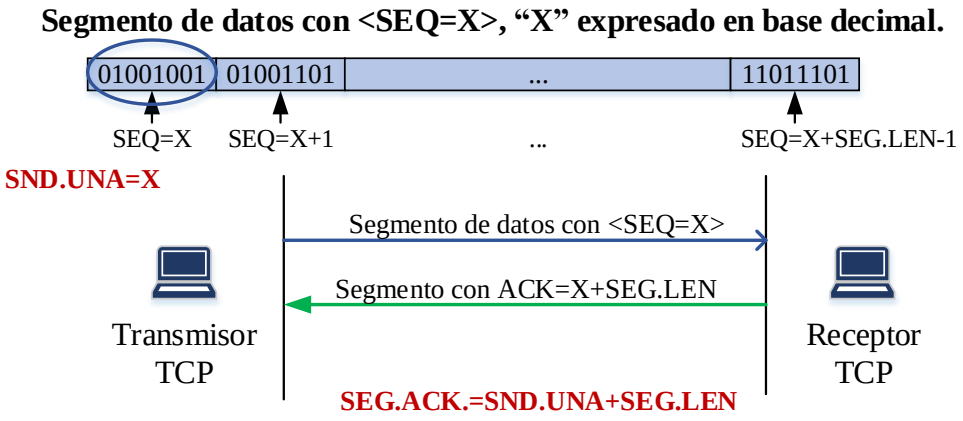


Figura 2. Número de secuencia y ACK (Elaboración propia, basado en [2]).

Un nuevo acuse de recibo es "aceptable" si cumple con la siguiente desigualdad: SND.UNA < SEG.ACK ≤ SND.NXT. Un segmento ubicado en la cola de retransmisión queda completamente confirmado por un acuse de recibo si la suma de su número de secuencia inicial y su longitud es menor o igual que el valor del acuse de recibo indicado en el segmento entrante [2].

Con respecto al lado del receptor, en [2], se emplea la siguiente nomenclatura:

RCV.NXT = siguiente número de secuencia esperado de los segmentos entrantes
(Límite inferior de la ventana de recepción).

RCV.NXT+RCV.WND-1 = último número de secuencia esperado en un segmento entrante
(Límite superior de la ventana de recepción).

SEG.SEQ = primer número de secuencia ocupado por el segmento entrante

SEG.SEQ+SEG.LEN-1 = último número de secuencia ocupado por el segmento entrante

En [2], menciona que debido a la existencia de ventanas y segmentos de tamaño cero, se tiene 4 posibilidades a considerar a la hora del segmento entrante como una posibilidad válida:

a) Longitud de segmento = 0 y Ventana de recepción =0.
SEG.SEQ = RCV.NXT

b) Longitud de segmento = 0 y Ventana de recepción >0.
RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND

c) Longitud de segmento > 0 y Ventana de recepción =0.
No aceptable.

d) Longitud de segmento > 0 y Ventana de recepción >0.
RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND o RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND.

2. CONGESTIÓN EN TCP.

2.1. Definición de ventana de congestión (cwnd).

La ventana de congestión es una variable de estado de TCP que limita la cantidad de datos que un transmisor puede enviar en una conexión TCP. En un tiempo dado, un transmisor TCP no está obligado a enviar datos con un número de secuencia más alto que la suma del número de secuencia reconocido más alto y el mínimo de cwnd y rwnd (rwnd: ventana anunciada por el receptor) [5]. Siendo "cwnd" de tamaño "N", lo mencionado líneas arriba se explica con el siguiente ejemplo:

- Asumimos: rwnd >> cwnd y cwnd está expresado en bytes. Si N = 15000 bytes, y el número de secuencia más alto que ha sido reconocido hasta el momento es 12000, significa que el siguiente segmento a enviar debe tener un número de secuencia menor a 27000.

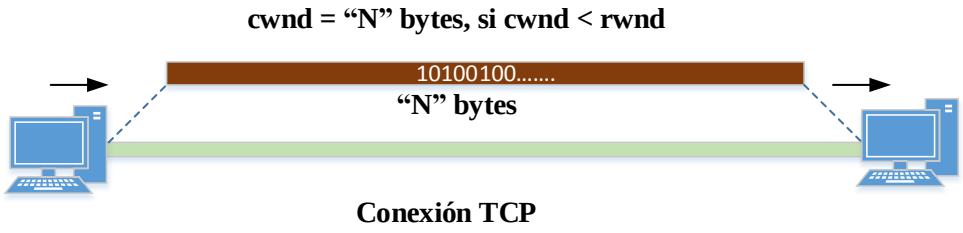


Figura 3. Ventana de congestión.

Es decir la ventana de congestión, la cual puede ser expresada en bytes o paquetes (o segmentos), representa la cantidad de datos que pueden ser enviados con el objetivo que no se produzca una congestión en la red. El valor de "cwnd" varía a lo largo de la transmisión, dependiendo de las pérdidas de datos que se produzcan y del algoritmo de control de congestión empleado [4] [1]. Se asume que los datos dentro de la ventana de congestión son transmitidos en un intervalo de tiempo equivalente a un RTT (Round-Trip-Time: Tiempo de ida y retorno) [4].

2.2. Ventana inicial

La ventana inicial es el tamaño de la ventana de congestión del transmisor después que el "handshake" de tres vías es completado [5], y será denotada como IW (Initial window), a partir de este momento. El tamaño de IW en TCP tradicional es de 1 segmento [4], mientras que en [14], menciona que la ventana inicial dependerá del valor del MSS del transmisor de la siguiente manera:

- Si (MSS ≤ 1095 bytes), entonces IW ≤ 4 ×MSS
- Si (1095 bytes < MSS < 2190 bytes), entonces IW ≤ 4380 bytes
- Si (2190 bytes ≤ MSS), entonces IW ≤ 2 ×MSS

Sin embargo las condiciones anteriores son referentes, la ventana inicial de congestión puede ser un valor distinto mayor. Según [14], se indica:

- Cuando la ventana de congestión inicial es un segmento, un receptor empleando ACK retardado, es forzado a esperar por un tiempo de expiración antes de generar un ACK (este tiempo de expiración común es de 200 a 500 ms.)
- Si la ventana de congestión inicial es mayor a las condiciones mencionadas, influye incrementando la tasa de congestión, pero este incremento no es notable.

2.3. Algoritmos de control de congestión en TCP

Según [5], existen cuatro algoritmos de control de congestión principales: Inicio lento (Slow-start), evitación de la congestión (congestion avoidance), retransmisión rápida (fast retransmit) y recuperación rápida (fast recovery)

2.3.1. Slow-start

Para implementar este algoritmo se utilizan las variables: ventana de congestión (cwnd) y el umbral del inicio lento (ssthresh: slow-start threshold), el cual se emplea para determinar si el "slow-start" o evitación de la congestión es usado para controlar la transmisión [5].

El mecanismo del "slow-start", planteado por Jacobson en [4] define lo siguiente:

- Se añade una ventana de congestión, cwnd al estado por conexión.
- Cuando ocurre una pérdida la ventana de congestión es reseteada al valor de un paquete.
- Por cada ACK de nuevos datos, se incrementa cwnd por un paquete:

$$cwnd = cwnd + 1 \quad \textbf{(1)}$$

- El transmisor envía el mínimo de la ventana de congestión "cwnd" y la ventana anunciada por el receptor denotada por "rwnd".

En las actuales implementaciones de TCP, "Slow-start" finaliza cuando la ventana de congestión "cwnd" excede el umbral "ssthresh" o cuando es presentada una congestión, asumiéndose que una congestión es producida cuando ocurre pérdida de paquetes [5]. La implementación tradicional de TCP, asume cada paquete enviado con un tamaño MSS (Máximo tamaño de segmento) [5] [4].

En la figura 4, se observa el comportamiento de la ventana de congestión durante la fase "slow-start". Asumiendo que la ventana inicial es 1 paquete de datos, cuando el primer paquete es reconocido por el receptor, este proceso tarda lo denominado 1 RTT (Tiempo de ida y retorno). Luego la ventana de congestión se incrementa a 2 paquetes, y cuando son reconocidos segundo y tercer paquetes, la ventana "cwnd" tendrá el valor de 4 paquetes, y así crecerá sucesivamente. Nótese que la numeración de la figura, no indica el número de ACK, sino el orden de cada paquete reconocido.

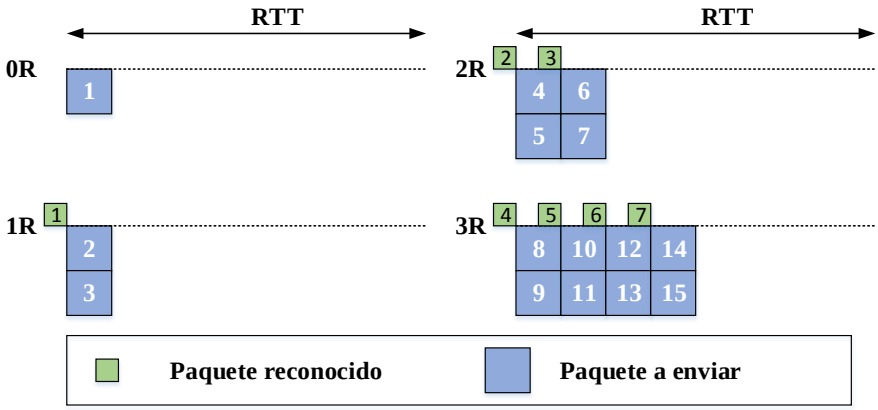


Figura 4. Cronología de "slow-start" [4]

2.3.2. Evitación de la congestión

En la implementación tradicional, definida por [4], el algoritmo de evitación de la congestión entra en operación cuando finaliza el "slow-start". A partir de este momento la ventana de congestión "cwnd", es reducida debido a una congestión producida, y dependiendo de la implementación tiene un comportamiento.

Si hablamos de la implementación tradicional de Jacobson en [4]:

- "cwnd" es disminuido a 1 paquete, cuando alcanza el valor de "ssthresh", o cuando se produce una congestión durante el "slow-start"
- Cuando se produce una congestión, "ssthresh" es reducido a la mitad del valor de la ventana de congestión.
- Después de la congestión "cwnd" crece siguiendo las reglas de "slow-start", si este no es mayor que el valor del umbral "ssthresh".
- Si "cwnd" supera el valor de "ssthresh", inicia un incremento de la siguiente manera: Por cada "cwnd" paquetes reconocidos el valor de la ventana de congestión incrementa en 1 paquete. Esto es denominado incremento aditivo.

Se asume que por cada ventana reconocida: "cwnd" aumenta en 1 paquete. Esto es:

$$cwnd = cwnd + 1/cwnd \quad \textbf{(2)}$$

2.3.3. Retransmisión rápida y recuperación rápida

Un receptor TCP debe enviar un ACK duplicado, cuando un segmento llega fuera de orden [5].

Analizando un ACK duplicado ocurre si se ha recibido el segmento con número de secuencia "Y", pero no se ha recibido el segmento de número de secuencia "X", siendo Y > X, el receptor debe enviar un ACK nuevamente, a fin de recibir el segmento que no ha llegado.

Según [5], desde el punto de vista del transmisor, un ACK duplicado, se puede producir por tres motivos:

a) ***Existen segmentos perdidos.*** Si sucede este caso, todos los segmentos enviados después del segmento perdido activaran ACKs duplicados hasta que la pérdida sea reparada.
b) Reordenamiento de segmentos.
c) Replicación de ACKs o segmentos de datos por la red.

El algoritmo de ***retransmisión rápida***, consiste en detectar y reparar basado en ACK's duplicados, de tal forma que cuando ocurren tres ACKs duplicados, será la indicación de que el segmento ha sido perdido, y el segmento será retransmitido sin esperar a que se cumpla el tiempo de expiración que ha sido fijado ("timeout") [5].

El algoritmo de ***recuperación rápida***, permite la transmisión de nueva información (nuevos segmentos de datos), hasta que un ACK no llegue duplicado, sin realizar el algoritmo de "slow-start", cuando se produce la recuperación. Cuando se produce la pérdida, el transmisor enviará los nuevos datos usando una ventana de congestión "cwnd" reducida en comparación a la ventana empleada cuando se ocasionó la pérdida [5].

A la vez cuando ocurren la retransmisión rápida y recuperación rápida, en TCP tradicional el umbral "ssthresh" es disminuido a la mitad de la cantidad de datos que han sido enviados y aún no han sido reconocidos, siempre que no sea menor de 2 segmentos [5], expresado en la ecuación:

$$ssthresh = min\left(\frac{Paquetes\ en\ vuelo}{2}, 2\right) \quad \textbf{(3)}$$

CAPITULO II: ESTUDIO DE PROTOCOLOS DE TRANSPORTE.

1. TCP TRADICIONAL.

1.1. Variantes de TCP tradicional.

1.1.1. TCP Tahoe.

Es el primer algoritmo de control de congestión aplicado a TCP, el cual fue introducido como la versión 4.2 de BSD UNIX [1], el cual se comportaba de la siguiente forma originalmente

- La conexión se inicia con un tamaño de ventana de congestión cwnd= IW ("ventana inicial"), y se emplea el algoritmo de "slow-start". El valor común de IW es 1SMSS (MSS de envío).
- Las pérdidas de paquetes son determinadas por "tiempo de expiración" (timeout) agotado.
- Cuando la pérdida es detectada, la ventana de congestión cwnd es reducida a su valor de ventana de reinicio RW (Reset window), y se realiza nuevamente el algoritmo de slow-start hasta que cwnd crezca por encima de **ssthresh**. El valor común de RW es 1 SMSS.
- El valor de ssthresh inicial es considerado muy alto (tiende al infinito), con el objetivo que el "slow-start" de inicio de conexión se lleve a cabo hasta que se produzca la primera pérdida de paquetes. El valor de ssthresh es reducido cuando ocurre una pérdida, empleando un decremento multiplicativo, como se indica en la ecuación 3.

Este algoritmo no emplea eficientemente el ancho de banda, debido a que reinicia cwnd a su mínimo valor cuando ocurre una pérdida de datos. Esto sería corregido en versiones posteriores como la versión 4.3 de BSD UNIX y sumando otros algoritmos, conocido como TCP Reno [1].

1.1.2. TCP Reno

Está basado en el algoritmo de control de congestión original de TCP (TCP Tahoe), sumado al uso de los algoritmos de retransmisión rápida (fast retransmit) y recuperación rápida (fast recovery) [1]. El comportamiento de este algoritmo es el siguiente:

- La conexión se inicia con cwnd = IW, y se emplea el algoritmo de "slow-start". El valor común de IW es 1SMSS.
- Las pérdidas de paquetes son determinadas por cada "tiempo de espera expirado" (timeout) ocurrido o en el caso en que han sido recibidos tres ACK's duplicados en el receptor, será la indicación que el segmento ha sido perdido.
- Cuando la pérdida es detectada el valor de ssthresh es reducido aplicando la ecuación 3.

- Cuando la pérdida es detectada por "tiempo de espera expirado", la ventana de congestión cwnd es reducida a su valor de ventana de reinicio RW (Reset window), y se realiza nuevamente el algoritmo de slow-start hasta que cwnd crezca por encima de ssthresh. El valor común de RW es 1 SMSS.
- Cuando la pérdida es detectada por triple ACK duplicado, cwnd será reducido al valor de ssthresh (el cuál fue obtenido con la ecuación 3). Al ocurrir esto, se puede emplear el algoritmo de recuperación rápida (fast recovery), el cual permite que cwnd se infle, creciendo temporalmente por 1 SMSS por cada ACK recibido (correspondiente a segmentos perdidos), hasta que un ACK "no duplicado" sea recibido en el transmisor, momento en el cual cwnd retorna a su valor pre-inflado, y sale de la fase de recuperación rápida, retornando a un incremento aditivo.

TCP Reno es base para el algoritmo de control de congestión conocido como "TCP Estándar" [1].

1.2. Cálculo del rendimiento

El rendimiento o ancho de banda de una conexión se mide en bits/segundos. A continuación se muestran algunas formas de calcular esta métrica.

1.2.1. Fórmula general

En [15] se considera que el ancho de banda (rendimiento) es la cantidad de paquetes enviados en un intervalo de tiempo [0, t], esto es: $B_t = N_t/t$, donde N_t es la cantidad de paquetes transmitidos. Si se considera que todos los paquetes tienen el mismo tamaño MSS en la transmisión, MSS es expresado en bits, y t en segundos, el ancho de banda se obtiene por la ecuación 4. Esta fórmula es muy simple y no considera las pérdidas producidas en una transmisión.

$$B_t[\frac{bits}{segundo}] = \frac{N_t}{t[segundos]} \times MSS[bits] \quad \textbf{(4)}$$

En [16], se considera al rendimiento como el tamaño de una ventana de congestión enviado en un tiempo dado, el cual es aproximadamente igual a un RTT [4]. En base a esto se expone la ecuación:

$$B \approx \frac{cwnd}{RTT[segundos]} \times MSS[bits] \quad \textbf{(5)}$$

En base a la ecuación 5, se afirma que normalmente el tráfico (rendimiento: "throughput") es inversamente proporcional al RTT, por ello cuando el RTT es más alto, menor es su valor. Obsérvese la figura 5 sobre la relación entre el rendimiento y RTT.

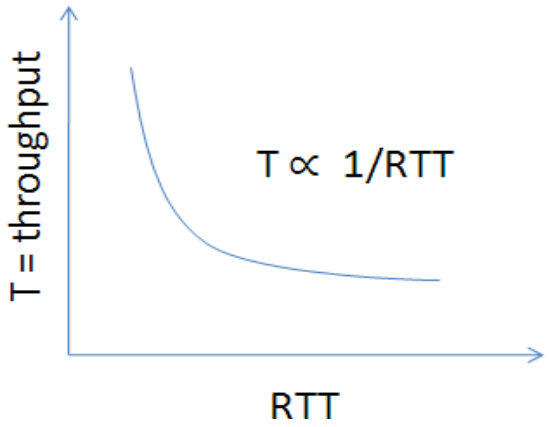


Figura 5. Rendimiento ("Throughput") vs. RTT

1.2.2. Modelo de Padhye

Siendo conocidos el valor promedio de RTT y la tasa de pérdidas de paquetes, para una transmisión que emplea TCP estándar (TCP Reno), el ancho de banda es determinado por la ecuación 6 [15].

$$B \approx min\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{\frac{2bp}{3}}+T_0 \min\left(1{,}3\sqrt{\frac{3bp}{8}}\right)p(1+32p^2)}\right) \text{[paquetes/segundo]} \quad \textbf{(6)}$$

Donde:

B	:	Máximo ancho de banda teórico de una conexión expresado en paquetes/segundo.
W_{max}	:	Tamaño de ventana máximo aceptable a ser recibida, anunciada por el receptor.
RTT	:	Tiempo de ida y vuelta (Promedio) expresado en segundos
b	:	Cantidad de paquetes enviados antes de recibir un ACK del receptor.
p	:	Proporción de pérdidas de paquetes.
T_0	:	Tiempo de espera para la expiración de una conexión (Timeout).

Si se asume que la ventana de datos permisible por el receptor es muy grande (inalcanzable por el transmisor), entonces la fórmula estaría dada por:

$$B \approx \frac{1}{RTT\sqrt{\frac{2bp}{3}}+T_0 \min\left(1{,}3\sqrt{\frac{3bp}{8}}\right)p(1+32p^2)} \text{[paquetes/segundo]} \quad \textbf{(7)}$$

En el denominador de la ecuación 7, se hallan dos sumandos: el primero depende de las pérdidas debido a triple duplicado ACK y el segundo corresponde a las pérdidas por "timeout". Al considerar que la mayor parte de las pérdidas son debido a triple duplicado ACK, la expresión es reducida a la ecuación 8, que representa el ancho de banda máximo alcanzable en una conexión.

$$B \approx \frac{1}{RTT\sqrt{\frac{2bp}{3}}} \text{[paquetes/segundo]} \quad \textbf{(8)}$$

1.2.3. Modelo de Mathis.

En [17], el cálculo del rendimiento teórico para una variante TCP estándar como lo es TCP Reno se realiza empleando la siguiente ecuación:

$$T = \frac{MSS}{RTT} \times \frac{C}{\sqrt{p}} \text{ [bits/segundo]} \quad \textbf{(9)}$$

Donde:

- *T* es el rendimiento expresado en cantidad de bits por segundo.
- *MSS* es el tamaño máximo del segmento (expresado en bits).
- *RTT* es el tiempo de ida y vuelta para que una ventana sea reconocida (expresado en segundos).
- *"p"* es el porcentaje de pérdidas de paquetes en una conexión TCP.
- *"C"* es denominada constante de proporcionalidad.

Si se desarrolla la ecuación 8, para que el ancho de banda sea expresado en bits/s, asumiendo que es recibido un ACK por cada paquete enviado (b=1), se obtiene la ecuación 10. El concepto de paquete hace referencia a un segmento con los encabezados añadidos para ser reconocido como una unidad de capa 3.

$$B \approx \frac{\sqrt{3/2}}{RTT\sqrt{p}} \times MSS \quad \text{[bits/segundo]} \quad \textbf{(10)}$$

Si se igualan las ecuaciones 8 y 10, *C* es igual a $\sqrt{3/2}$. Sin embargo en [17] se considera que el valor de *C* depende de las condiciones de pérdida y reconocimiento de segmentos, lo cual es mostrado en la tabla 3, siendo $C = \sqrt{3/2}$ cuando las pérdidas son periódicas y es recibido un ACK por paquete reconocido. El concepto de ACK retardado es explicado en el Anexo A.

Condición	Estrategia de reconocimiento	Valor de "C"
Pérdidas periódicas	ACK por cada paquete	$1.22 = \sqrt{3/2}$
Pérdidas periódicas	ACK retardado	$0.87 = \sqrt{3/4}$
Pérdidas aleatorias	ACK por cada paquete	1.31
Pérdidas aleatorias	ACK retardado	0.93

Tabla 3. Valores de "C" bajo diferentes condiciones [17].

Las ventajas de este modelo son el uso de métricas disponibles en una conexión y la facilidad de cálculo, pero considera una tasa de pérdidas mayor a cero en la ecuación 10, caso contrario el ancho de banda sería infinito, además no se toma en cuenta la ocurrencia de "timeouts" [16].

En ambos modelos: de Mathis y de Padhye, se observa que el rendimiento es inversamente proporcional al porcentaje de pérdidas en una transmisión: Si más paquetes son perdidos en una transmisión, menor será el rendimiento. Gráficamente se observa en la figura 6.

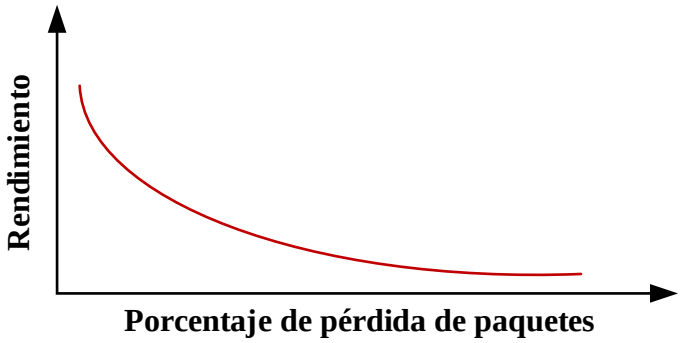


Figura 6. Rendimiento ("Throughput") vs. Porcentaje de pérdida de paquetes.

1.3. Additive Increase Multiplicative Decrease (AIMD).

AIMD es un algoritmo empleado por TCP tradicional durante la fase de evitación de la congestión, el cual comprende:

a) Un incremento aditivo, cuando no se produce congestión y la ventana de congestión se encuentra por encima del umbral de inicio (slow-start threshold).

b) Un decremento multiplicativo, cuando se produce una congestión.

El incremento aditivo es representado por "α" e indica que por cada paquete enviado reconocido.

$$cwnd_{n+1} = cwnd_n + {}^{\alpha}/_{cwnd_n} \quad \textbf{(11)}$$

Donde:

α	Factor de incremento aditivo. ($\alpha>0$)
$cwnd_n$	Ventana de congestión en un instante cuando se envía un paquete
$cwnd_{n+1}$	Ventana de congestión cuando el paquete ha sido reconocido

El decremento multiplicativo se emplea cuando existe una congestión, siendo:

$$cwnd_{n+1} = \beta . cwnd_n \quad \textbf{(12)}$$

Donde:

β	Factor de decremento multiplicativo. ($0<\beta<1$)
$cwnd_n$	Ventana de congestión en el instante que se produce la congestión
$cwnd_{n+1}$	Ventana de congestión después que se produce la congestión

TCP tradicional emplea las ecuaciones 11 y 12, considerando α=1 y β=1/2. La tasa de envío TCP es controlada por la ventana de congestión, cuyo valor es reducido a la mitad cuando ocurre una pérdida de paquetes, y es incrementado por 1 paquete por ventana de datos reconocida [6].

2. General Additive Increase – Multiplicative Decrease (GAIMD)

2.1. Comportamiento de la ventana de congestión

Una sesión que emplea GAIMD inicia en el estado "slow-start", en el cual la ventana de congestión se duplica por cada ventana de paquetes reconocida. Cuando ocurre la primera congestión, el valor de la ventana de congestión es reducido a la mitad y se ingresa en la fase de evitación de congestión. Según [6], durante la evitación de congestión, GAIMD emplea el siguiente algoritmo:

Si un paquete es reconocido:

$$cwnd_{n+1} = cwnd_n + {}^{\alpha}/_{cwnd_n} \quad \textbf{(11)}$$

Si ocurre una congestión:

Si la congestión es por triple-duplicado ACK:

$$cwnd_{n+1} = \beta . cwnd_n \quad \textbf{(12)}$$

Si la congestión es por una espera de tiempo prolongada ("timeout")

$$cwnd = 1 \quad \textbf{(13)}$$

GAIMD establece una relación entre los parámetros de incremento aditivo y decremento multiplicativo, expresado en la ecuación 14, válida para una tasa de pérdidas de hasta el 20% [6], debiéndose cumplir la condición de AIMD para ($\alpha>0$) y ($0<\beta<1$).

$$\alpha = 4.(1 - \beta^2)/3 \quad \textbf{(14)}$$

TCP es amistoso cuando: α=0.31 y β =7/8. Estos valores satisfacen la ecuación anterior.

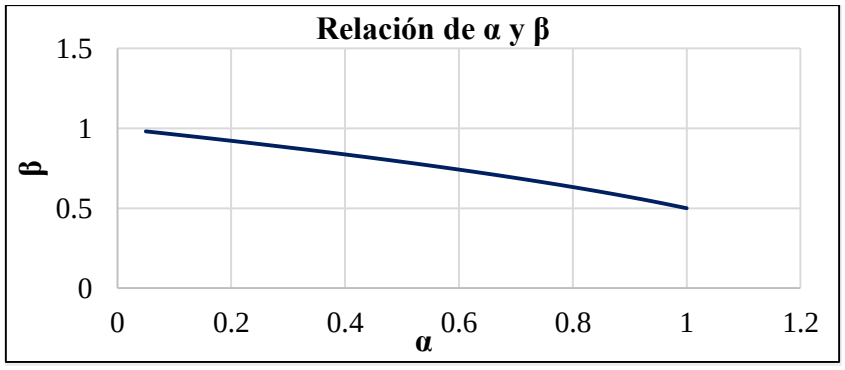


Figura 7. Relación de α y β en GAIMD

Muchas implementaciones TCP envían un ACK acumulativo por cada 2 paquetes consecutivos recibidos. En este caso, el tamaño de la ventana es incrementado α/2 por RTT [6].

2.2. Consideraciones del modelo.

En [6], GAIMD es propuesto asumiendo las siguientes consideraciones:

a) El transmisor siempre envía datos. El receptor anuncia una ventana de recepción muy grande, de modo que la ventana de envío es determinado por el valor de la ventana de congestión.

b) La tasa de envío es un proceso aleatorio.

c) El impacto del "slow-start" es ignorado

d) El modelamiento es por rondas: Una ronda consiste en la transmisión de "n" paquetes, siendo "n" el tamaño de la ventana de congestión. Se asume que la duración de una ronda es igual a un RTT, independiente del tamaño de la ventana.

e) Las pérdidas en diferentes rondas son independientes.

f) **"p"** es la probabilidad de que un paquete sea perdido, dado que este es cualquiera de ambos: el primer paquete en su ronda o el paquete precedente en su ronda que no es perdido.

2.3. Tasa de envío (Rendimiento)

La tasa de envío promedio, en [6], es obtenida mediante la ecuación 15.

$$T = \frac{1}{RTT.\sqrt{\frac{2b(1-\beta)}{\alpha(1+\beta)}p} + To.min\left(1{,}3.\sqrt{\frac{(1-\beta^2)b}{2\alpha}.p}\right).p.(1+32.p^2)} \quad \textbf{(15)}$$

Donde:

p: Probabilidad de pérdidas.

RTT: Round trip time (Tiempo ida y vuelta)

To: Timeout (Tiempo de espera durante inactividad)

b: Número de paquetes reconocidos por ACK

T: Tasa de envío promedio.

En la expresión anterior, el denominador puede ser descompuesto en dos términos $TD_{\alpha\beta}$ y $TO_{\alpha\beta}$, determinados por las ecuaciones 16 y 17. El primer término indica que la congestión se debe a pérdidas debido a triple ACKs duplicados, y el segundo término indica congestión por "timeout".

$$TD_{\alpha,\beta} = RTT.\sqrt{\frac{2b(1-\beta)}{\alpha(1+\beta)}p} \quad \textbf{(16)}$$

$$TO_{\alpha,\beta} = To.min\left(1{,}3.\sqrt{\frac{(1-\beta^2)b}{2\alpha}}.p\right).p.(1+32.p^2) \quad \textbf{(17)}$$

Cuando la probabilidad de pérdidas es pequeña, $TD_{\alpha\beta}$ domina a $TO_{\alpha\beta}$. GAIMD, es eficiente cuando las pérdidas no superan el 20% de la transmisión, con lo cual se concluye que $TD_{\alpha\beta}$ predomina. Por tanto la ecuación 15 se reduce a:

$$T \approx \frac{1}{RTT.\sqrt{\frac{2b(1-\beta)}{\alpha(1+\beta)}p}} \quad \textbf{(18)}$$

2.3.1. Tasa de envío promedio para TCP tradicional.

TCP tradicional es un protocolo AIMD, que satisface la ecuación 14, tal que $\alpha = 1$ y $\beta = 1/2$. Al remplazar estos parámetros en la ecuación 18, y considerando que es recibido un ACK por cada paquete reconocido (b=1) se obtiene la siguiente expresión:

$$T_{TCP} \approx (1/RTT).\sqrt{\frac{3}{2p}}, tal\ que\ \alpha = 1\ y\ \beta = 1/2 \quad \textbf{(19)}$$

2.3.2. Tasa de envío promedio para GAIMD amistoso

Asumiendo b=1, al emplear la ecuación 16, $TD_{\alpha,\beta}$ para GAIMD amistoso (α =0.31 y β=7/8) es: $TD_{0.31,7/8} = RTT.\sqrt{0.43p}$. Por tanto la tasa de envío promedio para GAIMD amistoso es dada por:

$$T_{GAIMD} \approx (1/RTT).\sqrt{\frac{2.325}{p}} \quad \textbf{(20)}$$

2.3.3. Eficiencia teórica de GAIMD sobre TCP tradicional.

Si se observa las ecuaciones 19 y 20, se aprecia que la tasa de envío promedio del protocolo GAIMD amistoso es mayor que la del protocolo TCP tradicional. La relación teórica entre la tasa de envío promedio de los protocolos GAIMD amistoso y TCP tradicional sería:

$$\frac{T_{GAIMD}}{T_{TCP}} = 1.24 \quad \textbf{(21)}$$

De lo expuesto líneas arriba, para una tasa de pérdidas menor al 20%:

a) La tasa de envío promedio máxima teórica de TCP tradicional, es: $T_{TCP} \approx (1/RTT).\sqrt{\frac{3}{2p}}$

b) La tasa de envío promedio máxima teórica de GAIMD amistoso es: $T_{GAIMD} \approx \frac{1}{RTT} \cdot \sqrt{\frac{2.325}{p}}$

c) Bajo las mismas condiciones de RTT y tasa de pérdidas de paquetes, teóricamente GAIMD amistoso tiene un rendimiento aproximadamente 24% más eficiente que en TCP tradicional.

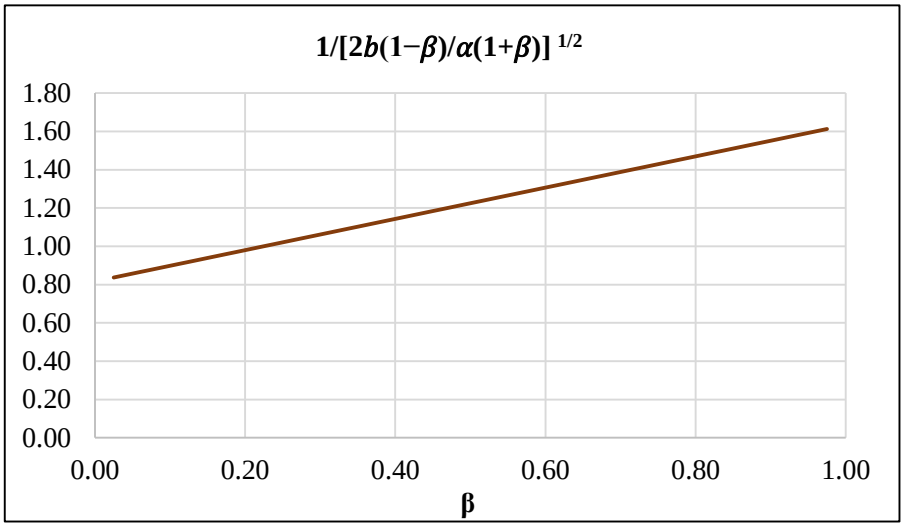


Figura 8. $1/[2b(1-\beta)/\alpha(1+\beta)]^{1/2}$ VS. β

T es proporcional al factor $\frac{1}{\sqrt{\frac{2b(1-\beta)}{\alpha(1+\beta)}}}$ y varía respecto a α y a β. Este factor es representado en relación a los valores de α y β en las figuras 8 y 9.

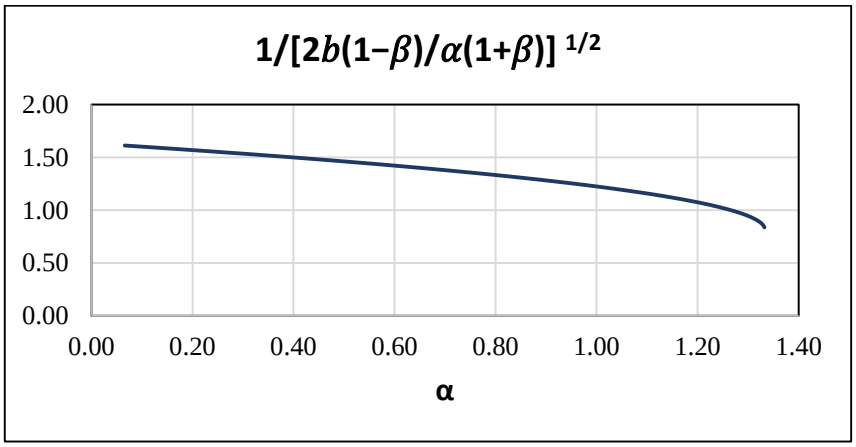


Figura 9. $1/[2b(1-\beta)/\alpha(1+\beta)]^{1/2}$ VS. α

3. PROTOCOLO DE TRANSPORTE DE SERVICIOS ETHERNET (ESTP)

3.1. CIR y EIR.

La tasa de información comprometida (CIR: Committed Information Rate) es la tasa promedio (en bytes por unidad de tiempo), a la cual la red está comprometida a transmitir tramas y encontrar los objetivos de desempeño [18]. Otra definición es: CIR es la tasa de información sostenida que la red está comprometida a transmitir mientras encuentra el nivel de desempeño garantizado en un acuerdo de nivel de servicio (SLA: Service Level Agreement) [19].

La tasa de información excedida (EIR: Excess Information Rate) es definida como la tasa promedio (en bytes por unidad de tiempo), en exceso de la CIR, a la cual la red quizá transfiera tramas sin una meta de desempeño [18]. La expectativa es que la red transporte este tráfico [19].

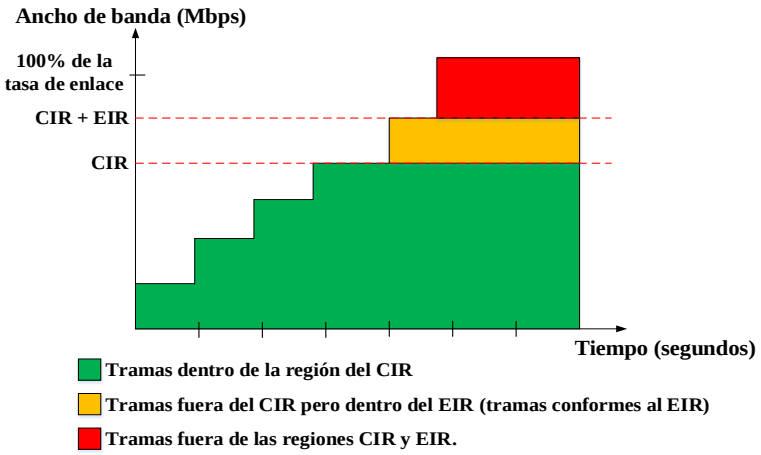


Figura 10. Clasificación de tramas de acuerdo a CIR y EIR. [2]

Los paquetes transmitidos son procesados de acuerdo a CIR y EIR, y se clasifican en 3 grupos:

a) Paquetes dentro de la región CIR: Serán marcados con alta prioridad y entregados con calidad de servicio garantizada.

b) Paquetes que ingresan fuera de la región CIR, pero aún están dentro de EIR: Serán marcados con baja prioridad y entregados en modo mejor esfuerzo.

c) Paquetes fuera de la región de EIR: Serán descartados.

3.2. Capacidad de canal y probabilidad de pérdidas

Para evaluar la capacidad de canal de una línea de una red CEN (Carrier Ethernet Network: Red Ethernet portadora), se emplean los parámetros EIR y CIR, y dos tipos de probabilidades de pérdidas: La probabilidad de pérdidas de paquetes de alta prioridad (dentro de la región CIR),

denotada como *p1* y la probabilidad de pérdidas de paquetes de mejor esfuerzo (Dentro de la región de EIR y fuera de la región CIR), denotada como *p2*. Los paquetes de alta prioridad presentan una probabilidad de pérdidas menor a los paquetes de mejor esfuerzo: $p1 <= p2 <= 1$ [7].

La capacidad del canal es expresada como "C" dada por la ecuación:

$$C = CIR \times (1 - p1) + EIR \times (1 - p2) \quad \textbf{(22)}$$

Donde: $(1 - p1)$ representa la proporción de paquetes no perdidos de alta prioridad y $(1 - p2)$ representa la proporción de paquetes no perdidos de mejor esfuerzo.

Ejemplo: En una línea CEN, durante la transmisión: Se envió un total de 100 paquetes: 40 paquetes de alta prioridad y 60 paquetes de mejor esfuerzo y se perdieron 2 paquetes de alta prioridad y 6 paquetes de mejor esfuerzo. La CIR es de 7.5×10^6 Bytes/s y la EIR es de 2.5×10^6 Bytes/s.

Para calcular *C*, primero se obtiene los valores de *p1* y *p2*: $p1 = 2/40 = 0.05$ y $p2 = 6/60 = 0.10$. Luego se expresa CIR y EIR en términos de Mb/s, los cuales son respectivamente: 60 y 20 Mb/s. Por tanto: $C = 60. (1-0.05) + 20. (1-0.10) = 57 + 18 = 75$ Mb/s.

Analizando si *p1* y *p2* fuesen 0 cada uno, es decir hablamos de un canal ideal, la capacidad máxima del canal sería: $C_{máx.} = CIR + EIR$. Por tanto se puede establecer que:

$$C \leq CIR + EIR \quad \textbf{(23)}$$

3.3. Funcionamiento de ESTP

ESTP es un protocolo que busca incorporar la información de calidad de servicio en el protocolo de transporte para controlar mejor el flujo de datos sobre la red que emplea servicios Ethernet. ESTP considera 3 aspectos: Nivel de congestión, Función de mapeo, y Algoritmo de control de evasión de la congestión [7].

a) El nivel de congestión es la cantidad de paquetes entregados satisfactoriamente entre 2 pérdidas más uno, renombrado aquí como el parámetro **δ**, siendo **δ ≥1** (δ = 1 cuando ocurren dos pérdidas consecutivas). El valor de **δ** es más pequeño a medida que la congestión sea más alta, y por consiguiente su valor es más grande cuando la congestión sea menor.

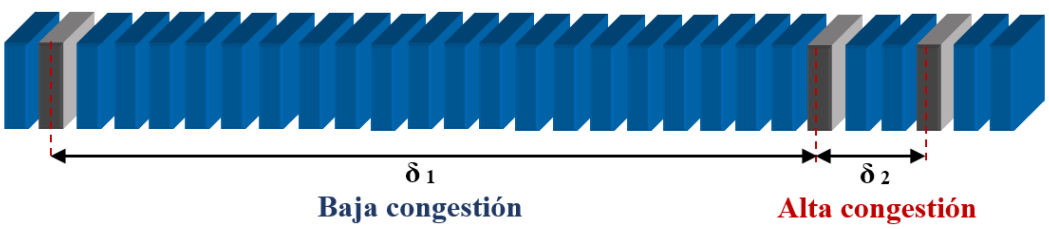


Figura 11. Relación de nivel de congestión con δ (Basado en [7]) .

b) La función de mapeo es influenciada por el nivel de cogestión en la transmisión, y es inversamente proporcional al factor de decremento multiplicativo de TCP [7], y ésta es denotada como *map(δ).*

$$map\ (\delta) = e^{-\frac{\boldsymbol{\delta}-1}{\tau}} + 1 \quad \textbf{(24)}$$

Donde **δ** es el nivel de congestión y τ es el promedio de paquetes transmitidos entre dos pérdidas, es decir es el promedio de los valores de **δ** durante una transmisión [7].

El factor de decremento multiplicativo en el protocolo ESTP, se expresa como:

$$\beta = \frac{1}{map(\delta)} \quad \textbf{(25)}$$

Ante una congestión producida la ventana de congestión ***cwnd***, se comporta:

$$cwnd_{n+1} = cwnd.\left(\frac{1}{e^{-\frac{\delta-1}{\tau}} + 1}\right) \quad \textbf{(26)}$$

Si δ = 1 (dos pérdidas consecutivas), entonces *map* (δ) = 2, por lo tanto $\beta = 1/2$, lo cual indicaría un comportamiento similar a TCP tradicional en ese momento.

El valor de τ depende de los valores de δ anteriormente presentados en una transmisión, los cuales pueden ser altos o pequeños, siendo esto más notable cuando las pérdidas son aleatorias. Por tanto, dado δ_n al momento de una congestión y siendo $\tau = \sum_{i=1}^{n} \delta_i / n$:

- Si $\delta_n \ll \tau$, la función de mapeo crece con tendencia a 2, y el comportamiento del decremento multiplicativo es similar a TCP tradicional.
- Si $\delta_n \gg \tau$, la función de mapeo se acerca a 1, y el decremento multiplicativo tiende a ser 1.

La influencia del nivel de congestión promedio presentado antes del último valor de α obtenido, es determinante en el valor de decremento multiplicativo.

c) En [7], se indica que al combinar mecanismos de control de congestión con mecanismos conscientes de SLA, la siguiente expresión de decremento multiplicativo es derivada:

$$cwnd_{n+1} = \frac{cwnd_n - cwnd_{MIN}}{e^{-\frac{\delta-1}{\tau}} + 1} + cwnd_{MIN} \quad \textbf{(27)}$$

En la ecuación 27: $cwnd_{MIN} = RTT.CIR$. Análogamente el tamaño máximo que podría alcanzar la ventana de congestión sería: $cwnd_{MAX} = RTT.EIR$. Si ocurre que $cwnd_{MIN} = 0$, entonces $cwnd_{n+1} = cwnd.\left(\frac{1}{e^{-\frac{\delta-1}{\tau}}+1}\right)$, esto significaría que todo el tráfico sería tratado sin

prioridad. Aun así en la práctica $cwnd_{MIN}$ no puede adquirir un valor de 0, ya que una ventana debe contener a lo menos un paquete.

Cuando no ocurre congestión el incremento aditivo en ESTP es α =1, esto es expresado:

$$cwnd_{n+1} = \min(cwnd_{MAX,} cwnd + \frac{1}{cwnd}) \quad \textbf{(28)}$$

3.4. Consideraciones en ESTP

3.4.1. Porcentaje de pérdidas constante

Si se asume que la proporción de pérdidas es constante, es decir cada N paquetes transmitidos, uno es perdido, entonces, ocurre una congestión: $\delta_n = \tau$, donde n es el número de congestiones producidas. Esto sucede porque $\tau = \sum_{i=1}^{n} \delta_i / n$ y todos los δ_i tienen el mismo valor.

Si se haya el valor de la función de mapeo y el decremento multiplicativo se observa que el valor de β es casi el mismo para diferentes valores de δ, indicándose que para este caso τ tiene el mismo valor en toda la transmisión, y es inversamente proporcional a la proporción de pérdidas.

Pérdidas	δ	τ	map(δ)	β
1.00%	100	100	1.372	**0.73**
1.05%	95	95	1.372	**0.73**
1.11%	90	90	1.372	**0.73**
1.18%	85	85	1.372	**0.73**
1.25%	80	80	1.373	**0.73**
1.33%	75	75	1.373	**0.73**
1.43%	70	70	1.373	**0.73**
1.54%	65	65	1.374	**0.73**
1.67%	60	60	1.374	**0.73**
1.82%	55	55	1.375	**0.73**

Pérdidas	δ	τ	map(δ)	β
2.00%	50	50	1.375	**0.73**
2.22%	45	45	1.376	**0.73**
2.50%	40	40	1.377	**0.73**
2.86%	35	35	1.379	**0.73**
3.33%	30	30	1.380	**0.72**
4.00%	25	25	1.383	**0.72**
5.00%	20	20	1.387	**0.72**
6.67%	15	15	1.393	**0.72**
10.00%	10	10	1.407	**0.71**
20.00%	5	5	1.449	**0.69**

Tabla 4. Función de mapeo y β cuando la proporción de pérdida es constante.

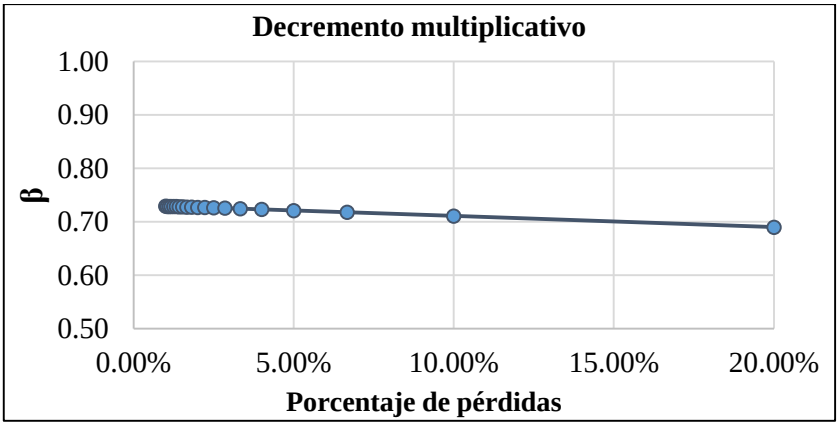


Figura 12. Decremento multiplicativo cuando el porcentaje de pérdida es constante.

En la tabla 4 se observa el valor de β para diferentes proporciones de pérdidas en un entorno ideal donde las pérdidas son periódicas.

3.4.2. Porcentaje de pérdidas aleatorio.

Las pérdidas son aleatorias, cuando el valor de la proporción de pérdida en cada pérdida es distinta de cuando se produjo la anterior pérdida.

En un tiempo igual a t_n se han producido "n" pérdidas, se obtiene el nuevo valor de β, éste puede variar dependiendo del valor de δ_n en el momento en que ocurre la congestión.

δ	τ	map (δ)	β	δ	τ	map (δ)	β
5	50	1.923	0.52	65	50	1.278	0.78
10	50	1.835	0.54	70	50	1.252	0.80
15	50	1.756	0.57	75	50	1.228	0.81
20	50	1.684	0.59	80	50	1.206	0.83
25	50	1.619	0.62	85	50	1.186	0.84
30	50	1.560	0.64	90	50	1.169	0.86
35	50	1.507	0.66	95	50	1.153	0.87
40	50	1.458	0.69	100	50	1.138	0.88
45	50	1.415	0.71	105	50	1.125	0.89
50	**50**	**1.375**	**0.73**	110	50	1.113	0.90
55	50	1.340	0.75	115	50	1.102	0.91
60	50	1.307	0.76	120	50	1.093	0.92

Tabla 5. Función de mapeo y β para diferentes valores de δ y τ

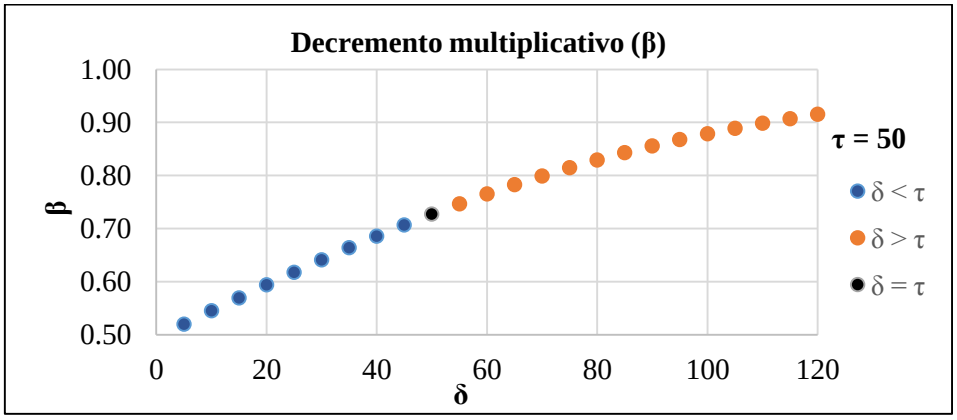


Figura 13. β para diferentes comportamientos de δ para un valor de τ.

Cada una de las filas en la tabla 5 es un caso para un valor diferente de δ, y cada combinación de valores de δ y τ se representa con un "punto" en el diagrama de dispersión (Figura 13).

La figura 13 se ha construido a partir de la tabla 5, asumiendo que se tiene un valor τ=50 en el momento que ocurre la "n-ésima" pérdida. El valor de β varía dependiendo del δ obtenido.

CAPITULO III: DISEÑO DE UN BANCO DE PRUEBAS PARA LA EVALUACIÓN DE PROTOCOLOS DE TRANSPORTE.

1. Herramientas para el diseño del banco de pruebas.

1.1. Herramientas de emulación.

1.1.1. Netem

Netem provee la funcionalidad de emulación de red para testear protocolos: emula retardo, pérdidas, duplicación y re-ordenamiento de paquetes. Netem está controlado por la utilidad de línea de comando "tc", que usa las librerías compartidas y archivos de datos en el directorio: /usr/lib/tc [20].

1.1.1.1. Emulación de RTT.

Se puede cambiar el RTT mediante la sintaxis de línea de comando:

```
tc qdisc replace dev [interfaz de red] root netem delay [RTT]
```

Ejemplo: Aplicar un RTT de 100ms a las transmisiones de paquetes sobre la interfaz Ethernet 0.

```
tc qdisc replace dev eth0 root netem delay 100ms
```

1.1.1.2. Emulación de pérdidas aleatorias.

Se puede añadir pérdidas mediante la sintaxis de línea de comando:

```
tc qdisc add dev [interfaz de red] root netem loss [porcentaje de pérdidas]
```

Cuando existen parámetros de RTT y pérdidas configurados en el terminal, es preferible primero remover estos mediante la sintaxis de línea de comando:

```
tc qdisc delete dev [interfaz de red] root netem.
```

Ejemplo: Fijar un RTT de 100ms a las transmisiones sobre Ethernet 0 y añadir pérdidas del 1%.

```
tc qdisc delete dev eth0 root netem

tc qdisc add dev eth0 root netem delay 100ms loss 1%
```

O de otra forma, se aplica la línea de comando:

```
tc qdisc replace dev eth0 root netem delay 100ms loss 1%
```

1.1.2. IPTABLES

Iptables es una utilidad usada para configurar, mantener e inspeccionar las tablas de reglas de paquetes Ipv4 en un kernel Linux. Es posible definir varias tablas, y cada tabla contiene un número de reglas y también pueden contener reglas definidas por usuarios [10]. Cada regla indica qué hacer con un paquete o un conjunto de paquetes, el cual es llamado objetivo.

1.1.2.1. Uso de iptables para configuración de pérdidas constantes.

Iptables permite definir reglas para descartar uno o más paquetes TCP en el receptor cada cierta cantidad de paquetes recibidos, de tal modo que el receptor no envíe los segmentos de reconocimiento (ACK) al transmisor, y éste lo considere una pérdida constante. La sintaxis de la línea de comando para definir lo expuesto en el host receptor es:

```
sudo iptables –A INPUT –i [interfaz de red] –p tcp –s [dirección IP origen] –destination-port [puerto de escucha] –m statistic –mode nth --every [cantidad de paquetes] --packet [n-ésimo paquete] –j DROP
```

Donde:

-A : Añade una regla o un conjunto de reglas.

INPUT : La regla es definida para tráfico de datos entrante al host en el cual es configurada.

-i : Permite fijar a qué interfaz de red se aplica la regla. Ejemplo: **-i eth0**.

-p : Tipo de puerto al que aplica la regla.

-s : Permite especificar la dirección de origen para la cual se aplica la regla.

--destination-port: Permite aplicar la regla a paquetes recibidos en un puerto destino específico.

-m : Fija la operación a ejecutar. (statistic: Operación estadística)

nth : La acción es aplicada para un enésimo objeto.

--every : Indica que cada cierta cantidad de paquetes se seleccione uno o más para la regla.

--packet: Selecciona el orden del paquete a cumplir con la regla. El orden del primer paquete es 0.

-j : Especifica el objetivo de la regla: Es acompañada de ACCEPT, DROP, QUEUE o RETURN.

DROP : Elimina los datos especificados en la regla.

Por ejemplo, para configurar una pérdida aproximada del 1% de los paquetes transmitidos, se indica que en el receptor se elimina el último paquete de cada 100 recibidos. El primer paquete recibido es de orden 0 y el último recibido es de orden 99. La regla aplica a los paquetes recibidos en la interfaz Ethernet, en el puerto TCP 5001, y cuya dirección de origen es 192.168.1.2, y sería definida con la línea de comando:

```
sudo iptables –A INPUT –i eth0 –p tcp –s 192.168.1.2  --destination-port 5001 –m statistic –mode nth --every 100 --packet 99 –j DROP
```

1.2. Herramientas para extracción de datos.

1.2.1. iPerf

iPerf es una herramienta para la medición activa del ancho de banda máximo disponible en redes IP y soporta varios protocolos como TCP, UDP, SCTP con IPv4 e IPv6. Este fue desarrollado

originalmente por NLANR / DAST. Actualmente iPerf es desarrollado por ESnet / Lawrence Berkeley National Laboratory. La última versión es iPerf3 [8].

1.2.1.1. Características de iPerf

Las principales características de iPerf, mencionadas en [8] son:

- Medición de ancho de banda.
- Reporta el tamaño de MSS.
- Puede funcionar durante un tiempo especificado, en lugar de una cantidad fija de datos a transferir.
- Soporte para tamaño de la ventana TCP a través de buffers de socket.
- Recoge las mejores unidades para el tamaño de los datos que se informa.
- El servidor maneja múltiples conexiones.
- Imprime periódicamente el ancho de banda intermedio.
- El puerto TCP empleado por defecto es 5001. Ofrece la posibilidad de configurar el puerto TCP para la transmisión.

1.2.1.2. Uso de iPerf en Linux.

Actualmente iPerf está disponible en las versiones iPerf2 e iPerf3, y es descargable desde el enlace: https://iperf.fr/iperf-download.php#more-recent. iPerf se puede emplear en modo servidor y modo cliente:

a) Servidor.

En este modo de ejecución, el terminal escucha las solicitudes (recibe) datos. Para nuestro caso: una terminal que ejecuta iPerf en modo servidor es un receptor.

Se ejecuta el modo "servidor" de iPerf con la línea de comando:

```
iperf -s
```

```
root@pena-HP-nx9005-DG200A:~# iperf -s
------------------------------------------------------------
Server listening on port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.5.12 port 5001 connected with 192.168.5.2 port 55014
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.2 sec  71.1 MBytes  58.4 Mbits/sec
```

Figura 14. Ejecución de iPerf en modo servidor.

b) Cliente

En este modo de ejecución, el terminal envía paquetes. Un terminal que ejecuta iPerf en modo cliente es un transmisor.

Se ejecuta el modo “cliente” de iPerf con la línea de comando:

```
iperf -c [Dirección IP del equipo en modo 'server']
```

iPerf muestra múltiples opciones, como definir el intervalo de tiempo en que se muestra el reporte, y el tiempo en que se mantiene la conexión establecida. Por ejemplo: Si la transmisión de paquetes se realiza durante 60 segundos y el destino tiene dirección IP 192.168.5.12, la línea de comando es:

```
iperf -i 10 -t 60 -c 192.168.5.12
```

-i 10 : El reporte de ancho de banda es hecho cada 10 segundos desde el último reporte.

-t 60 : Es el tiempo en segundos a transmitir. En este caso es de 60 segundos.

```
root@alumno-HP-nx9005-DG200A:~# iperf -i 10 -t 60 -c 192.168.5.12
------------------------------------------------------------
Client connecting to 192.168.5.12, TCP port 5001
TCP window size: 1024 KByte (default)
------------------------------------------------------------
[  3] local 192.168.5.2 port 55016 connected with 192.168.5.12 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  71.5 MBytes  60.0 Mbits/sec
[  3] 10.0-20.0 sec  79.0 MBytes  66.3 Mbits/sec
[  3] 20.0-30.0 sec  86.1 MBytes  72.2 Mbits/sec
[  3] 30.0-40.0 sec  28.8 MBytes  24.1 Mbits/sec
[  3] 40.0-50.0 sec  32.0 MBytes  26.8 Mbits/sec
[  3] 50.0-60.0 sec  39.1 MBytes  32.8 Mbits/sec
[  3]  0.0-60.9 sec   337 MBytes  47.0 Mbits/sec
root@alumno-HP-nx9005-DG200A:~#
```

Figura 15. Ejecución de iPerf en modo cliente

El reporte de iPerf puede ser almacenado añadiendo: **> log.txt** a la línea de comando.

```
iperf -i 10 -t 60 -c 192.168.5.12 > log.txt
```

Donde log.txt es el archivo donde se guardará el reporte de ancho de banda de iPerf.

1.2.2. TCP Probe

TCP Probe es un módulo que registra el estado de una conexión TCP en respuesta a los paquetes entrantes. Permite que la ventana de congestión y número de secuencia puedan ser capturados [9].

1.2.2.1. Uso de TCP Probe

Se ejecuta TCP Probe en el transmisor mediante la siguiente línea de comando.

```
modprobe tcp_probe port=5001 full=1
cat /proc/net/tcpprobe >/tmp/salida.out &
CAP=$!
iperf -i 10 -t 60 -c 192.168.5.12
kill $CAP
```

En las líneas anteriores se indica:

- TCP Probe registra el estado de la conexión TCP en el puerto 5001.

- Se captura el estado de conexión cuando iPerf genera una transmisión al host con dirección IP 192.168.5.12 en un tiempo de 60 segundos (El reporte de iPerf es cada 10 segundos).
- Se copia la salida de tcpprobe hacia una archivo salida.out en el directorio /tmp.

En el receptor se ejecuta la línea de comando: **iperf –s**.

1.2.2.2. Archivo de captura generado por TCP Probe

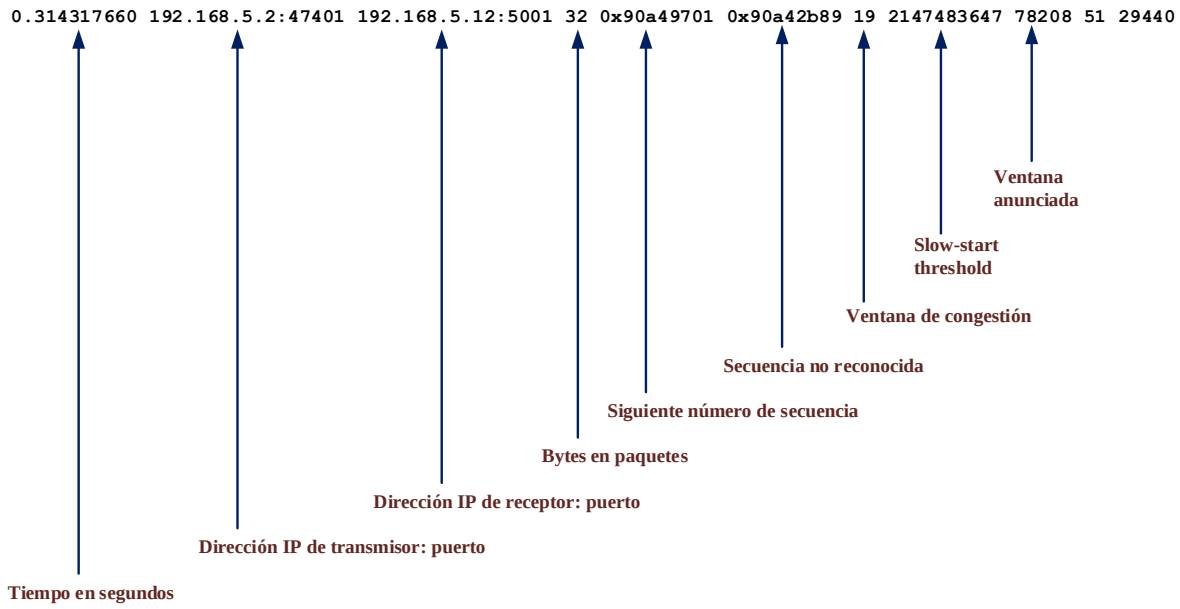


Figura 16. Parámetros de una línea generada con TCP Probe.

Cada una de las filas del archivo de captura de TCP Probe muestra los datos, tal y como se observa en la figura 16.

1.2.3. Tshark

TShark es un analizador de protocolos de red. Éste permite capturar paquetes de datos de una red viva, o lee los paquetes de un archivo de captura guardado previamente, ya sea la impresión de una forma decodificada de esos paquetes a la salida estándar o escribiendo los paquetes a un archivo. EL Formato nativo de archivo de captura de TShark es **pcap**.

Sin ningún conjunto de opciones, TShark utilizará la librería pcap para capturar el tráfico de la primera interfaz de red disponible y muestra una línea de resumen en la salida estándar para cada paquete recibido. TShark es capaz de detectar, leer y escribir en los mismos archivos de captura que son compatibles con Wireshark.

1.2.3.1. Captura de paquetes con Tshark

Se puede especificar que se realice la captura de paquetes en una cantidad de segundos especificada y que el informe de captura sea escrito en un archivo específico. También se puede especificar el puerto TCP sobre el cual se realiza la captura.

La sintaxis de captura con tshark es:

```
tshark -a duration: [t] -ni [interfaz de red] 'port [X]' -w /ruta/archivo.cap
```

En la línea anterior: La captura se realiza durante [**t]** segundos, sobre [interfaz de red], puerto [X]. El informe de captura se guarda en una archivo, indicándose su ruta: /ruta/archivo.cap
Los archivos de extensión ".cap" se encuentran escritos en formato pcap, y los mismos pueden ser visualizados correctamente con Wireshark, más no con un editor de textos.

1.2.3.2. Captura de paquetes con Tshark de la ejecución de iPerf.

Para que una ejecución de iPerf de duración 60 segundos y destino 192.168.5.12, sea capturada por tshark, se emplea la línea de comando:

```
iperf -i 10 -t 60 -c 192.168.5.12 & tshark -a duration:60 -ni eth0 'port 5001' -w /tmp/capture.cap
```

El símbolo & permite la ejecución simultánea de los comandos iPerf y tshark.

1.2.3.3. Conversión a formato de texto.

La conversión de un archivo de formato pcap a un archivo de texto legible, se realiza por la sintaxis de línea de comando:

```
tshark -r [archivo en formato pcap] -T text > [archivo de texto]
```

Donde:

-r : Lee el archivo en formato cap.

-T : Convierte el archivo leído a otro formato especificado.

text : Formato de texto especificado.

Ejemplo: `tshark -r /tmp/capture.cap -T text > /tmp/capture.txt`

1.2.4. Otras herramientas

Existen otras herramientas opcionales que brindan información sobre una conexión TCP, de estas se puede destacar "tcptrace" que analiza el resultado del archivo de captura de una conexión y "sock" que es un comando antiguo que permite conocer la disponibilidad y estado de puertos TCP para una conexión. La descripción de estas herramientas se encuentra en el Anexo C.

1.3. Herramienta de análisis

Para el análisis de datos se emplea el software Matlab, el cual permite extraer la información de los archivos de captura generados por las herramientas de extracción de datos TCP Probe y tshark, procesar esta información y representarla gráficamente.

2. Características y funcionamiento del banco de pruebas.

El banco de pruebas consiste en 4 estaciones terminales y una estación de análisis de datos, donde todos ellos se encuentran dentro de un mismo segmento de red. Se recomienda que el banco de pruebas tenga las siguientes características:

a) Estación de trabajo: Cada una de las estaciones de trabajo debe poseer lo siguiente:

- Un sistema operativo Linux. Se recomienda una distribución liviana, por ejemplo: Lubuntu.
- Una tarjeta de red (Fast-Ethernet o Gigabit-ethernet).
- Cabeceras y código fuente correspondientes a la versión de kernel Linux empleada: deben estar instalados y configurados, con el objetivo de permitir implementar, elegir o modificar un protocolo de transporte en el mismo. Se da un mayor detalle del concepto de código fuente y su proceso de instalación en el Anexo "B".
- Herramientas de emulación de perdidas (netem e iptables), con uso habilitado.
- Herramientas de extracción de datos iPerf, tcpprobe y tshark instalados.
- Cliente SSH habilitado.

b) Estación de análisis (Analizador)

- Sistema operativo Linux.
- Cumplir con los requisitos recomendados para la instalación del programa Matlab.
- Herramienta de análisis Matlab instalado (Con licencia vigente).
- Servidor SSH habilitado.
- Software estadístico "R" instalado (opcional).

c) Adicionalmente se requiere un equipo de red que permita la conexión de las estaciones (switch), para la conexión de las mismas en un segmento de red.

El funcionamiento del banco de pruebas consiste en los siguientes pasos:

i) Configuración del protocolo de transporte a ser empleado:

La variante de TCP y conjunto de opciones a ser empleados son "fijados" en la estación o estaciones que transmitirán la información, a fin evaluar su comportamiento.

ii) Ejecución de la transmisión, extracción y exportación de datos.

- Se configura el valor de RTT y porcentaje de pérdidas en la(s) estación(es) de transmisión mediante comandos de "netem". Si el objetivo es emular pérdidas constantes, se configura línea(s) de comandos con "iptables" en la(s) estación(es) de recepción.
- En la(s) estación(es) receptora(s) se ejecuta iPerf en modo servidor antes que sea ejecutado iPerf cliente en la estación de la cual se transmitirá los datos.
- En la(s) estación(es) transmisora(s) simultáneamente se inicializa la captura de tráfico con **TCP Probe** y **tshark** en el puerto TCP que empleará iPerf para la transmisión y se direcciona que ambas capturas sean copiadas a archivos en formatos de texto y pcap respectivamente, y se ejecuta iPerf en modo cliente indicando la estación destino y un tiempo fijo de la transmisión. Las capturas de tráfico con TCP Probe y tshark deben haber sido configuradas con el tiempo de duración que emplea la ejecución de iPerf en modo cliente. En caso de emplearse transmisiones simultáneas se configura un número de puerto distinto para la ejecución de iPerf, y se realizan capturas en cada puerto TCP.

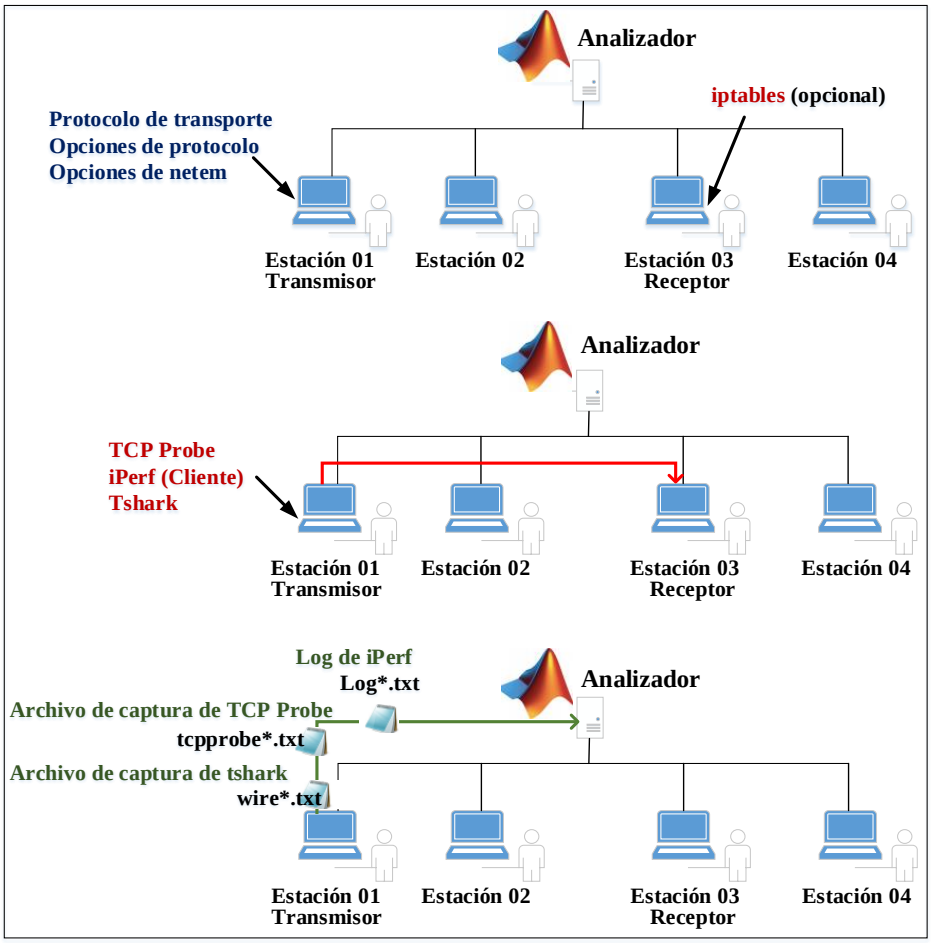


Figura 17. Funcionamiento de banco de pruebas (Una transmisión).

- Cuando concluye la transmisión, se finaliza el proceso iniciado por TCP Probe y tshark y se obtiene los archivo de captura. La información de la captura de tshark se copia en un archivo en formato pcap, el cual luego es convertido a formato de texto legible. Los archivos de captura en formato de texto legible generados por TCP Probe y tshark son exportados a la estación de análisis. El log de ejecución de iPerf también es exportado a la estación de análisis.

iii) Análisis de datos.

La información de archivos de captura generados por TCP Probe y tshark son procesados con Matlab, el cual muestra los resultados de la captura de la transmisión numérica y gráficamente.

2.1. Configuración del protocolo de transporte a ser empleado.

El protocolo TCP que se encuentra activado por defecto en un sistema operativo Linux es CUBIC, el cual es activado mediante la carga del archivo **tcp_cubic.ko**, ubicado en <Directorio de código fuente>/net/ipv4 (Ver detalles relacionados a código fuente en Anexo B).

La verificación del protocolo activo se realiza mediante la línea de comandos.

```
sudo sysctl net.ipv4.tcp_congestion_control
```

Si se desea utilizar un protocolo diferente al protocolo por defecto, y éste se encuentra dentro del "stack" de protocolos de transporte del sistema operativo, se emplea la sintaxis de comando:

```
sysctl –w net.ipv4.tcp_congestion_control = <Nombre de protocolo>
```

<*Nombre de protocolo*> es el protocolo de transporte a ser empleado, por ejemplo: bic, highspeed, vegas, etc., cuyos archivos de extensión ".ko" se ubican en <Directorio de código fuente>/net/ipv4.

Si se desea activar un protocolo no hallado en el código fuente, y asumiéndose que el nuevo protocolo se llama "**new**", y que para la programación del mismo se crea el archivo **tcp_new.c**, se realizan los siguientes pasos para la configuración del nuevo protocolo:

- El archivo **tcp_new.c** es creado en <Directorio de código fuente>/net/ipv4, y se programa el código para la ejecución del nuevo protocolo, indicándose en el mismo que el nombre del protocolo es **new**.
- Se accede a la edición del archivo **Makefile**, dentro de la carpeta en <Directorio de código fuente>/net/ipv4, y se observa que originalmente se encuentra una línea escrita:

  obj-$(CONFIG_TCP_CONG_YEAH) += tcp_yeah.o

- En ésta línea **tcp_yeah.o** debe ser remplazado por el nombre de archivo que contiene el código fuente que deseamos cargar pero empleando la extensión ".o". En este caso el texto es remplazado por **tcp_new.o**.
- Dentro del archivo Makefile, se añaden las líneas al final del texto:

```
KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)
default:
$(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
```

```
obj-$(CONFIG_TCP_CONG_WESTWOOD) += tcp_westwood.o
obj-$(CONFIG_TCP_CONG_HSTCP) += tcp_highspeed.o
obj-$(CONFIG_TCP_CONG_HYBLA) += tcp_hybla.o
obj-$(CONFIG_TCP_CONG_HTCP) += tcp_htcp.o
obj-$(CONFIG_TCP_CONG_VEGAS) += tcp_vegas.o
obj-$(CONFIG_TCP_CONG_VENO) += tcp_veno.o
obj-$(CONFIG_TCP_CONG_SCALABLE) += tcp_scalable.o
obj-$(CONFIG_TCP_CONG_LP) += tcp_lp.o
obj-$(CONFIG_TCP_CONG_YEAH) += tcp_new.o
obj-$(CONFIG_TCP_CONG_ILLINOIS) += tcp_illinois.o
obj-$(CONFIG_MEMCG_KMEM) += tcp_memcontrol.o
obj-$(CONFIG_NETLABEL) += cipso_ipv4.o
obj-$(CONFIG_XFRM) += xfrm4_policy.o xfrm4_state.o xfrm4_input.o \
                      xfrm4_output.o
KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)
default:
        $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
```

Figura 18. Archivo Makefile modificado

- Concluida la edición de Makefile, dentro de la carpeta <Directorio de código fuente>/net/ipv4, se ejecuta el comando sudo make, y se generan los archivos de extensiones: ".ko", ".o", ".mod.o" y ".mod.c": tcp_new.o, tcp_new.ko (módulo a ocupar), tcp_new.mod.o y tcp_new.mod.ko.
- Se realiza la carga del archivo generado tcp_new.ko mediante el comando **insmod**:

```
sudo insmod tcp_new.ko
```

Para verificar la carga, se ejecuta el comando **lsmod**.

Si se desea remover la carga del archivo de extensión ".ko" se emplea el comando rmmod:

```
sudo rmmod tcp_new.ko
```

- Se realiza la activación del protocolo mediante el comando sysctl:

```
sudo sysctl –w net.ipv4.tcp_congestion_control = new
```

2.1.1. Configuración de variables de TCP.

Las variables TCP forman parte del protocolo, y su configuración es accesible mediante la línea de comandos de distribuciones de Linux.

Las variables definidas pueden variar de una versión de kernel a otra. Para conocer las variables de TCP disponibles a ser modificadas en la distribución de Linux, se ejecuta la línea de instrucción en la consola de la estación transmisora:

```
sudo sysctl.net.ipv4 | grep tcp | sort
```

Para una versión de kernel 3.13.0-32 las variables de tcp se muestran en la figura 19. Cada una de las variables TCP se muestra de la siguiente forma: **net.ipv4.<Nombre_de_variable>.**

```
net.ipv4.tcp_abort_on_overflow = 0
net.ipv4.tcp_adv_win_scale = 1
net.ipv4.tcp_allowed_congestion_control = cubic reno
net.ipv4.tcp_app_win = 31
net.ipv4.tcp_autocorking = 1
net.ipv4.tcp_available_congestion_control = cubic reno
net.ipv4.tcp_base_mss = 512
net.ipv4.tcp_challenge_ack_limit = 100
net.ipv4.tcp_congestion_control = cubic
net.ipv4.tcp_dsack = 1
net.ipv4.tcp_early_retrans = 3
net.ipv4.tcp_ecn = 2
net.ipv4.tcp_fack = 1
net.ipv4.tcp_fastopen = 1
net.ipv4.tcp_fastopen_key = 00000000-00000000-00000000-00000000
net.ipv4.tcp_fin_timeout = 60
net.ipv4.tcp_frto = 2
net.ipv4.tcp_fwmark_accept = 0
net.ipv4.tcp_keepalive_intvl = 75
net.ipv4.tcp_keepalive_probes = 9
net.ipv4.tcp_keepalive_time = 7200
net.ipv4.tcp_limit_output_bytes = 131072
net.ipv4.tcp_low_latency = 0
net.ipv4.tcp_max_orphans = 16384
net.ipv4.tcp_max_syn_backlog = 128
net.ipv4.tcp_max_tw_buckets = 16384
net.ipv4.tcp_mem = 91974   122633    183948
net.ipv4.tcp_min_tso_segs = 2
net.ipv4.tcp_moderate_rcvbuf = 1
net.ipv4.tcp_mtu_probing = 0
net.ipv4.tcp_no_metrics_save = 0
net.ipv4.tcp_notsent_lowat = -1
net.ipv4.tcp_orphan_retries = 0
net.ipv4.tcp_reordering = 3
net.ipv4.tcp_retrans_collapse = 1
net.ipv4.tcp_retries1 = 3
net.ipv4.tcp_retries2 = 15
net.ipv4.tcp_rfc1337 = 0
net.ipv4.tcp_rmem = 4096    87380     6291456
net.ipv4.tcp_sack = 1
net.ipv4.tcp_slow_start_after_idle = 1
net.ipv4.tcp_stdurg = 0
net.ipv4.tcp_syn_retries = 6
net.ipv4.tcp_synack_retries = 5
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_thin_dupack = 0
net.ipv4.tcp_thin_linear_timeouts = 0
net.ipv4.tcp_timestamps = 1
net.ipv4.tcp_tso_win_divisor = 3
net.ipv4.tcp_tw_recycle = 0
net.ipv4.tcp_tw_reuse = 0
net.ipv4.tcp_window_scaling = 1
net.ipv4.tcp_wmem = 4096   16384     4194304
net.ipv4.tcp_workaround_signed_windows = 0
```

Figura 19. Variables de TCP en una distribución con kernel 3.13.0-32.

La descripción de las variables de TCP de interés y los valores a los cuales se ajustan se detallan en la sección 1 del Anexo "D".

Las variables TCP son fijadas en el archivo script **opciones_tcp**, el cual es localizado en el directorio de trabajo /home/alumno (Ver sección 2 del Anexo "D").

Otro aspecto de importancia para una conexión TCP es la configuración del valor del búfer de los sockets del sistema operativo. Dos archivos importantes en un S.O. Linux que contienen opciones de socket son:

a) /proc/sys/net/core/rmem_max

Configura la opción de socket "SO_RCVBUF", para determinar el máximo tamaño del búfer del socket de recepción (expresado en bytes). El mínimo valor de esta opción es 256 [21].

Esta opción se puede visualizar en el S.O emplean el comando: **`sysctl net.core.rmem_max`**.

Y es modificable, empleando la sintaxis de línea de comando:

`sysctl –w net.core.rmem_max = <Nuevo valor>`.

b) /proc/sys/net/core/wmem_max

Configura la opción de socket "SO_SNDBUF", para determinar el máximo tamaño del búfer del socket de envío (expresado en bytes). El mínimo valor de esta opción es 2048 [21].

Esta opción se puede visualizar en el S.O emplean el comando: sysctl net.core.wmem_max.

Y es modificable, empleando la sintaxis de línea de comando:

`sysctl –w net.core.wmem_max = <Nuevo valor>`.

Para el presente trabajo, se considera los valores por defecto de la versión de kernel 3.13.0-32, en la cual se identifican: net.core.rmem_max= 163840 y net.core.wmem_max= 163840.

2.2. Ejecución de la transmisión, extracción y exportación de datos.

2.2.1. Ejecución en el receptor.

a) Emulación de transmisión con pérdidas aleatorias

En la consola del receptor se ejecuta la instrucción: iperf –s.

b) Emulación de transmisión con pérdidas determinantes (constante).

Para el ejemplo de un porcentaje de pérdidas del 1% en la consola del receptor se ejecuta las siguientes líneas de comando:

```
sudo iptables –A INPUT –i eth0 –p tcp –s 192.168.1.2  --destination-port 5001 –m
statistic –mode nth --every 100 --packet 99 –j DROP
iperf -s
```

2.2.2. Ejecución en el transmisor, extracción y exportación.

La ejecución en el transmisor se encuentra definida en el script **/home/alumno/run,** el cual incluye la ejecución del script **opciones_tcp** (Ver Anexo D). El archivo **run** es modificable. Para el siguiente ejemplo se ejecuta el script para una transmisión con pérdidas aleatorias del 1% y un RTT de 50ms.

```
#!/bin/bash
#Ejecución de una transmisión con RTT y pérdidas definidos.
tc qdisc delete dev eth0 root netem
tc qdisc add dev eth0 root netem delay 50ms loss 1%
# Para simular perdidas constantes borre el parámetro loss, en la línea
anterior.
./opciones_tcp
modprobe tcp_probe port=5001 full=1
cat /proc/net/tcpprobe >/tmp/salida.out &
CAP=$!
iperf -i 10 -t 60 -c 192.168.5.12 >log.txt & tshark -a duration:60 -ni
    eth0 'port 5001' -w /tmp/capture.cap
kill $CAP
tshark -r /tmp/capture.cap -T text > /tmp/capture.txt
sshpass -p 'contraseña' scp /tmp/capture.txt owl-wkstn-tcp@192.168.5.10:/
    home/owl-wkstn-tcp/experimento/wire1.txt
sshpass -p 'contraseña' scp /tmp/salida.out owl-wkstn-tcp@192.168.5.10:/
    home/owl-wkstn-tcp/experimento/tcpprobe1.txt
sshpass -p 'contraseña' scp /tmp/log.txt owl-wkstn-tcp@192.168.5.10:/home/
    owl-wkstn-tcp/experimento/log1.txt
```

Figura 20. /home/alumno/run

En la figura 20, la transmisión considera pérdidas aleatorias y los archivos que muestran la captura de iPerf, TCP Probe y tshark, son log1.txt, tcpprobe1.txt y wire1.txt, los cuales son copiados a la estación donde se realizará el análisis con Matlab. Si se requiere ejecutar una transmisión con pérdidas constantes, se borra "loss 1%" de la línea correspondiente del script, y se define las pérdidas en el receptor, antes de la transmisión.

Para la exportación de datos capturados, se requiere la instalación de openssh-cliente en las estaciones de transmisión, y la instalación de openssh-server en la estación de análisis. La exportación de datos se realiza mediante el comando "sshpass" como se observa en la figura 20, siendo la sintaxis de línea de comando para realizar la copia de archivos a otro host:

sshpass –p 'password' scp [ruta archivo local] [nombre equipo destino]@[Dirección IP]:[ruta archivo destino]

Ejemplo: Copia del archivo de captura generado por tshark capture.txt

```
sshpass -p 'contraseña' scp /tmp/capture.txt owl-wkstn-tcp@192.168.5.10:/home/owl-wkstn-
tcp/experimento/wire2.txt
```

2.3. Analisis de los datos extraidos de la transmisión TCP.

2.3.1. Analizador de los datos extraídos de una única muestra de conexión TCP.

2.3.1.1. Obtención de los datos desde TCP Probe.

Cuando se obtiene el archivo TCPProbe, se procede a extraer la data del mismo, empleando MATLAB, procedimiento que consiste en los siguientes pasos:

a) Importación de datos.

La importación de los datos se realiza con la línea de instrucción en MATLAB:

```
[pbfile path] = uigetfile('tcpprobe*.*','TCPprobe File');
```

Se nombra a cada archivo generado que será importado como tcpprobe*.*. Por ejemplo si se tienen "n" muestras se nombra a los archivos: tcpprobe1.txt, tcpprobe2.txt, tcpprobe3.txt, etc.

b) Extracción de datos.

Mediante la siguiente línea de instrucción en Matlab, se emplea la función **tcpprobe_reader**, cuyo detalle se muestra en el Anexo "E".

```
[pbtime_i, pbcwnd_i, ssthresh] = tcpprobe_reader([path, pbfile], rtt);
```

"pbtime_i", "pbcwnd_i", "ssthresh", son los vectores que contienen los valores de tiempo, cwnd (ventana de congestión) y el umbral (ssthresh).

La función **tcpprobe_reader** realiza los siguientes pasos:

- Definición de la función.- La línea de instrucción es:

```
function [time, cwnd, ssthresh] = tcpprobe_reader(fname, rtt)
```

"fname" representa el archivo, cuya información se desea utilizar y rtt es el valor de "tiempo de ida y retorno" de la muestra.

- Importación del texto en una celda.- Se realiza mediante las siguientes instrucciones:

```
fileID = fopen(fname);
C = textscan(fileID, '%s', 'delimiter','\n');
fclose(fileID);
```

"C" es una celda de texto, de "L" líneas de texto.

- Inicialización de valores.- Se inician los valores:

  L : Longitud de la celda "C", y determina la longitud de los vectores.

  time : Vector a contener el tiempo de cada muestra que genera TCP Probe.

  cwnd : Vector a contener la ventana de congestión de cada muestra de tiempo.

  ssthresh : Vector a contener el valor del umbral "ssthresh" de cada muestra de tiempo.

- Extracción de valores de tiempo, ventana de congestión y "slow-start threshold".

Se realiza la conversión de la celda “C” a matriz. La nueva matriz contiene “L” filas por una cantidad desconocida de columnas.

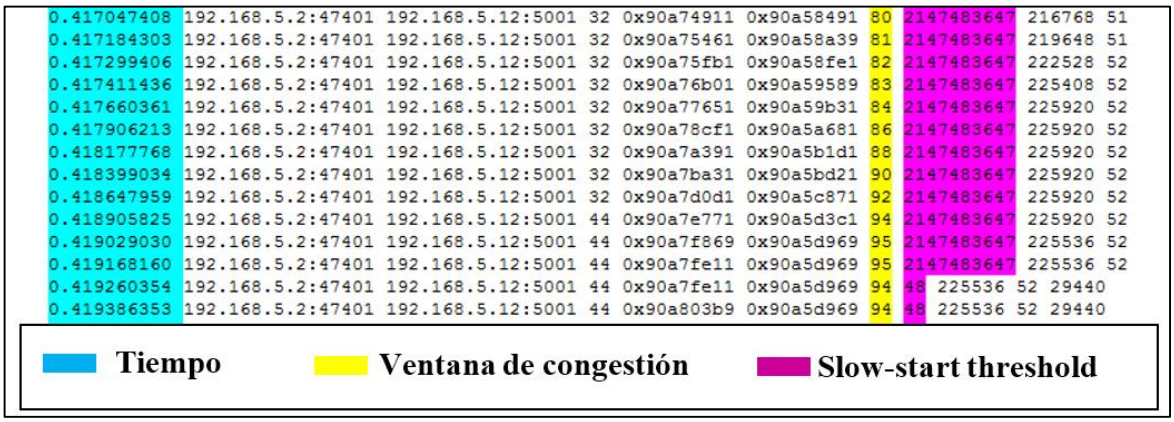


Figura 21. Archivo generado por TCP Probe. (Variables de interés)

Nuestras variables de interés son: Tiempo, ventana de congestión y “slow-start threshold”. Los valores resaltados en la figura 21 constituirán los nuevos vectores “time”, “cwnd” y “ssthresh” mediante la función tcpprobe_reader. Los valores de “time” pueden ser manejados de tal modo que el primer valor time(1)=rtt, mediante la instrucción: `time=time-time(1)+rtt;` Los vectores de tiempo, ventana de congestión y “slow-start threshold” al ser invocados en el archivo de ejecución principal serán los vectores “pbtime_i”, “pbcwnd_i” y “ssthresh” respectivamente.

2.3.1.2. Obtención de los datos desde Tshark

Cuando se obtiene el archivo Tshark, se procede a extraer la data del mismo, empleando MATLAB, procedimiento que consiste en los siguientes pasos:

a) Importación de datos.

La importación de los datos se realiza con la línea de instrucción en MATLAB:

```
[fshark path] = uigetfile('wire*.*','Archivo shark');
```

Para la correcta importación, se nombrará a cada archivo generado como wire*.*. Por ejemplo, para “n” muestras se nombra a los archivos: wire1.txt, wire2.txt, wire3.txt, etc.

b) Extracción de datos.

Mediante la siguiente línea de instrucción en Matlab, se emplea la función shark_reader, cuyo detalle se muestra en el Anexo “E”

```
[time_i, ACK, SEQ, LEN, REORDER, RTX] = shark_reader([path, fshark]);
```

"time_i", "ACK", "SEQ" y "LEN" son los vectores que contienen los valores de tiempo, número de reconocimiento de paquete (ACK), número de secuencia y longitud de MSS.

El vector REORDER tiene "L" valores, siendo L la cantidad de elementos capturados, ya sea referente a paquetes enviados o recibidos

Del conjunto de "L" elementos, existen paquetes que han sido marcados como "Out-of-order" por Tshark, éstos son definidos con el valor 1 en el vector REORDER, mientras aquellos que no lo son, son definidos con el valor 0.

Los paquetes marcados como "Out-of-order" son paquetes transmitidos después de tiempo. Se asumen que estos son "paquetes retransmitidos", que por error tshark, asumió que llegaron a fuera de orden (Esto se comprobó al ejecutar transmisiones con pérdidas constantes).

El vector RTX también posee "L" elementos, cuyos valores son 1 y 0, dependiendo de:

- Si representa un paquete retransmitido, el valor es de 1.
- Caso contrarios, el valor es de 0.

La función shark_reader realiza los siguientes pasos:

- Definición de la función.- La línea de instrucción es:

```
function [TIME, ACK, SEQ, LEN, REORDER, RTX ] = shark_reader(fname)
```

"fname" representa el archivo, cuya información se desea utilizar.

- Importación del texto en una celda.- Se realiza mediante las siguientes instrucciones:

```
fileID = fopen(fname);
C = textscan(fileID, '%s', 'delimiter','\n');
fclose(fileID);
```

"C" es una celda de texto, de "L" líneas de texto.

- Inicialización de valores.- Se inician los valores:

  L : Longitud de la celda "C", y determina la longitud de los vectores.

  TIME: Vector a contener el tiempo de cada muestra que genera Tshark.

  ACK : Vector a contener el número de ACK de cada paquete enviado o recibido.

  SEQ : Vector a contener el número de secuencia de cada paquete enviado o recibido.

  LEN : Vector que define el valor de MSS de cada segmento

  REORDER: Vector que define si el paquete ha sido marcado como fuera de orden por Tshark. (Requiere ser reordenado para el análisis)

  RTX : Vector que define si el paquete ha sido marcado como retransmitido por Tshark.

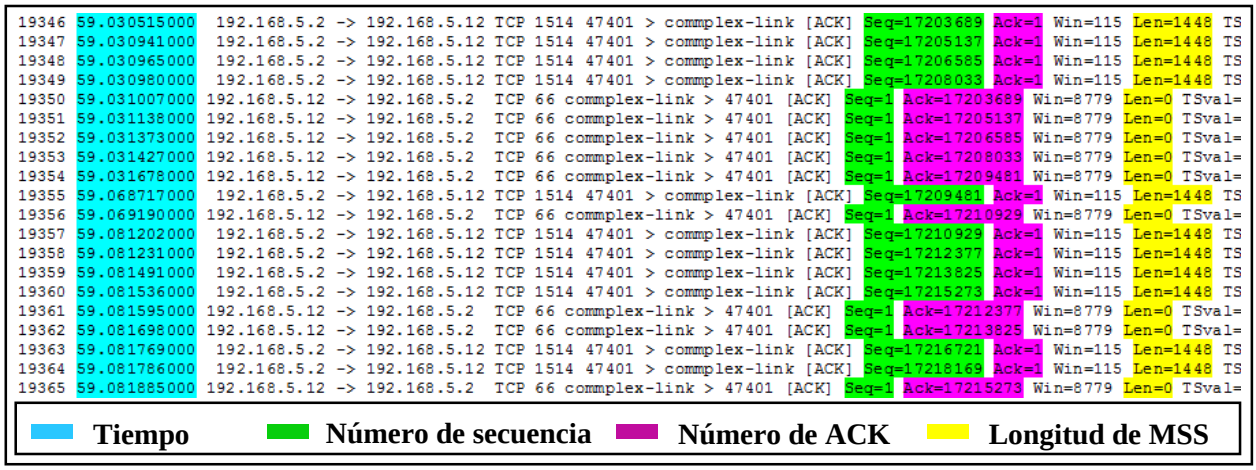


Figura 22. Archivo generado por TSHARK. (Variables de interés)

- Extracción de valores de tiempo, número de ACK, número de secuencia, longitud de segmento (MSS) e indicador de paquete reordenado o retransmitido.

  Se realiza la conversión de la celda "C" a matriz. La nueva matriz contiene "L" filas por una cantidad desconocida de columnas.

  Nuestras variables de interés son: tiempo, número de ACK, número de secuencia y longitud de segmento (MSS).

  Los valores resaltados en la figura anterior constituirán los vectores TIME, ACK, SEQ y LEN de la función shark_reader, y al ser invocada esta función en el archivo principal constituyen los vectores: time_i, ACK, SEQ y LEN.

  La figura 23 muestra paquete fuera de orden, es decir éste fue capturado por Tshark después que uno o varios paquetes con número de secuencia posterior, la ubicación de aquellos paquetes marcados con "TCP Out-of-Order" es representada en el vector REORDER.

  En el archivo de captura de Tshark, se muestran dos tipos de paquetes retransmitidos: Los paquetes marcados como "TCP Fast Retransmission" siguen el algoritmo de Retransmisión rápida, es decir el paquete es retransmitido cuando se produce "Triple ACK duplicado", y aquellos paquetes marcados como "TCP Retransmission" son retransmitidos a causa de que el tiempo de espera de reconocimiento ha expirado.

  Cuando existe un paquete retransmitido se toma la ubicación, y ésta se representa en el vector RTX.

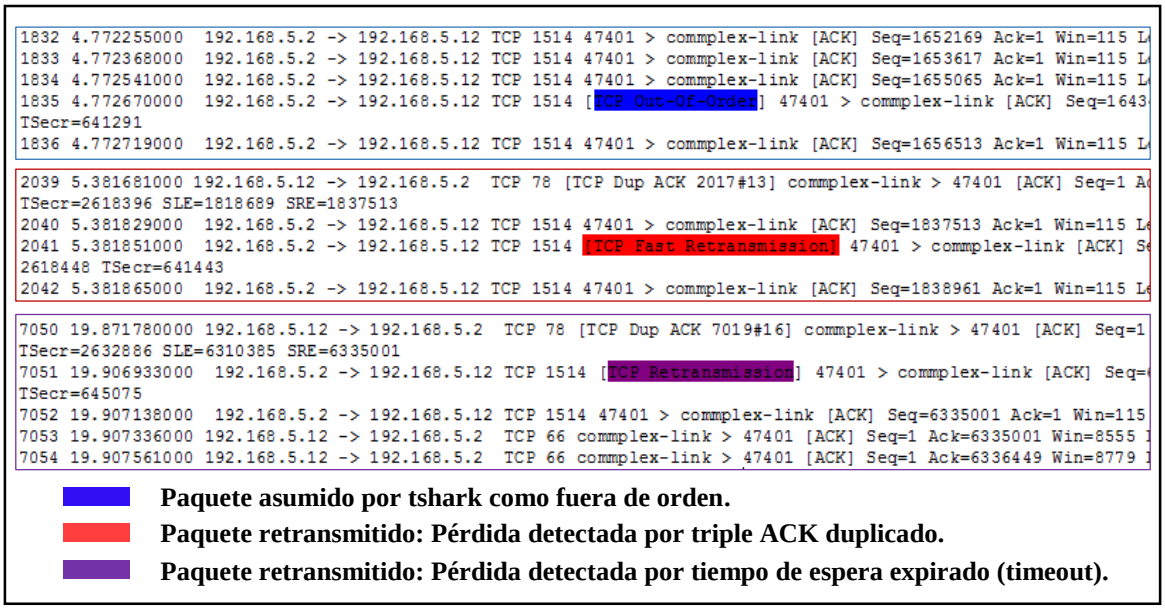


Figura 23. Archivo generado por TSHARK. (Paquetes fuera de orden y retransmitidos)

2.3.1.3. Tratamiento de datos obtenidos desde TCP Probe.

El tratamiento de los datos extraídos en los vectores pbtime_i, pbcwnd_i y ssthresh, a los cuales llamaremos vectores iniciales, se realiza mediante la función en MATLAB llamada **tcpprobe_values.**

```
function [time, cwnd, slope, decrease, thrput, beta_reno, beta_gaimd] =
tcpprobe_values(time_i, cwnd_i, ssthresh, mss, step)
```

Donde:

time : Vector de tiempo, cuya diferencia entre dos elementos contiguos es un RTT.

cwnd : Vector que contiene valores de ventana correspondiente a cada valor de tiempo mostrado en el vector time.

slope : Vector que contiene los valores de pendiente.
Estos valores representan el incremento aditivo de la ventana de congestión.

decrease: Vector que contiene el valor de decremento real. Esto es:
decrease = (“cwnd” después de la pérdida)/(“cwnd” al momento de la pérdida)

thrput : Valor de un rendimiento (bits/segundo) aproximado de la transmisión capturada.

beta_reno: “β” cuando la transmisión emplea TCP RENO como protocolo de transporte.

beta_gaimd: “β” cuando la transmisión emplea GAIMD como protocolo de transporte.

time_i, cwnd_i y ssthresh son los vectores de longitud “L” obtenidos al aplicar la función tcpprobe_reader, mss es el valor de MSS en bytes y step es el valor del RTT.

El procedimiento que realiza la función **tcpprobe_values** consiste en los siguientes pasos:

a) Se obtiene un vector **slope_i** que contiene los valores de incremento aditivo **α**, hallados a partir de los valores de cwnd_i, dividiendo cada valor cwnd_i(i) por la cantidad de veces que este valor se repite antes de cambiar de valor. Por ejemplo dados algunos valores de **cwnd_i**:

n	200	201	202	203	204	205	206
cwnd_i(n)	4	5	5	5	5	5	6

El valor de 5 se repite desde cwnd_i(201) hasta cwnd_i(205) un total de 5 veces. Por lo tanto: slope_i(201:205) =1 (5 dividido por 5). Esto representa que está ocurriendo un incremento aditivo de 1. Los valores de "slope_i" sólo son calculados en etapas cuando "cwnd" crece.

b) Se obtiene el vector **time** al filtrar los valores hallados en el vector **time_i** que sean iguales o inmediatamente mayores a un valor "$n \times RTT$", donde n=1, 2, 3,… y el valor de RTT es el tiempo de ida y vuelta. Por ejemplo dados los primeros 8 elementos del vector time_i, y conociéndose RTT = 0.05 se comparan con la condición explicada:

Elemento de "time"	Condición	Valor comparado
time_i(1) = 0.05000	**es igual a**	**1×RTT**
time_i(2) = 0.10067	**es el primer número mayor a**	**2×RTT**
~~time_i(3) = 0.10076~~		
time_i(4) = 0.15121	**es el primer número mayor a**	**3×RTT**
~~time_i(5) = 0.15131~~		
~~time_i(6) = 0.15142~~		
~~time_i(7) = 0.15152~~		
time_i(8) = 0.20176	**es el primer número mayor a**	**4×RTT**

Después de realizar un conjunto de instrucciones de programación se obtiene el vector time de longitud "length(time)", donde los 4 primeros valores son:

time(1) = time_i(1) = 0.05000

time(2) = time_i(2) = 0.10067

time(3) = time_i(4) = 0.15121

time (4) = time_i(8) = 0.20176

Asimismo los vectores cwnd y slope tendrán una longitud "length(time)" construidos a partir de las mismas ubicaciones cuyos valores extrajo **time** de **time_i**, pero de cwnd_i y slope_i respectivamente. Es decir para el ejemplo dado:

cwnd(1) = cwnd_i(1) slope(1) = slope_i(1)

cwnd(2) = cwnd_i(2) slope(2) = slope_i(2)

cwnd(3) = cwnd_i(4) slope(3) = slope_i(4)

cwnd(4) = cwnd_i(8) slope(4) = slope_i(8)

c) Se calcula el rendimiento "thrput" a partir del promedio de ventana de congestión, mediante la línea de instrucción:

```
thrput = mean(cwnd)*8*mss/step;
```

"mss" es el tamaño del segmento en bytes y step es el valor de RTT.

d) Se obtiene un vector dec_i de longitud "L", donde cada elemento equivale al valor del nuevo "slow-start threshold" dividido por la ventana de congestión cuando se produce una congestión.

Esto es:

Si cwnd_i(i) < cwnd_i(i-1) y cwnd(i) ≥ ssthresh(i):

dec_i(c) = ssthresh(i)/cwnd(i-1)

Donde "c", inicia en 1 e incrementa aditivamente en 1 unidad conforme se produce una nueva congestión.

Es decir existen los primeros "N" elementos de **dec_i** con un valor de decremento real.

Entonces se calcula: **dec_pr = dec_i(1:N)**

El tiempo en que se produce la congestión (por ende ocurre un decremento) se registra en el vector **timedec_pr** de longitud "N".

Por ejemplo, si se presentaron 20 congestiones en la transmisión, dec_pr y timedec_pr tienen 20 valores de decremento multiplicativo y de tiempo respectivamente.

Se inicializa el vector **decrease** como un vector de longitud igual a **time**, y contendrá los valores de decremento multiplicativo para cada elemento de **time** igual o inmediatamente mayor a cada elemento de **timedec_pr.** Cuando no ocurre congestión el decremento es un valor "NaN" (Se detalla instrucciones en Anexo E).

e) beta_reno, es un vector cuyos elemento son iguales a 0.5 cuando:

("cwnd" después de la pérdida) = 0.5 × ("cwnd" al momento de la pérdida)

Donde ambos valores de "cwnd" son números enteros positivos.

f) beta_gaimd, es un vector cuyos elemento son iguales a 0.875 cuando:

("cwnd" después de la pérdida) = 0.875 × ("cwnd" al momento de la pérdida)

Donde ambos valores de "cwnd" son números enteros positivos.

La función tcpprobe_values es llamada en el programa principal mediante una línea de instrucción:

```
[pbtime, pbcwnd, pbslope, pbdec, pbthrput_total, beta_reno, beta_gaimd] =
tcpprobe_values(pbtime_i, pbcwnd_i, ssthresh, mss, rtt);
```

2.3.1.4. Tratamiento de datos obtenidos desde Tshark.

El tratamiento de los datos extraídos en los vectores time_i, SEQ, LEN, ACK, REORDER y RTX, a los cuales llamaremos vectores iniciales, se realiza por medio de la función en MATLAB llamada **shark_values.**

```
function [time, wscwnd, pktsend, thrput, beta_estp] = shark_values(time_i, SEQ, LEN,
ACK, REORDER, RTX, MSS, step, tau)
```

Donde:

tau : Valor de τ para ESTP.

time : Vector de tiempo, cuya diferencia entre dos elementos contiguos es un RTT.

wscwnd: Vector que contiene valores de ventana correspondiente a cada valor de tiempo mostrado en el vector time.

pktsend: Vector en el cual cada elemento pktsend(n) equivale al número de paquetes enviados a un tiempo time(n).

thrput : Valor de un rendimiento (bits/segundo) aproximado de la transmisión capturada.

beta_estp: Valores de "β" cuando se emplea el protocolo ESTP.

Los vectores time_i, SEQ, LEN, ACK REORDER y RTX son obtenidos mediante la función shark_reader previamente. Los valores de MSS y step corresponden al tamaño máximo del segmento y RTT de la transmisión.

La función **shark_values** realiza los siguientes pasos:

a) Se realiza un filtrado de los segmentos para obtener los valores correspondientes a segmentos que son enviados desde el transmisor, que serán aquellos con una longitud diferente de 0.
   Se obtienen nuevos time_itx, SEQtx, LEN, REORDER_i y RTX_i de longitud "Ltcp", donde Ltcp es la cantidad de capturas de segmentos enviados del archivo generado por Tshark.
   En la figura 24 las líneas resaltadas en "amarillo" corresponden a los nuevos vectores obtenidos.

b) Se define SEQtx = SEQ(LEN~=0). Es decir los números de secuencia de paquetes enviados.

c) Se define nuevos vectores:
   - ACK_3DUP: Números de secuencia de los paquetes retransmitidos bajo la condición triple ACK duplicado.

- RTX_3DUP: Cada elemento es “1” o “0”, siendo “1” cuando en la posición de ese elemento ocurre un triple ACK duplicado.
- **RTX_TOTAL:** Es el vector que representa con el valor “1” todas las posiciones de SEQtx, donde ocurrió una retransmisión de paquete perdido. Es decir realiza una operación AND entre RTX_i, REORDER_i y RTX_3DUP.

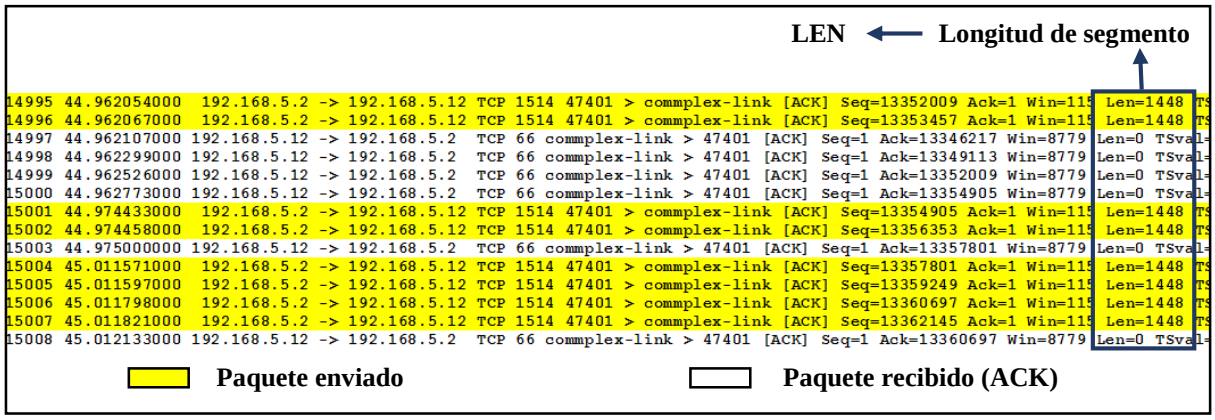


Figura 24. Archivo generado por TSHARK. (Paquetes enviados y recibidos)

d) Se emplea un nuevo vector **pktsend_i**, el cual es el número de paquete que corresponde al número de secuencia enviado. Es decir pktsend_i(n) = 1+ (SEQtx(n) –SEQtx(1))/MSS. Siendo el MSS=1448 se tiene por ejemplo:

Si SEQtx(1)=1, entonces pktsend_i(1) = 1

Si SEQtx(2)=1449, entonces pktsend_i(2) = 2

Si SEQtx(13)=14481, entonces pktsend_i(13) = 11

e) Cuando es capturado un paquete retransmitido, la cantidad de paquetes enviados aumenta en 1, actualizándose en el vector denominado aquí como **“pktsend_i”**, el cual ahora indica la cantidad aproximada de paquetes pktsend_i(n) enviados a un tiempo **time_itx(n).**

Por ejemplo, dados los números de secuencia de paquetes enviados:

144801 147697 150593 152041 149145 153489 146249

Capturados por Tshark en el orden dado, y registrados en SEQtx, asumiendo MSS = 1448 bytes y el número de secuencia inicial es 1, entonces SEQtx(1) = 1 representa el primer paquete enviado, y conociéndose que no ha ocurrido ninguna pérdida antes.

Siendo los valores del vector RTX_TOTAL que corresponden a los paquetes dados:

0 0 0 0 1 0 1

El número de secuencia 144801 representa el 101° paquete enviado, y no habiendo ocurrido pérdidas antes, indica que la cantidad de paquetes enviados en esta posición es 101.

Considerando los supuestos, las cantidades de paquetes para estas posiciones de tiempo son:

101 103 105 106 107 108 109

Estos valores son actualizados en el vector pktsend_i en las posiciones dadas.

f) Se obtiene del rendimiento "thrput" conociendo los paquetes enviados (pktsend_i) y las posiciones de los paquetes retransmitidos (RTX_TOTAL).

Se emplean las siguientes líneas de instrucciones en MATLAB:

```
packetsend = pktsend_i(Ltcp);            % Paquetes enviados
timetotal = max(time_i);                 % Tiempo total de transmisión
thrput = (packetsend)*MSS*8/timetotal;
```

g) Se obtiene el vector **time** al filtrar los valores hallados en el vector **time_itx** que sean iguales o inmediatamente mayores a un valor "$n \times RTT$", donde n=1, 2, 3,… y el valor de RTT es el tiempo de ida y vuelta. Así mismo, un vector **pktsend** tendrán una longitud "length(time)" que es el valor de pktsend_i correspondiente al nuevo vector time.

h) La importancia de pksend está en que permite un cálculo aproximado de la ventana de congestión en cada tiempo time(n).

pktsend : Vector que contiene la cantidad de paquetes enviados en un valor de tiempo del vector time.

Las líneas de instrucción para obtener el vector ventana de congestión "wscwnd" son:

```
wscwnd=zeros(1,z);   %z=longitud del vector time.
wscwnd(1)= 1;
for i=2:z
        wscwnd(i)=pktsend(i)-pktsend(i-1);  % Vector de cwnd
end
```

i) El vector **beta_estp,** es calculado en base a los números de secuencia de los paquetes perdidos de ACK_DUP, calculando:

δ = (N° de secuencia del "n" paquete perdido)-(N° de secuencia del "n+1" paquete perdido)+1

Los valores de "δ" obtenidos y un valor fijo de "τ" se usan para calcular los valores de β, ubicados en el vector **beta_estp** en las posiciones en las cuales se estima se produce una pérdida.

Los vectores de tiempo, ventana de congestión, paquetes enviados, el valor de rendimiento y β para ESTP (time, wscwnd, pktsend, thrput y beta_estp) de la función shark_values, son llamados en el programa principal como **time, wscwnd, pktsend, thrput_total** y **beta_estp** respectivamente.

2.3.1.5. Vectores de rendimiento en el tiempo.

2.3.1.5.1. Rendimiento máximo en el tiempo con datos desde TCP Probe.

El rendimiento máximo depende del tamaño de la ventana de congestión, éste cuenta las pérdidas producidas. Se entiende este rendimiento máximo como la **tasa de transferencia** propiamente dicha. Las líneas de instrucción para hallar el rendimiento promedio en

```
thrput_pb = zeros(1,length(pbcwnd));
for i=1:length(pbcwnd)
        thrput_pb(i) = mean(pbcwnd(1:i))*8*mss/rtt; % Rendimiento en bits/s (TCP Probe)
end
```

Donde:

thrput_pb : Promedio de ventana de congestión (bits) a un tiempo pbtime(i) dividido por RTT

pbcwnd : Vector con los valores de ventana de congestión en cada tiempo pbtime(i)

pbtime : Vector de valores de tiempo tomados cada RTT segundos aproximadamente.

mss : Máximo tamaño del segmento transmitido (Bytes)

rtt : Tiempo de ida y vuelta (segundos)

2.3.1.5.2. Rendimiento con datos desde Tshark.

La línea de instrucción en Matlab para el cálculo del rendimiento empleando datos de tshark es:

```
thrput = 8*mss*(pktsend./time);
```

Donde:

thrput : Vector de paquetes enviados correctamente (expresado en bits) dividido por el tiempo que toma el envío de los mismos (segundos).

pktsend : Vector que muestra la cantidad de paquetes enviados por cada cantidad de tiempo en segundos.

time : Vector de valores de tiempo tomados cada RTT segundos aproximadamente.

mss : Máximo tamaño del segmento transmitido (Bytes)

2.3.1.6. Gráfica de los datos

a) Ventana de congestión en el tiempo con datos desde TCP Probe.
   Eje X: pbtime, Eje Y: pbcwnd

b) Ventana de congestión en el tiempo con datos desde Tshark.
   Eje X: time, Eje Y: wscwnd

c) Pendiente de la ventana de congestión en el tiempo
   Eje X: pbtime, Eje Y: pbslope

d) "β" en el tiempo

Si el protocolo es RENO o GAIMD: Eje X: pbtime, Eje Y: beta

Si el protocolo es ESTP ("τ" es constante y fijo): Eje X: time, Eje Y: beta

e) Decremento real de la ventana de congestión en el tiempo

Eje X: pbtime, Eje Y: pbdec

f) Rendimiento promedio en el tiempo, empleando datos de TCP Probe y tshark.

Eje X: pbtime, Eje Y: thrput_pb Eje X: time, Eje Y: thrput

g) Valores mostrados:

- MSS y RTT
- Ventana de congestión promedio: Media de "pbcwnd"
- Rendimiento empleando datos de tcpprobe: pbthrput_total
- Rendimiento empleando datos de tshark: thrput_total
- Decremento real (Media de "pbdec") y Media de "β"

Las líneas de programación explicadas en la sección 2.3.1 de este capítulo se detallan en el Anexo E.

2.3.2. Analizador de datos de varias muestras de conexión TCP.

El analizador de varias muestras permite la comparación de varias muestras de una conexión TCP punto a punto empleando los mismos valores de RTT, MSS, ventana de congestión inicial, protocolo de transporte y opciones de protocolo.

Para un conjunto de 10 muestras se obtienen 10 archivos empleando TCP Probe y 10 archivos generados por Tshark. Estos 20 archivos son importados desde MATLAB con el comando "uigetfile".

```
[pbfile1 path] = uigetfile('tcpprobe*.*','Archivo TCPprobe 1');
      ⋮                                   ⋮
[pbfile10 path] = uigetfile('tcpprobe*.*','Archivo TCPprobe 1');
[fshark1 path] = uigetfile('wire*.*','Archivo shark 1');
      ⋮                                   ⋮
[fshark10 path] = uigetfile('wire*.*','Archivo shark 1');
```

Se emplean las funciones tcpprobe_reader, tcpprobe_values, shark_reader y shark_values para hallar los valores de ventana de congestión, tiempo y rendimiento para cada muestra, y se obtienen los vectores y valores:

- **Desde TCP Probe:**

```
pbtime1, pbcwnd1, pbslope1, pbdec1, pbthrput1, beta_reno1, beta_gaimd1
      ⋮                                   ⋮
pbtime10, pbcwnd10, pbslope10, pbdec10, pbthrput10, beta_reno10, beta_gaimd10
```

Los 4 primeros son vectores de tiempo, ventana de congestión, incremento aditivo y decremento multiplicativo, el quinto valor es el rendimiento total de cada muestra, y los últimos dos vectores calculan β cuando se emplea TCP RENO o GAIMD.

- **Desde Tshark:**

```
time1, wscwnd1, pktsend1, thrput1, beta_estp1
      ⋮                                ⋮
time10, wscwnd10, pktsend10, thrput10, beta_estp10
```

Los 3 primeros son vectores de tiempo, ventana de congestión y paquetes enviados, mientras el cuarto valor es el rendimiento total de cada muestra. El último vector calcula β cuando se emplea ESTP con un valor "τ" fijo. Los valores de interés a graficar son: tiempo, ventana de congestión, rendimiento y valores de β y decremento real multiplicativo (Ver Anexo F).

2.3.2.1. Gráfica de los datos

FIGURA 1.- Se grafica:

a) Ventana de congestión en el tiempo por cada muestra (10 gráficas).- Se emplea los datos obtenidos desde TCP Probe.
b) Rendimiento obtenido de datos de TCP Probe y rendimiento obtenido de datos de Tshark por muestra (Eje Y: Rendimiento (Mb/s), Eje X: Orden de transmisión)

FIGURA 2.- Se grafica:

a) "β" y decremento real en el tiempo por cada muestra (10 gráficas).

FIGURA 1.- Se muestran los datos:

a) Tabla que incluye:
   - Columna de rendimientos de la muestra, con datos de TCP Probe.
   - Columna de rendimientos de la muestra, con datos de Tshark.
   - Columna de ventana de congestión promedio de la muestra, con datos de TCP Probe.
b) Promedio de rendimiento, promedio de ventana de congestión.

FIGURA 2.- Se muestran los datos:

a) Tabla que incluye:
   - Columna de valores de β promedio de la muestra.
   - Columna de valores de "Decremento real" promedio de la muestra.
b) Promedio total de los "β" y valores de "Decremento real".

CAPITULO IV: RESULTADOS

1. TCP TRADICIONAL

1.1. Consideraciones de TCP en Linux

1.1.1. Ventana inicial de envío

El tamaño de ventana de congestión inicial se indica en el archivo **tcp.h** ubicado en el directorio: <Directorio de código fuente>/include/linux, mediante la siguiente línea de instrucción.

```
#define TCP_INIT_CWND      <Tamaño de la ventana inicial en paquetes>
```

En el código fuente de la versión linux 3.13.0 se encuentra la ventana de congestión inicial como 10 MSS por defecto.

```
#define TCP_INIT_CWND      10
```

Es posible la modificación de este valor en el archivo tcp.h, o se puede establecer en el archivo de extensión .c, como se explica en páginas posteriores.

1.1.2. Slow-start threshold

Ésta opción puede ser fijada en una distribución Debian/Ubuntu/Lubuntu mediante sintaxis de línea de comando: **sudo sysctl -w net.ipv4.tcp_max_ssthresh= <tamaño máximo en bytes>.** Se debe tener en cuenta que el valor de "ssthresh" en bytes es igual al "ssthresh" expresado en cantidad de segmentos multiplicado por el número de bytes de un MSS.

1.1.3. Código fuente de TCP en Linux

El código fuente de TCP permite la configuración y modificación del protocolo TCP o cualquiera de sus variantes y se ubica en <Directorio de código fuente>/net/ipv4/. Comúnmente: <Directorio de código fuente> = /usr/src/<versión de kernel>.

Dentro de <Directorio de código fuente>/net/ipv4/ se encuentran los archivos de extensión .c, correspondientes a las diferentes variantes de implementación de TCP: tcp_bic.c, tcp_cubic.c, tcp_highspeed.c, tcp_htcp.c, etc. Estos archivos de extensión .c se encuentran escritos en lenguaje de programación C, y son un conjunto de instrucciones que usan las dos principales variables: Ventana de congestión y "Slow-start sthreshold", bajo el protocolo de transporte al cual corresponde el archivo.

1.1.4. Variables de TCP en código fuente

Las variables de TCP se encuentran definidas en el archivo denominado:
<Directorio de código fuente>/include/linux/tcp.h, mostrado en la figura 25.

```
/*
 *  Slow start and congestion control (see also Nagle, and Karn & Partridge)
 */
    u32 snd_ssthresh;   /* Slow start size threshold        */
    u32 snd_cwnd;   /* Sending congestion window        */
    u32 snd_cwnd_cnt;   /* Linear increase counter      */
    u32 snd_cwnd_clamp; /* Do not allow snd_cwnd to grow above this */
    u32 snd_cwnd_used;
    u32 snd_cwnd_stamp;
    u32 prior_cwnd; /* Congestion window at start of Recovery. */
    u32 prr_delivered;  /* Number of newly delivered packets to
                 * receiver in Recovery. */
    u32 prr_out;    /* Total number of pkts sent during Recovery. */

    u32 rcv_wnd;    /* Current receiver window      */
    u32 write_seq;  /* Tail(+1) of data held in tcp send buffer */
    u32 notsent_lowat;  /* TCP_NOTSENT_LOWAT */
    u32 pushed_seq; /* Last pushed seq, required to talk to windows */
    u32 lost_out;   /* Lost packets         */
    u32 sacked_out; /* SACK'd packets           */
    u32 fackets_out;    /* FACK'd packets           */
    u32 tso_deferred;
```

Figura 25. Variables definidas en <Directorio de código fuente>/include/linux/tcp.h (Versión de código fuente: Linux 3.13.0-32)

Las variables, cuyo comportamiento será modificado a partir de este momento son:

a) **snd_cwnd (**Ventana de congestión a enviar).- define el valor de la ventana de congestión enviada durante la transmisión, el cual varía de acuerdo al algoritmo y protocolo empleado.

b) **snd_ssthresh** (Slow-start threshold).- Define el valor máximo que puede alcanzar la ventana de congestión durante la fase de slow-start, y éste variará cada vez que sea producida una congestión, reduciendo su valor, a una fracción de snd_cwnd cuando se produjo la pérdida, la cual depende del protocolo de transporte aplicado. Para el caso de TCP Reno y Tahoe: snd_ssthresh es reducido a la mitad de snd_cwnd, cuando se produce la pérdida.

c) **snd_cwnd_cnt** (Contador de incremento lineal).- Cuenta los segmentos que deben ser enviados y reconocidos hasta el momento en que la ventana debe aumentar su tamaño durante la etapa de incremento aditivo. Si la ventana de congestión tiene un valor "N", **snd_cwnd_cnt** cuenta que se hayan enviado y reconocido los "N" segmentos, para incrementar el valor de **snd_cwnd**, y luego colocar el valor de **snd_cwnd_cnt** nuevamente a 0 (cero), y realizar un nuevo conteo para la siguiente ventana de congestión.

d) **snd_cwnd_clamp** (Límite de ventana).- Es el valor máximo que puede alcanzar la ventana de congestión durante la transmisión. Evita que la ventana de congestión tenga un crecimiento infinito de no existir una congestión.

snd_cwnd, snd_snd_sstrhesh y snd_cwnd_cnt se encuentran establecidos en cantidad de segmentos (paquetes).

1.1.5. Estructuras en TCP

El uso de estructuras es esencial en la programación de TCP y sus variantes. Las estructuras definen una acción específica como podría ser inicialización de datos, evitar la congestión, modificar el "ssthresh" después de la congestión, entre otros.

La figura 26 muestra las estructuras que usualmente son añadidas en el control de congestión de TCP o cualquiera de sus variantes, las cuales son establecidas en la función **tcp_congestion_ops**, en el archivo **tcp.h**, ubicado en el fichero <Directorio de código fuente>/include/net.

Las estructuras definidas en tcp_congestion_ops son empleadas en cada uno de los archivos de extensión .c que determina la variante de TCP que controlará la transmisión. Por ejemplo la estructura **void (*init) (struct sock *sk)** en el archivo de configuración tcp_highspeed.c (Ubicado en <Directorio de código fuente>/net/ipv4/) es invocada en la línea: **"static void hstcp_init(struct sock *sk)"** de este archivo.

```
struct tcp_congestion_ops {
    struct list_head    list;
    unsigned long flags;

    /* initialize private data (optional) */
    void (*init)(struct sock *sk);
    /* cleanup private data  (optional) */
    void (*release)(struct sock *sk);

    /* return slow start threshold (required) */
    u32 (*ssthresh)(struct sock *sk);
    /* lower bound for congestion window (optional) */
    u32 (*min_cwnd)(const struct sock *sk);
    /* do new cwnd calculation (required) */
    void (*cong_avoid)(struct sock *sk, u32 ack, u32 acked, u32 in_flight);
    /* call before changing ca_state (optional) */
    void (*set_state)(struct sock *sk, u8 new_state);
    /* call when cwnd event occurs (optional) */
    void (*cwnd_event)(struct sock *sk, enum tcp_ca_event ev);
    /* new value of cwnd after loss (optional) */
    u32  (*undo_cwnd)(struct sock *sk);
    /* hook for packet ack accounting (optional) */
    void (*pkts_acked)(struct sock *sk, u32 num_acked, s32 rtt_us);
    /* get info for inet_diag (optional) */
    void (*get_info)(struct sock *sk, u32 ext, struct sk_buff *skb);

    char        name[TCP_CA_NAME_MAX];
    struct module   *owner;
};
```

Figura 26. Estructuras de TCP definidas en <Directorio de código fuente>/include/net/tcp.h (Versión de código fuente: Linux 3.13.0-32)

En tcp_congestion_ops se define:

a) **void (*init) (struct sock *sk):** Permite inicializar los datos, es decir indican qué valor adoptará cada variable al inicio de la transmisión.

b) **u32 (*ssthresh)(struct sock *sk):** Permite retornar un nuevo valor de snd_ssthresh, cuando ocurre una congestión.

c) **void (*cong_avoid)(struct sock *sk, u32 ack, u32 acked, u32 in_flight):** Define el proceso de evitar la congestión, algoritmo de "slow-start" e incremento de ventana.

d) **u32 (*min_cwnd)(const struct sock *sk):** Define el valor mínimo que puede adoptar la ventana de congestión durante la transmisión. No se debe confundir este valor con el concepto de "Ventana inicial". La ventana mínima puede ser diferente del valor de la ventana inicial, sin embargo, no debería ser mayor a esta última.

e) **void (*cwnd_event)(struct sock *sk, enum tcp_ca_event ev)** invoca cuando un evento de la ventana de congestión (cwnd) ocurre, pudiéndose dar cualquiera de los eventos que se indican a continuación:

CA_EVENT_TX_START : Primera transmisión cuando ningún paquete está en vuelo.
CA_EVENT_CWND_RESTART: Reinicio de la ventana de congestión.
CA_EVENT_COMPLETE_CWR: Fin de la recuperación de la congestión.
CA_EVENT_LOSS : Tiempo de espera expirado en pérdida.
CA_EVENT_FAST_ACK : ACK en secuencia.
CA_EVENT_SLOW_ACK : Otro ACK

La figura 27 muestra la ubicación de los eventos de cwnd en el código fuente de TCP.

```
/* Events passed to congestion control interface */
enum tcp_ca_event {
    CA_EVENT_TX_START,  /* first transmit when no packets in flight */
    CA_EVENT_CWND_RESTART,  /* congestion window restart */
    CA_EVENT_COMPLETE_CWR,  /* end of congestion recovery */
    CA_EVENT_LOSS,      /* loss timeout */
    CA_EVENT_FAST_ACK,  /* in sequence ack */
    CA_EVENT_SLOW_ACK,  /* other ack */
};
```

Figura 27. Estados de cwnd en TCP definidas en …/include/net/tcp.h

(Versión de código fuente: Linux 3.13.0-32)

1.2. Algoritmo de TCP Reno.

Se describió TCP Reno en el capítulo 2. El algoritmo de TCP Reno se representa en el diagrama de flujo, el cual se observa en la figura 28.

1.3. Programación de TCP Reno en Linux.

Comúnmente TCP Reno se encuentra implementado en las distribuciones de Linux, y éste es seleccionado como la variante de TCP a ser utilizada durante la transmisión mediante la línea de comando: **"sysctl –w net.ipv4.tcp_congestion_control = reno".** Sin embargo, este documento muestra como emplear las variables para la construcción de TCP Reno en Linux, en caso éste no

se encuentre implementado en el sistema operativo. El sistema operativo por defecto implementa "retransmisión rápida" (Fast-retransmit) y "recuperación rápida" (Fast-recovery).

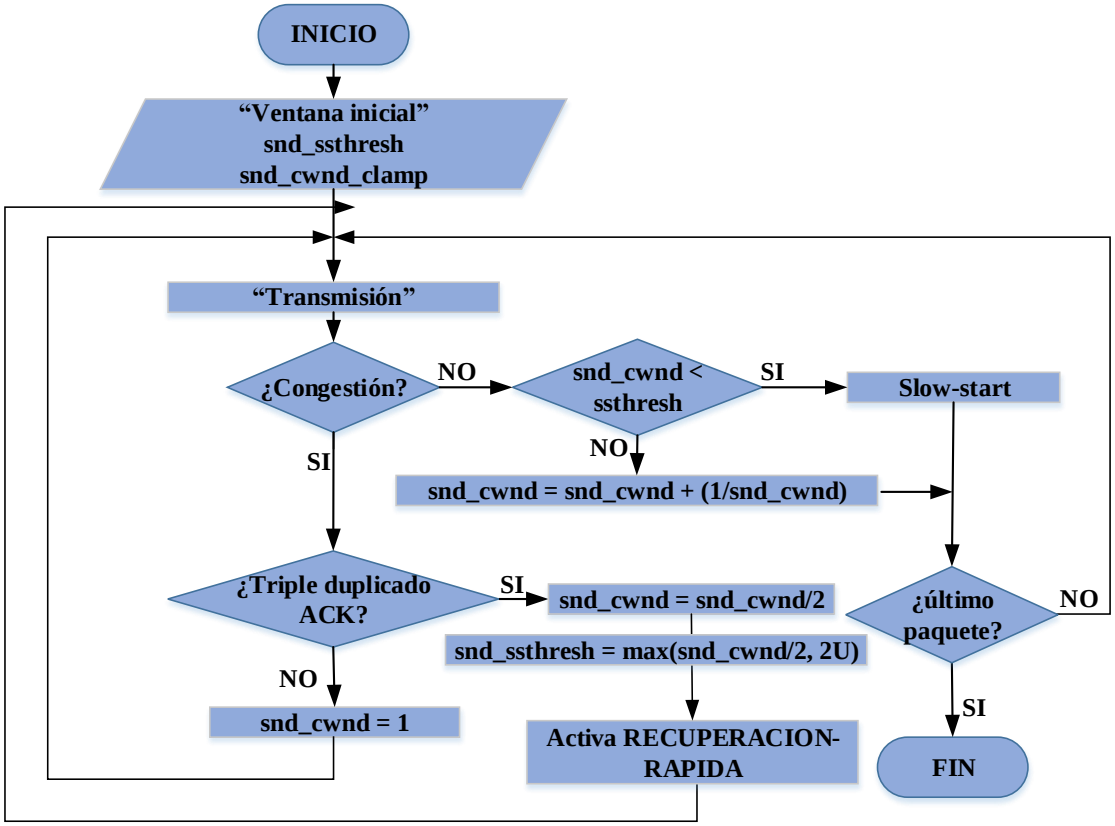


Figura 28. Algoritmo de TCP Reno.

Se debe tener en cuenta que para la programación de TCP Reno se debe inicialmente invocar las librerías mediante "include", de las siguientes líneas:

```
#include <linux/module.h>
#include <net/tcp.h>
```

A continuación se muestra como se programa TCP Reno.

a) Establecimiento del valor inicial de snd_cwnd y snd_cwnd_clamp

Se define snd_cwnd con un valor inicial igual a 1 segmento y snd_cwnd_clamp se establece como un valor muy grande. Las líneas de comando son:

```
static void tcp_reno2_init(struct sock *sk)
{
 struct tcp_sock *tp = tcp_sk(sk);
 tp->snd_cwnd = 1;
 tp->snd_cwnd_clamp = min_t(u32, tp->snd_cwnd_clamp, 0xffffffff/128);
}
```

b) Valor de snd_ssthresh posterior a la congestión.

Como se mencionó antes, éste debe ser equivalente a la mitad de la última ventana de cuando se produjo la congestión. Se hace la validación para que este valor no sea inferior a 2 segmentos. Se describe en las siguientes líneas de instrucciones:

```
static u32 tcp_reno2_ssthresh(struct sock *sk)
{
 const struct tcp_sock *tp = tcp_sk(sk);
 return max(tp->snd_cwnd - (tp->snd_cwnd >> 1), 2U);  // Se divide entre 2
}
```

c) **Evitación de la congestión**: Comprende:

- Se permite la transmisión sólo si existen paquetes por transmitir.
- Se realiza el llamado a la función de "slow-start" cuando snd_cwnd <= snd_sstresh. Esta función se encuentra definida en el código fuente de TCP por defecto.
- Se realiza el incremento aditivo empleando el contador snd_cwnd_cnt.

Las líneas de instrucciones para la evitación de la congestión son:

```
static void tcp_reno2_cong_avoid(struct sock *sk, u32 ack, u32 acked, u32
in_flight)
{
 struct tcp_sock *tp = tcp_sk(sk);
 if (!tcp_is_cwnd_limited(sk, in_flight))
        return;
// Ejecuta slow-start si: snd_cwnd <= snd_ssthresh
 if (tp->snd_cwnd <= tp->snd_ssthresh)
        tcp_slow_start(tp, acked);
 else {
        /* Realiza incremento aditivo */
        if (tp->snd_cwnd < tp->snd_cwnd_clamp) {
               /* Contador snd_cwnd_cnt incrementa en 1 */
               tp->snd_cwnd_cnt ++;
               if (tp->snd_cwnd_cnt >= tp->snd_cwnd) {
                      /* Contador snd_cwnd_cnt retorna a 0 */
                      /* cuando alcanza el valor de snd_cwnd */
                      tp->snd_cwnd_cnt =0;
                      tp->snd_cwnd++;
 }      }      }
}
```

El nombre con el cual se invoca a cada función puede variar, sin embargo cada uno de ellas deben definirse en las líneas de instrucciones posteriores, además se puede modificar el tamaño mínimo de ventana (Ver programa completo en Anexo H).

1.4. Análisis de una conexión que emplea TCP Reno.

El análisis con MATLAB de una conexión que emplea TCP RENO toma en cuenta las siguientes métricas: Retardo (RTT), Proporción de pérdidas y rendimiento (ancho de banda), y se realiza empleando el analizador de datos de una conexión, explicado en la sección 2.3.1 del capítulo 4.

A continuación se explica el análisis gráfico de una conexión TCP RENO con MSS = 1448 Bytes.

1.4.1. Transmisión con RTT=0.05s y proporción de pérdidas=0.01 durante 60 segundos.

1.4.1.1. Suposiciones a contrastar.

Suposición A: La transmisión inicia con el algoritmo de slow-start.
Suposición B: El incremento aditivo a lo largo de la transmisión es 1 y el decremento real multiplicativo se aproxima a 0.5.
Suposición C: La ventana de congestión es reducida a la mitad o a 1, cuando ocurre una congestión por triple ACK duplicado, o por tiempo de expiración agotado ("timeout").
Suposición D: El rendimiento experimental es menor al rendimiento máximo teórico.

1.4.1.2. Análisis gráfico

El análisis gráfico con MATLAB mostrado en la **figura K1, del ANEXO K**, muestra lo siguiente:

- La ventana de congestión tiene un comportamiento variable de acuerdo al algoritmo de "slow-start", incremento aditivo y decremento multiplicativo.
- El valor del incremento aditivo (α) es 1.
- El valor de decremento real se ubica entre 0.5 y 0.6. Dada la aproximación de la ventana de congestión a un valor entero cuando ocurre una congestión, difiere ligeramente del valor de β=0.5. Por ejemplo: Si la ventana de congestión es 11 cuando la pérdida por "triple ACK duplicado" es detectada, al multiplicar por el decremento es $11 \times 0.5 = 5.5$, pero siendo la ventana de congestión de recuperación un valor entero, ésta es 6, por tanto el decremento real es $6/11 = 0.5455$.

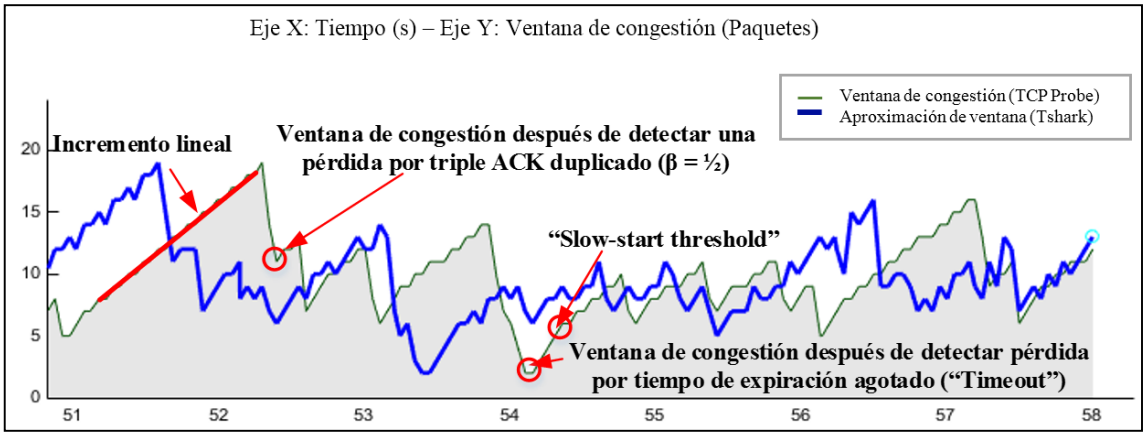


Figura 29. Ventana de congestión en el tiempo empleando TCP Reno (Gráfica con Matlab).

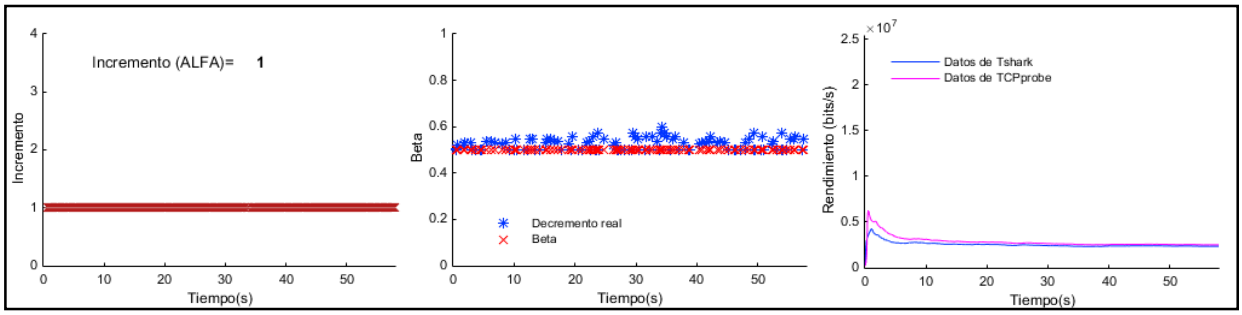


Figura 30. Incremento aditivo, Decremento multiplicativo y Rendimiento empleando TCP Reno (Gráfica con Matlab)

- El rendimiento inicial es más alto, debido al algoritmo de "slow-start" presente cuando inicia la transmisión.

1.4.1.3. Comparación con el rendimiento teórico

Herramienta/método	Rendimiento experimental (Mb/s)	Rendimiento teórico (Mb/s)	Error (%)
IPERF	2.430	3.035	19.93%
TCP PROBE	2.513	3.035	17.20%
TSHARK	2.354	3.035	22.44%

Tabla 6. Comparación de datos en conexión única que emplea TCP Reno.

El rendimiento obtenido experimentalmente ofrecido por IPERF, se compara con el valor teórico del rendimiento. También se muestra el error del rendimiento experimental con respecto al rendimiento teórico.

1.4.1.4. Conclusión.

Las suposiciones A, B, C y D son aceptadas.

1.5. Evaluación del comportamiento de TCP Reno empleando conjuntos de muestras.

1.5.1. Parámetros de evaluación de TCP Reno en una conexión cliente-servidor

Se considera:

- Total de muestras = 4 y cantidad de elementos (transmisiones) por muestra = 10.
- Parámetros fijos por cada muestra: RTT y proporción de pérdidas.
- Duración de cada transmisión = 60 segundos.
- La ventana de congestión inicial = 1 segmento.
- El parámetro a calcular es el rendimiento (Ancho de banda).

TCP Reno es configurado en la estación transmisora, y la estación receptora posee el campo "Ventana" configurado con un valor muy grande de modo que el tamaño de la ventana de congestión sea determinado por el transmisor TCP.

Muestra	RTT	Proporción de pérdidas	Rendimiento teórico (Mb/s)
1	50 ms.	1%	3.035
2	50 ms.	2%	2.146
3	100 ms.	1%	1.518
4	100 ms.	2%	1.073

Tabla 7. Parámetros fijos para la evaluación de TCP Reno y Rendimiento teórico.

El rendimiento se calcula teóricamente empleando la ecuación 9 del capítulo 1.

1.5.2. Suposiciones a contrastar.

Suposición A: Si aumentan las pérdidas para un mismo RTT, el rendimiento disminuye.

Suposición B: Para dos conexiones con el mismo porcentaje de pérdidas, el rendimiento disminuye cuando aumenta el RTT.

1.5.3. Valores obtenidos.

1.5.3.1. Valores obtenidos por IPERF

Los valores obtenidos por IPERF, para cada transmisión de cada una de las muestras de la tabla 7, para una conexión punto a punto, se observan en la tabla 8, en la cual se aprecia una desviación estándar baja que indica una tendencia de aproximación del rendimiento obtenido en cada muestra al promedio para cada prueba. En la figura 31 se representa gráficamente los datos obtenidos en la tabla 8.

N° de transmisión	Rendimiento (Mb/s)			
	Muestra 1	Muestra 2	Muestra 3	Muestra 4
1	2.430	1.860	1.260	1.010
2	2.290	1.740	1.330	0.874
3	2.540	1.740	1.330	0.874
4	2.530	1.770	1.250	0.861
5	2.340	1.760	1.360	0.958
6	2.570	1.730	1.290	0.919
7	2.360	1.810	1.370	0.997
8	2.800	1.800	1.280	0.884
9	2.360	1.710	1.220	0.919
10	2.460	1.780	1.420	0.964
Promedio	**2.470**	**1.770**	**1.310**	**0.926**
Desviación estándar	0.150	0.040	0.060	0.054

Tabla 8. Rendimiento obtenido por IPERF empleando TCP Reno.

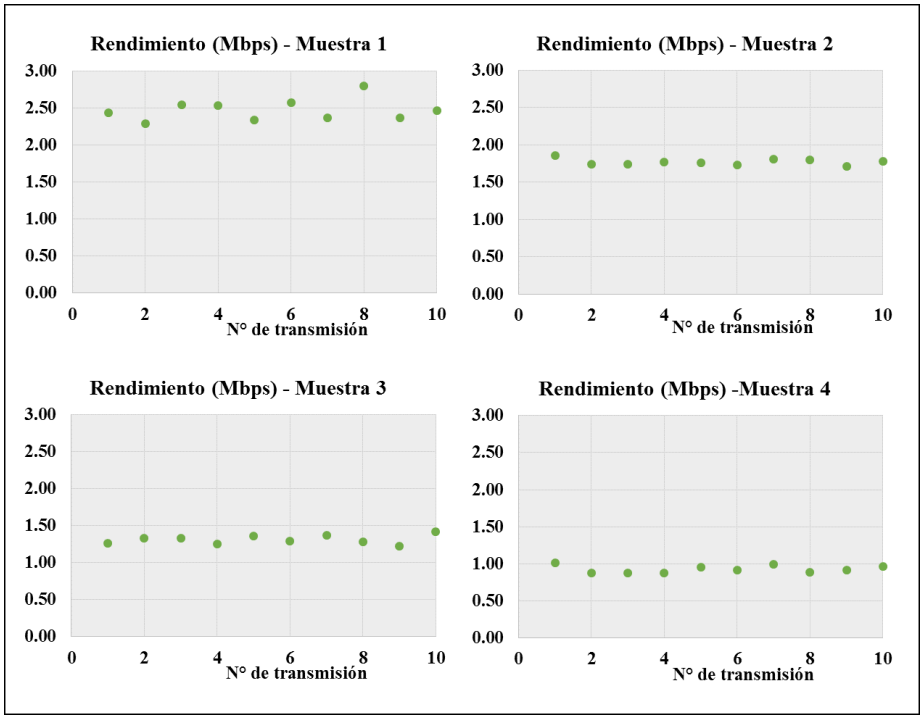


Figura 31. Diagramas del rendimiento obtenido por IPERF empleando TCP Reno.

1.5.3.2. Rendimiento obtenido empleando datos de captura de TCP Probe y Tshark.

Mediante TCPProbe se obtiene el rendimiento máximo mediante la ecuación 5, considerando MSS = 11584 bits, los valores de RTT y proporción de pérdidas.

De los datos capturados por Tshark es posible realizar el cálculo aproximado de los paquetes transmitidos y usarlo junto al tiempo de duración de la transmisión, para calcular el rendimiento, empleando la ecuación 4 del capítulo 1.

Los datos generados empleando Matlab para el cálculo, se muestran en la tabla 9.

N° de transmisión	Rendimiento (Mb/s)							
	Muestra 1		Muestra 2		Muestra 3		Muestra 4	
	TCPProbe	Tshark	TCPProbe	Tshark	TCPProbe	Tshark	TCPProbe	Tshark
1	2.513	2.354	1.920	1.776	1.230	1.190	0.983	0.952
2	2.338	2.226	1.793	1.677	1.342	1.243	0.844	0.807
3	2.644	2.458	1.971	1.830	1.291	1.238	1.010	0.956
4	2.609	2.473	1.820	1.684	1.312	1.230	0.859	0.818
5	2.410	2.304	1.809	1.692	1.339	1.296	0.939	0.904
6	2.638	2.433	1.786	1.701	1.275	1.236	0.916	0.871
7	2.395	2.222	1.880	1.777	1.371	1.293	0.964	0.929
8	2.836	2.376	1.845	1.744	1.246	1.197	0.903	0.857
9	2.412	2.272	1.758	1.655	1.197	1.147	0.900	0.855
10	2.500	2.409	1.864	1.729	1.409	1.345	0.974	0.914
Promedio	**2.530**	**2.353**	**1.844**	**1.726**	**1.301**	**1.241**	**0.929**	**0.886**
Desviación estándar	0.152	0.093	0.066	0.055	0.066	0.058	0.054	0.053

Tabla 9. Rendimiento obtenido por TCP Probe y Tshark al emplear TCP Reno.

1.5.3.3. Ventana de congestión promedio calculada de los datos de TCP Probe

Los valores de congestión promedio obtenidos en cada muestra en la tabla 10.

N° de transmisión	Ventana de congestión promedio(Paquetes)			
	Muestra 1	Muestra 2	Muestra 3	Muestra 4
1	10.847	8.288	10.618	8.483
2	10.091	7.737	11.581	7.287
3	11.411	8.509	11.141	8.719
4	11.261	7.854	11.326	7.411
5	10.404	7.809	11.563	8.104
6	11.386	7.707	11.003	7.908
7	10.339	8.114	11.839	8.325
8	12.241	7.963	10.753	7.797
9	10.409	7.586	10.330	7.767
10	10.790	8.045	12.159	8.407
Promedio	**10.918**	**7.961**	**11.231**	**8.021**
Desviación estándar	0.658	0.284	0.572	0.470

Tabla 10. Ventana de congestión promedio al emplear TCP Reno (Unidades: Paquetes).

La desviación estándar en las tablas 9 y 10 es baja, lo cual indica que los valores obtenidos para cada prueba se aproximan entre sí.. La representación gráfica de los valores obtenidos en las tablas 9 y 10, se puede observar en la sección 2 del Anexo K.

1.5.4. Test de hipótesis y conclusiones.

En la sección 1.5.2 son planteadas las suposiciones A y B. Para contrastar las suposiciones mencionadas, se considera los valores promedios de rendimiento (expresado en Mb/s) obtenidos haciendo uso de las herramientas iPerf, Tcpprobe y Tshark (Vistos en tablas 8 y 9).

Los valores ordenados para contrastar las suposiciones A y B se observan en las tablas 11 y 12.

RTT (s.)	Herramienta de medición	Rendimiento promedio (Mb/s)		Suposición a contrastar
		Menor pérdida: 1%	Mayor pérdida: 2%	
0.05	iPerf	2.470	1.770	A mayor pérdida, el rendimiento disminuye.
	Tcpprobe	2.530	1.844	
	Tshark	2.353	1.726	
0.10	iPerf	1.310	0.926	
	Tcpprobe	1.301	0.929	
	Tshark	1.241	0.886	

Tabla 11. Valores para contrastar la suposición A.

Pérdidas	Herramienta de medición	Rendimiento promedio (Mb/s)		Suposición a contrastar
		Menor RTT: 0.05 s	Mayor RTT: 0.10 s	
1%	iPerf	2.470	1.310	A mayor RTT, el rendimiento disminuye.
	Tcpprobe	2.530	1.301	
	Tshark	2.353	1.241	
2%	iPerf	1.770	0.926	
	Tcpprobe	1.844	0.929	
	Tshark	1.726	0.886	

Tabla 12. Valores para contrastar la suposición B.

Al observar la tabla 11 es posible realizar una simple comparación arimética, y concluir que la **suposición A es aceptable**. De forma análoga al comparar ariméticamente los valores en la tabla 12, **se acepta la suposición B.** Para una mayor confiabilidad, se realizan tests de hipótesis para cada uno de los casos planteados. Se explica a continuación:

1.5.4.1. Test de hipótesis empleando R.

a) Test de proporciones (prop.test)

Dadas dos muestras “x” e “y”, cada una con “n” elementos, el test de proporciones compara si existe igualdad entre las proporciones x/y. El test compara lo siguiente

Hipótesis nula: $x_1/y_1 = x_2/y_2 = \ldots = x_n/y_n$

Hipótesis alternativa: $x_1/y_1 \neq x_2/y_2 \neq \ldots \neq x_n/y_n$

Un nivel de confianza es dado para cada test, qué es el grado de precisión del test. Éste es por defecto 0.95. El nivel de significancia es el complemento del nivel de confianza. Se considera aquí: Nivel de significancia = 0.05.

Para la realización del test cada valor de "x" debe ser menor o igual a cada valor de "y", caso contrario el test mostrará un mensaje indicando el error mencionado.

Para interpretar el test de hipótesis se emplea:

- **Proporciones:** Son las proporciones, que indican que "x" es menor a "y". Cada proporción mostrada es menor o igual a 1 ("y" es mayor o igual a "x").
- **"p"**: Si "p" es mayor al nivel de significancia, se rechaza la hipótesis alternativa. Caso contrario se acepta la hipótesis alternativa.

La sintaxis del test de proporciones en R es:

```
prop.test(x, y, conf.level = 0.95, correct = FALSE)
```

Se realizan varios tests para los valores de rendimiento obtenidos a partir de los datos de cada una de las herramientas de captura (IPERF, TCP PROBE y TSHARK). Los tests a ejecutar son:

- Test 1.- Cuando RTT = 0.05s:

  x = [Rendimiento con tasa de pérdidas=2%], y = [Rendimiento con tasa de pérdidas=1%]
- Test 2.- Cuando RTT = 0.10s:

  x = [Rendimiento con tasa de pérdidas=2%], y = [Rendimiento con tasa de pérdidas=1%]
- Test 3.- Cuando la tasa de pérdidas es de 1%:

  x = [Rendimiento con RTT=0.10s.], y = [Rendimiento con RTT=0.05s.]
- Test 4.- Cuando la tasa de pérdidas es de 2%:

  x = [Rendimiento con RTT=0.10s.], y = [Rendimiento con RTT=0.05s.]

Los tests 1 y 2 consideran que los valores de "x" e "y" corresponden a valores de la 4ª y 3ª columna de la tabla 11 respectivamente.

Los tests 3 y 4 consideran que los valores de "x" e "y" corresponden a valores de la 4ª y 3ª columna de la tabla 12 respectivamente.

	Valor de "p"	Proporciones	Conclusión
Test 1	0.9991	0.7165992; 0.7288538; 0.7335317	$x < y$; $x_1/y_1=x_2/y_2=x_3/y_3$
Test 2	0.9999	0.7068702; 0.7140661; 0.7139404	$x < y$; $x_1/y_1=x_2/y_2=x_3/y_3$
Test 3	0.9993	0.5303644; 0.5142292; 0.5274118	$x < y$; $x_1/y_1=x_2/y_2=x_3/y_3$
Test 4	0.9993	0.5231638; 0.5037961; 0.5133256	$x < y$; $x_1/y_1=x_2/y_2=x_3/y_3$

Tabla 13. Resultados del test de proporciones.

Dados los resultados de los tests 1 y 2 en la tabla 14, se expone:

- Se acepta la hipótesis de que las proporciones sean iguales entre sí: La relación [Rendimiento con tasa de pérdidas de 2%]/ [Rendimiento con tasa de pérdidas de 1%] empleando diferentes herramientas (iPerf, TCPPROBE y TSHARK) es la misma.

- Se reafirma que el rendimiento cuando se aumenta la tasa de pérdidas de 1% a 2%, disminuye, no importa cuál de las herramientas para el cálculo mostradas se emplee.

Dados los resultados de los tests 3 y 4 en la tabla 14, se expone:

- Se acepta la hipótesis de que las proporciones sean iguales entre sí: La relación [Rendimiento con RTT=0.10s]/[Rendimiento con RTT=0.05s] empleando diferentes herramientas (iPerf, TCPPROBE y TSHARK) es la misma.
- Se reafirma que el rendimiento cuando se aumenta el valor de RTT de 0.05s a 0.10s disminuye, no importa cuál de las herramientas para el cálculo mostradas se emplee.

En base a la ecuación 9 (Modelo de Mathis), es posible realizar tests de proporciones basados en:

- Cuando se compara rendimientos obtenidos en transmisiones bajo una misma tasa de pérdidas y MSS, se valida la suposición A, la cual puede ser expresada como: $\frac{T_2}{T_1} = \frac{\sqrt{p_1}}{\sqrt{p_2}}$, donde T_1 y T_2 son los valores de rendimiento para tasas de pérdidas p_1 y p_2 respectivamente.
- Cuando se compara rendimientos obtenidos en transmisiones bajo un mismo valor de RTT y MSS, se valida la suposición B, la cual puede ser expresada como: $\frac{T_2}{T_1} = \frac{RTT_1}{RTT_2}$, donde T_1 y T_2 son los valores de rendimiento para tasas de pérdidas RTT_1 y RTT_2 respectivamente.

Los test de proporciones de la forma $w_1/z_1 = w_2/z_2$ son los siguientes

- Test 5.- $z = (z_1, z_2) = (T_{1a}, \sqrt{p_2})$ $w = (w1, w2) = (T_{2a}, \sqrt{p_1})$

  T_{1a} : Rendimiento cuando RTT=0.05s. y la tasa de pérdidas es p_1 =0.01.

  T_{2a} : Rendimiento cuando RTT=0.05s. y la tasa de pérdidas es p_2 =0.02.

- Test 6.- $z = (z_1, z_2) = (T_{1b}, \sqrt{p_2})$ $w = (w_1, w_2) = (T_{2b}, \sqrt{p_1})$

  T_{1b} : Rendimiento cuando RTT=0.10s. y la tasa de pérdidas es p_1 =0.01.

  T_{2b} : Rendimiento cuando RTT=0.10s. y la tasa de pérdidas es p_2 =0.02.

- Test 7.- $z = (z_1, z_2) = (T_{1c}, \sqrt{RTT_2})$ $w = (w_1, w_2) = (T_{2c}, \sqrt{RTT_1})$

  T_{1c} : Rendimiento cuando la tasa de pérdidas es de 1% y el delay: RTT_1 = 0.05s.

  T_{2c} : Rendimiento cuando la tasa de pérdidas es de 1% y el delay: RTT_2 = 0.10s.

- Test 8.- $z = (z_1, z_2) = (T_{1d}, \sqrt{RTT_2})$ $w = (w_1, w_2) = (T_{2d}, \sqrt{RTT_1})$

  T_{1d} : Rendimiento cuando la tasa de pérdidas es de 2% y el delay: RTT_1 = 0.05s.

  T_{2d} : Rendimiento cuando la tasa de pérdidas es de 2% y el delay: RTT_2 = 0.10s.

La proporción w_1/z_1 es igual para los valores de rendimiento calculados a partir de datos de iPerf, tshark o tcpprobe. Por tanto, para los tests 5, 6, 7 y 8, sólo se emplean los valores de rendimiento obtenidos empleando iPerf, mostrados en las tablas 11 y 12.

	Valor de "p"	Proporciones	Conclusión
Test 5	0.9939	0.7165992; 0.7071068	$w_1/z_1 = w_2/z_2$
Test 6	0.9999	0.7068702; 0.7071068	$w_1/z_1 = w_2/z_2$
Test 7	0.9850	0.5303644; 0.5000000	$w_1/z_1 = w_2/z_2$
Test 8	0.9886	0.5231638; 0.5000000	$w_1/z_1 = w_2/z_2$

Tabla 14. Resultados del test de proporciones: $w_1/z_1 = w_2/z_2$

Dados los resultados de los tests 5 y 6 en la tabla 15, se afirma que el rendimiento y la tasa de pérdidas son inversamente proporcionales. Mientras los resultados de los test 7 y 8 afirman que el rendimiento y delay (RTT) son inversamente proporcionales.

b) Correlación

El coeficiente de correlación de dos muestras "x" e "y", denotado como "r" indica si las muestras están correlacionadas. **r (x, y)** es el coeficiente de correlación, y:

- Si r = 1, se afirma que y=k.x, donde "k" es un valor constante (Correlación lineal).
- Si r se aproxima a 1, existe una alta correlación entre "x" e "y".
- Si r se aproxima a -1, existe una correlación negativa, es decir si "x" crece, "y" decrece.
- Si r es igual o se aproxima a 0, no existe correlación entre "x" e "y".

Los coeficientes de correlación obtenidos empleando R, al usar los valores de rendimiento de los tests 1, 2, 3 y 4 listados líneas arriba, son:

	Test 1	Test 2	Test 3	Test 4.
Coeficiente de correlación	0.9461814	0.9833421	0.8960396	0.8960396

Tabla 15. Coeficiente de correlación

El coeficiente de correlación obtenido en todos los casos es cercano a 1. Entonces se dice:

"Para un mismo RTT, existe alta correlación entre el rendimiento con una tasa de pérdidas de 1% y rendimiento con una tasa de pérdidas de 2%".

"Para una misma tasa de pérdidas, existe alta correlación entre las mediciones de rendimiento con un valor RTT=0.05s y rendimiento con un valor RTT=0.10s."

Teniendo en cuenta los resultados del test de proporciones y los valores de coeficiente de correlación obtenidos: **SE ACEPTAN LAS SUPOSICIONES A Y B.**

Los resultados del test de hipótesis en R, se observan en el Anexo N.

2. GAIMD.

2.1. Algoritmo de GAIMD.

Para GAIMD se implementa una nueva variable, denominada como "**dec",** dado que el decremento multiplicativo β es igual a dec/8. Es decir dec=7 para β=7/8 y dec=4 para β=1/2.

El diagrama de flujo del algoritmo de GAIMD se muestra en la figura 32.

2.2. Programación de GAIMD en Linux.

La implementación de GAIMD en Linux inicia con la creación del archivo, denominado aquí como **tcp_gaimd.c** dentro de <Directorio de código fuente>/net/ipv4.

Las líneas de programación de tcp_gaimd.c inician con la invocación de las librerías **<linux/module.h>** y **<net/tcp.h>** mediante include (Ver Anexo I).

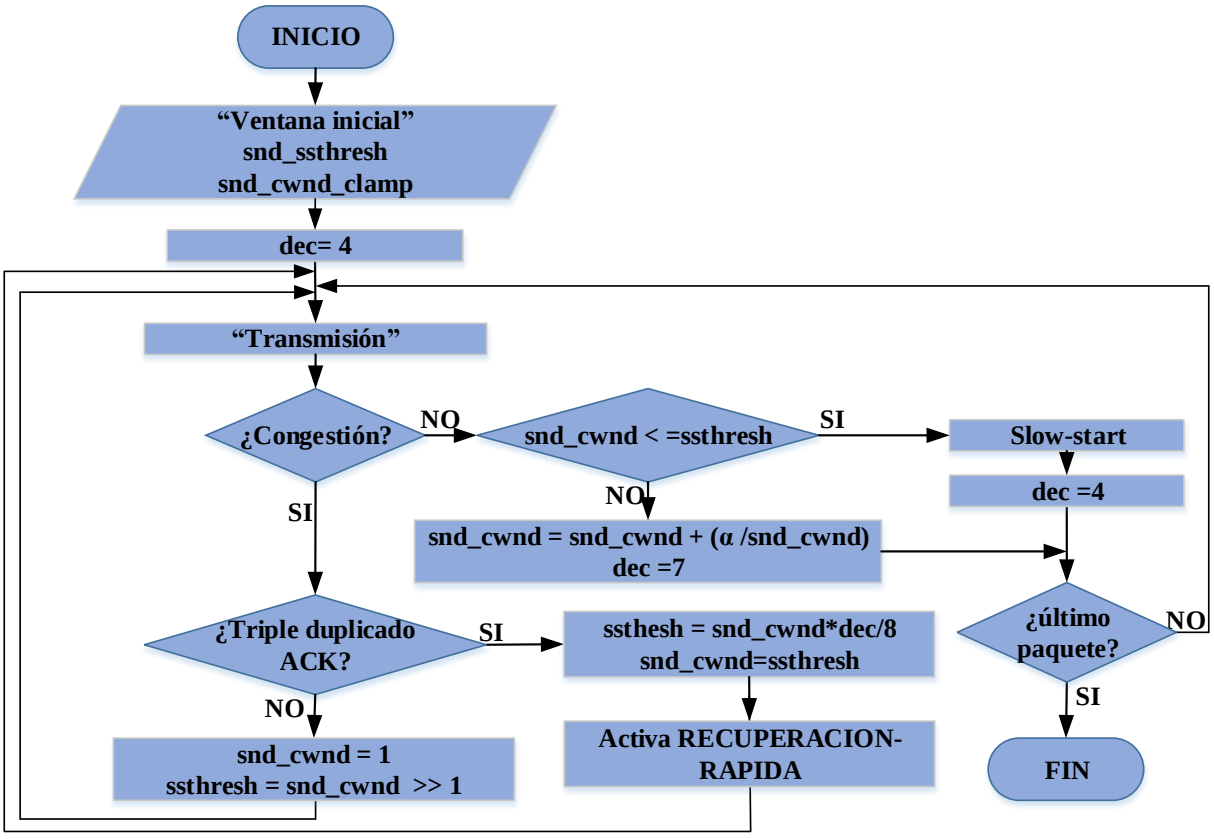


Figura 32. Diagrama de flujo de GAIMD

Las etapas del programa de GAIMD son:

a) Establecimiento de variables y valores iniciales

- Se define la nueva variable **dec.**
- Se establecen valores iniciales:
- Valor inicial de snd_cwnd es 1.
- Valor inicial de dec=4. Es decir en la fase de slow-start (fase inicial) $\beta=1/2$.
- snd_cwnd_clamp se establece como un valor muy grande.

Las líneas de programación se observan en el Anexo I.

b) Evitación de la congestión.

Cuando snd_cwnd <= snd_sstresh (Inicio de la transmisión) se llama a la función de "slow-start" y se considera un valor de dec=4, es decir si ocurre una congestión por triple ACK duplicado: $\beta=1/2$.

Cuando snd_cwnd>ssthresh se realiza el incremento aditivo, considerando:

- El incremento aditivo $\alpha=0.3125$ en el lenguaje de código fuente, programado como una suma directamente no es posible debido a que solo acepta valores enteros para snd_cwnd.
- Para el incremento de snd_cwnd, se toma en cuenta que se deben transmitir snd_cwnd_cnt = $100/(100\alpha)$ × snd_cwnd paquetes para incrementar snd_cwnd en 1. El valor 100α se aproxima al valor superior entero cercano $100\alpha=32$
- Se considera un valor dec=7, es decir $\beta=7/8$ si una pérdida por triple ACK duplicado ocurre durante la fase de incremento aditivo.

La programación de éstas instrucciones se observan en el Anexo I.

c) Valor de snd_cwnd y snd_ssthresh posterior a la congestión.

El valor de snd_ssthresh y snd_cwnd, posterior a la congestión es igual al valor de ventana en el momento de la congestión multiplicada por dec/8 cuando ocurre una pérdida por triple ACK duplicado.

Si ocurre una pérdida por tiempo de expiración agotado ("timeout"), snd_cwnd adquiere el valor mínimo de snd_cwnd (En este caso: snd_cwnd=1), y snd_ssthresh obtiene un valor menor igual a snd_cwnd, con el objetivo que snd_cwnd continúe realizando un incremento aditivo **(Ver programación en Anexo I).**

Siendo tcp_gaimd.c programado correctamente, se realiza el procedimiento explicado en el capítulo 4 para su compilación y selección como protocolo de transporte.

2.3. Análisis de una conexión que emplea GAIMD

El análisis con MATLAB de una conexión que emplea GAIMD se realiza empleando el analizador de datos de una conexión, explicado en el capítulo 4. En este análisis se emplea GAIMD con α=0.3125 y β=7/8, y un valor de MSS = 1448 bytes.

2.3.1. Transmisión con RTT=0.1s y Proporción de pérdidas=0.01 durante 60 segundos.

2.3.1.1. Suposiciónes a contrastar.

Suposición A: El incremento aditivo a lo largo de la transmisión es 0.3125 y el decremento real multiplicativo promedio se aproxima a 0.875.

Suposición B: El rendimiento experimental es menor al rendimiento máximo teórico.

2.3.1.2. Análisis gráfico

El análisis gráfico con MATLAB mostrado en la figura L3, del ANEXO L, muestra lo siguiente:

- La ventana de congestión tiene un comportamiento variable de acuerdo al algoritmo de "slow-start", incremento aditivo y decremento multiplicativo.
- El valor del incremento aditivo (α) se aproxima a 0.3125.
- El valor de decremento real se aproxima a 0.875.

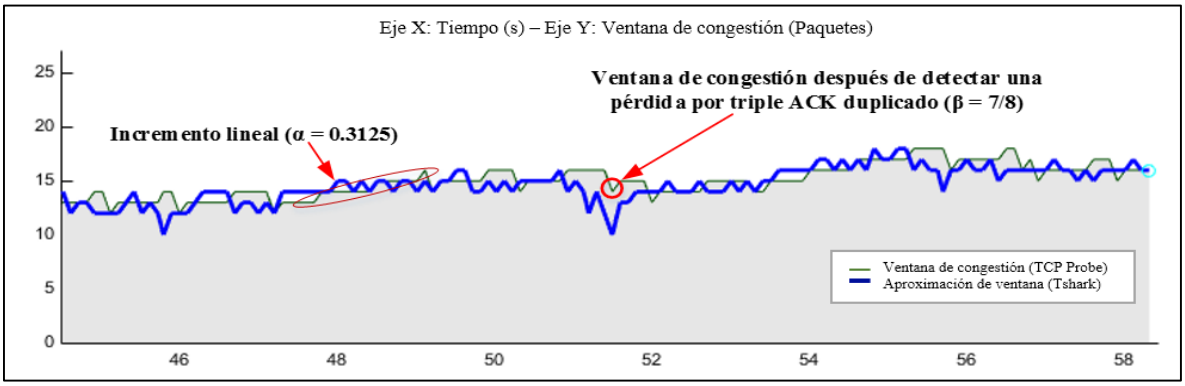


Figura 33. Ventana de congestión en el tiempo empleando GAIMD (Gráfica con Matlab).

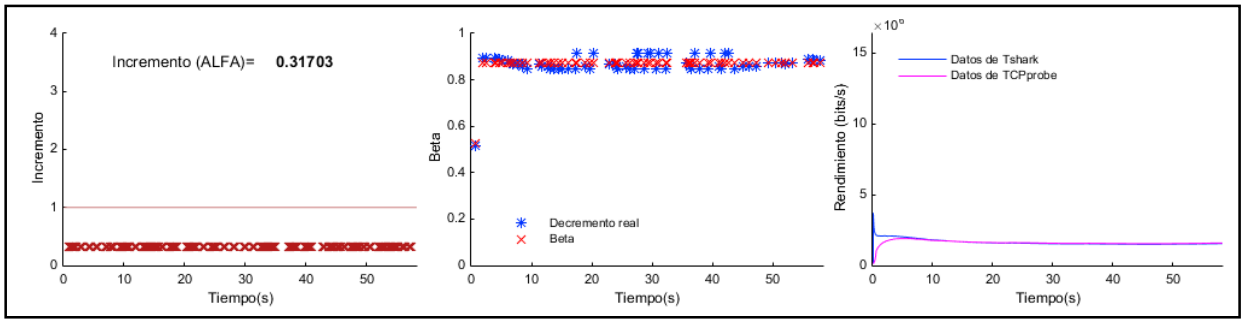


Figura 34. Incremento aditivo, Decremento multiplicativo y Rendimiento con GAIMD (Gráfica con Matlab)

- El rendimiento inicial es más alto, debido al algoritmo de "slow-start" presente cuando inicia la transmisión.

2.3.1.3. Comparación con el rendimiento teórico

Para el cálculo teórico del rendimiento, se emplea la ecuación 9, mostrada en el capítulo 1, considerando: MSS = 1448bytes = 11584 bits, C = 1.31 (Se consideran pérdidas aleatorias), RTT=0.10 y p =0.01. El valor de rendimiento obtenido experimentalmente, se compara con respecto al cálculo teórico del rendimiento en la tabla 16. También se muestra el error del rendimiento experimental con respecto al rendimiento teórico.

Herramienta/método	Rendimiento experimental (Mb/s)	Rendimiento teórico (Mb/s)	Error (%)
IPERF	1.620	1.795	9.75%
TCP PROBE	1.558	1.795	13.20%
TSHARK	1.587	1.795	11.59%

Tabla 16. Comparación de datos en conexión única que emplea GAIMD.

2.3.1.4. Conclusión.

Las suposiciones A y B son aceptadas.

2.4. Evaluación del comportamiento de GAIMD empleando conjuntos de muestras.

2.4.1. Parámetros de evaluación de GAIMD en una conexión cliente-servidor.

Se toman 4 pruebas (muestras), donde cada muestra consta de 10 transmisiones de 60 segundos cada una, para una combinación de parámetros de RTT y tasa de pérdidas establecidos (Ver tabla 17).

Muestra	RTT	Proporción de pérdidas	Rendimiento teórico (Mb/s)
1	50 ms.	1%	3.589
2	50 ms.	2%	2.538
3	100 ms.	1%	1.795
4	100 ms.	2%	1.269

Tabla 17. Parámetros para la evaluación de GAIMD y Rendimiento teórico.

El rendimiento se calcula teóricamente empleando la ecuación 18 del capítulo 2.

2.4.2. Suposiciones a constrasatar.

Suposición A: El rendimiento promedio es mayor cuando se emplea GAIMD en comparación a TCP RENO.

Suposición B: La ventana de congestión promedio es mayor cuando se emplea GAIMD en comparación a TCP RENO.

2.4.3. Valores obtenidos.

2.4.3.1. Valores obtenidos por IPERF

Los valores obtenidos para cada una de las pruebas de la tabla 17, para una conexión punto a punto, se observan en la tabla 18.

	Rendimiento (Mb/s)			
N° de transmisión	**Muestra 1**	**Muestra 2**	**Muestra 3**	**Muestra 4**
1	3.490	2.070	1.620	1.060
2	3.430	2.530	1.620	1.190
3	3.430	2.530	1.960	1.190
4	3.220	2.430	1.740	1.380
5	3.010	2.460	1.550	0.985
6	3.530	2.200	1.450	1.320
7	3.170	2.340	1.400	1.020
8	3.340	2.240	1.510	1.480
9	2.870	2.220	2.210	1.230
10	3.050	2.510	1.570	1.240
Promedio	**3.254**	**2.353**	**1.663**	**1.210**
Desviación estándar	0.226	0.163	0.249	0.158

Tabla 18. Rendimiento obtenido por IPERF empleando GAIMD.

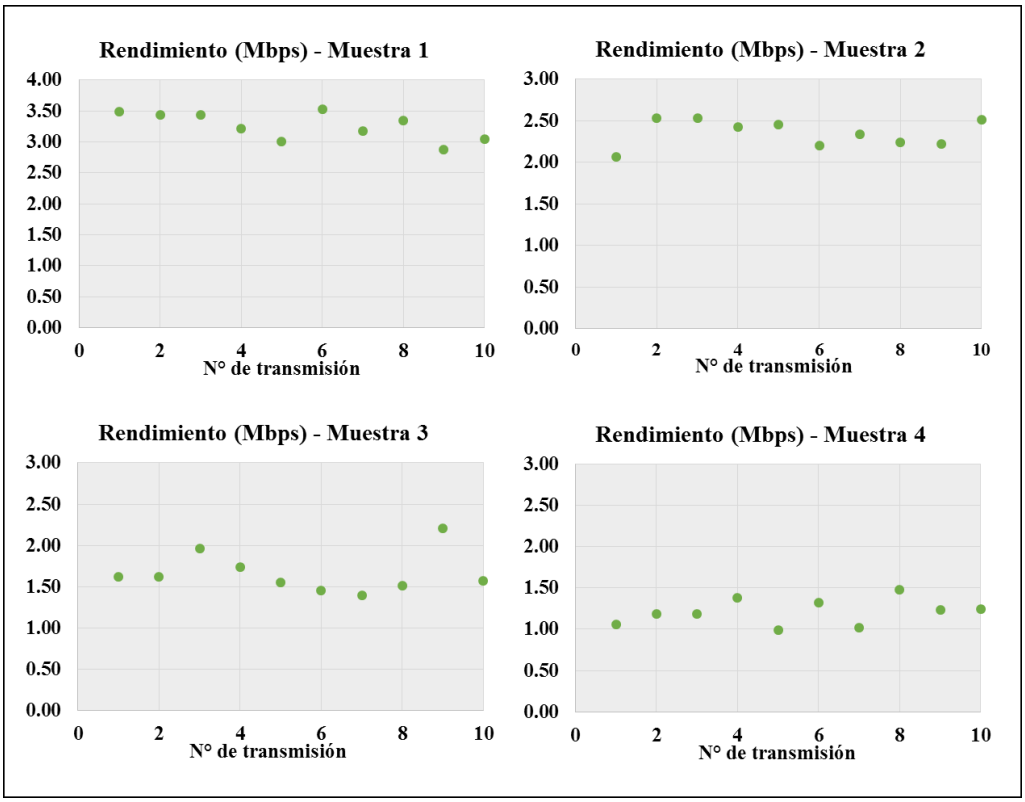


Figura 35. Diagramas del rendimiento obtenido por IPERF empleando GAIMD.

En la tabla 18, se observa una desviación estándar baja que indica una tendencia de aproximación del rendimiento obtenido en cada muestra al promedio para cada prueba.

2.4.3.2. Rendimiento obtenido empleando datos de captura de TCP Probe y Tshark.

Mediante TCPProbe se obtiene el rendimiento máximo mediante la ecuación 18, considerando MSS = 11584 bits. También se obtiene el rendimiento a partir de datos de Tshark empleando la ecuación 4. Los datos generados empleando Matlab para el cálculo, se muestran en la tabla 19.

	Rendimiento (Mb/s)							
	Muestra 1		Muestra 2		Muestra 3		Muestra 4	
N° de transmisión	TCPProbe	Tshark	TCPProbe	Tshark	TCPProbe	Tshark	TCPProbe	Tshark
1	3.565	3.352	2.176	2.100	1.587	1.558	1.055	0.999
2	3.500	3.309	2.590	2.481	1.762	1.722	1.165	1.132
3	3.480	3.278	2.365	2.207	1.955	1.900	1.114	1.081
4	3.281	3.094	2.483	2.374	1.755	1.721	1.391	1.351
5	3.095	2.970	2.519	2.353	1.530	1.493	0.957	0.938
6	3.601	3.400	2.260	2.152	1.447	1.409	1.301	1.262
7	3.238	3.142	2.383	2.293	1.362	1.344	1.003	0.972
8	3.392	3.083	2.301	2.204	1.492	1.457	1.470	1.416
9	2.928	2.853	2.301	2.211	2.208	2.153	1.255	1.203
10	3.068	2.961	2.584	2.476	1.832	1.728	1.290	1.197
Promedio	**3.315**	**3.144**	**2.396**	**2.285**	**1.693**	**1.649**	**1.200**	**1.155**
Desviación estándar	0.231	0.185	0.142	0.132	0.260	0.248	0.169	0.161

Tabla 19. Rendimiento obtenido por TCP Probe y Tshark al emplear GAIMD

La representación gráfica de los valores obtenidos en la tabla 19 se observa en la sección 2 del Anexo L.

2.4.3.3. Ventana de congestión promedio calculada de los datos de TCP Probe

	Ventana de congestión promedio(Paquetes)			
N° de transmisión	Muestra 1	Muestra 2	Muestra 3	Muestra 4
1	15.388	9.394	13.700	9.106
2	15.106	11.181	15.209	10.053
3	15.022	10.206	16.875	9.619
4	14.163	10.716	15.146	12.005
5	13.357	10.873	13.205	8.262
6	15.544	9.756	12.487	11.229
7	13.974	10.286	11.754	8.656
8	14.640	9.934	12.882	12.693
9	12.638	9.931	19.062	10.833
10	13.242	11.152	15.811	11.138
Promedio	**14.307**	**10.343**	**14.613**	**10.360**
Desviación estándar	0.995	0.613	2.248	1.463

Tabla 20. Ventana de congestión promedio al emplear GAIMD (Unidades: Paquetes).

Los valores de ventana de congestión de la tabla 20 se obtienen al realizar una transmisión de aproximadamente 1 minuto, y empleando los parámetros de cada una de las cuatro muestras.

La desviación estándar en las tablas 19 y 20 es baja, lo cual indica que los valores obtenidos para cada prueba se aproximan entre sí.

2.4.4. Comparación de GAIMD con TCP RENO

Los valores de rendimiento para el análisis son los siguientes:

Métrica	Rendimiento promedio (Mb/s)						Ventana de congestión	
Herramienta	iPerf		TCP Probe		Tshark		TCP Probe	
Muestra	RENO	GAIMD	RENO	GAIMD	RENO	GAIMD	RENO	GAIMD
1	2.470	3.254	2.530	3.315	2.353	3.144	10.918	14.307
2	1.770	2.353	1.844	2.396	1.726	2.285	7.961	10.343
3	1.310	1.663	1.301	1.693	1.301	1.649	11.231	14.613
4	0.926	1.210	0.929	1.200	0.886	1.155	8.021	10.360

Tabla 21. Rendimiento y ventana de congestión promedio al emplear TCP Reno y GAIMD.

2.4.4.1. Test de hipótesis empleando R.

a) Test de Student

El test de hipótesis consiste en contrastar una hipótesis donde se comparan dos muestras x, y.

El test de Student realiza un test de diferencias entre los valores de dos muestras "x" e "y", donde:

Hipótesis nula: $x - y = 0$ **Hipótesis alternativa:** $x - y \neq 0$

En cada test, se debe plantear un nivel de confianza, que es el grado de precisión del test. El nivel de confianza por defecto es 0.95, el cual es empleado en todos los casos a contrastar.

Además existe un nivel de significancia que es el complemento del nivel de confianza. En los casos actuales: Nivel de significancia = 0.05.

La interpretación del test de hipótesis considera:

- **Un valor "p"**: Si "p" es mayor al nivel de significancia, se rechaza la hipótesis alternativa. Caso contrario se acepta la hipótesis alternativa.
- **El intervalo de confianza [a; b]** ofrece confiabilidad en el test si el intervalo no incluye el valor "0". Es decir el test es confiable si a y b son valores positivos o negativos.
- Dado que la hipótesis alternativa sea aceptada:

  Si [a; b] es un conjunto de valores positivos, entonces: $x > y$.

  Si [a; b] es un conjunto de valores negativos, entonces $x < y$.

La sintaxis del test de Student en R a emplear es:

```
t.test(x, y, conf.level = 0.95, paired= TRUE)
```

Se realiza un test de Student para los valores de rendimiento obtenidos a partir de los datos de cada una de las herramientas de captura (IPERF, TCP PROBE y TSHARK), donde:

x = [Rendimiento con GAIMD]; y = [Rendimiento con TCP RENO].

De manera análoga se realiza el cuarto test para los valores de ventana de congestión, siendo

x = [Ventana de congestión con GAIMD]; y = [Ventana de congestión con TCP RENO]

Los testeos se realizan del siguiente modo con los datos de la tabla 21:

- Test 1.- x = Valores de 3ª columna; y = Valores de 2ª columna (Rendimiento).
- Test 2.- x = Valores de 5ª columna; y = Valores de 4ª columna (Rendimiento).
- Test 3.- x = Valores de 7ª columna; y = Valores de 6ª columna (Rendimiento).
- Test 4.- x = Valores de 9ª columna; y = Valores de 8ª columna (Ventana de congestión).

Los resultados de los test son:

	Valor de "p"	Intervalo de confianza	Conclusión
Test 1	0.021810	[0.1383710; 0.8636290]	x > y
Test 2	0.020470	[0.1465311; 0.8534689]	x > y
Test 3	0.024600	[0.1193055; 0.8641945]	x > y
Test 4	0.002323	[1.9309140; 3.8150860]	x > y

Tabla 22. Resultados del test de Student que compara GAIMD con TCP Reno.

Se concluye: ***"EL RENDIMIENTO Y VENTANA DE CONGESTIÓN ES MAYOR CUANDO SE EMPLEA GAIMD EN COMPARACIÓN A TCP RENO"***

b) Correlación

El coeficiente de correlación de dos muestras "x" e "y", denotado como "r" indica si las muestras están correlacionadas. Los coeficientes de correlación obtenidos empleando R, son:

	Test 1	Test 2	Test 3	Test 4.
Coeficiente de correlación	0.9993329	0.9999747	0.9993107	0.9997605

Tabla 23. Coeficiente de correlación entre muestras de TCP Reno y GAIMD.

El coeficiente de correlación obtenido en todos los casos es: r≈ 1. Entonces se dice:

"Existe linealidad entre el rendimiento de Tcp Reno y GAIMD"

"Existe linealidad entre la ventana de congestión de Tcp Reno y GAIMD"

Teniendo en cuenta los resultados del test de Student e intervalos de confianza de éste último, y los valores de coeficiente de correlación obtenidos:**SE ACEPTAN LAS SUPOSICIONES A Y B.**

Los resultados del test de hipótesis en R, se observan en el Anexo N.

2.5. Construcción de una relación entre TCP RENO y GAIMD.

La relación entre TCP RENO y GAIMD puede ser construida a partir de los valores de la tabla 22. La razón lineal es "k", siendo:

$$k = {}^{T_{RENO}}/_{T_{GAIMD}} = {}^{CWND_{RENO}}/_{CWND_{GAIMD}} \quad \textbf{(29)}$$

Donde T_{GAIMD} y T_{RENO} son los rendimientos empleando GAIMD y TCP RENO, y $CWND_{GAIMD}$ y $CWND_{RENO}$ son los valores de ventana de congestión promedio.

Para hallar la relación entre TCP RENO y GAIMD, y considerando que en la ecuación 21: $T_{GAIMD}/\,T_{RENO}$ es un valor constante, es conveniente realizar un **test de proporciones y/x**, esto es $y_1/x_1 = y_2/x_2 = y_3/x_3 = y_4/x_4$, donde "x" e "y" son conformados por los valores de cada una de las 4 muestras de la tabla 21.

- Test k_1.- x = Valores de 3ª columna; y = Valores de 2ª columna (Rendimiento).
- Test k_2.- x = Valores de 4ª columna; y = Valores de 4ª columna (Rendimiento).
- Test k_3.- x = Valores de 7ª columna; y = Valores de 6ª columna (Rendimiento).
- Test k_4.- x = Valores de 9ª columna; y = Valores de 8ª columna (Ventana de congestión).

Los resultados al realizar los tests k_1, k_2, k_3 y k_4 muestran valores de "p" de **0.9998, 1.0000, 0.9997** y **0.9999** respectivamente (Ver Anexo "N").

Los valores "k" (proporciones) aproximados a milésimas (de decimal) se muestran en la tabla 24.

	T_{RENO}/T_{GAIMD}			$CWND_{RENO}/CWND_{GAIMD}$	Promedio
Herramienta	**iPerf**	**TCP Probe**	**Tshark**	**TCP Probe**	
Muestra	**k**	**k**	**k**	**k**	
1	0.759	0.763	0.748	0.763	0.758
2	0.752	0.770	0.755	0.770	0.762
3	0.788	0.768	0.789	0.769	0.778
4	0.765	0.774	0.767	0.774	0.770
Promedio	**0.766**	**0.769**	**0.765**	**0.769**	**0.767**
Desviación estándar	0.015	0.005	0.018	0.005	0.011

Tabla 24. Relación entre TCP RENO y GAIMD.

El valor "k" promedio del conjunto de muestras es 0.767, es decir: $T_{RENO} = 0.767 \times T_{GAIMD}$.

Esta expresión permite ajustar la ecuación 21: $T_{GAIMD}/\,T_{RENO} \approx 1.24$, como:

$$T_{GAIMD}/\,T_{RENO} = 1.303 \pm \varepsilon \qquad \textbf{(30)}$$

Es conveniente realizar nuevos tests de proporciones que verifiquen la ecuación 30. Estos tests tienen la forma: $y_1/x_1 = y_2/x_2 = y_3/x_3 = y_4/x_4 = T_{RENO}/T_{GAIMD}$. Esto es: $y_1/x_1 = y_2/x_2 = y_3/x_3 = y_4/x_4 = 1/1.303$. Los valores para las primeras 4 proporciones son los empleadas en los tests k_1, k_2, k_3 y k_4.

Los resultados de los nuevos tests, llamados aquí como test k^*_1, test k^*_2, test k^*_3 y test k^*_4 respectivamente, muestran los valores de "p": 1, 1, 1 y 1 (Ver Anexo "K"). Por lo tanto, para las muestras obtenidas experimentalmente, la ecuación 30 es válida.

3. ESTP

3.1. ESTP en Linux

3.1.1. Variables TCP empleadas por ESTP.

Siendo ESTP una variante de TCP, utiliza las mismas variables TCP, y se basa en la construcción de un algoritmo de evitación de congestión. Las variables de importancia a emplear por ESTP son:

a) Ventana de congestión de envío: **snd_cwnd**

b) Slow-start ssthreshold: **snd_ssthresh**

c) Contador: **snd_cwnd_cnt**

d) Número de secuencia del primer paquete no reconocido: **snd_una**

e) Tamaño máximo del segmento: **mss_cache**

"snd_una" se convierte en una variable de importancia, debido a que la misma permitirá un correcto conteo de los paquetes transmitidos satisfactoriamente. El contador snd_cwnd_cnt sólo realiza el conteo de paquetes transmitidos durante la fase de incremento aditivo.

3.1.2. Paquetes transmitidos satisfactoriamente.

Para hallar la cantidad de paquetes transmitidos satisfactoriamente entre dos pérdidas, es decir aquellos que han sido reconocidos por el receptor, se emplean los siguientes parámetros:

seq_i: El menor número de secuencia de paquete no reconocido que es enviado.

seq_f: El número de secuencia del paquete perdido.

mss_size: Valor de mss_cache

La cantidad de paquetes transmitidos satisfactoriamente entre dos pérdidas, cuando ocurre una pérdida, es definido por la variable count, que equivale numéricamente a: (seq_i – seq_f)/mss_size seq_i y seq_f son hallados del siguiente modo:

a) Al inicio de la transmisión: seq_i = snd_una (snd_una es el ISN en este caso)

b) Durante la transmisión
   - seq_f=snd_una (menor número de secuencia de paquete no reconocido)
   - count = (seq_f-seq_i)/mss_size (número de paquetes transmitidos después de una pérdida)

c) Cuando ocurre una pérdida:
   - seq_f equivale a el número de secuencia del paquete perdido,
   - seq_f – seq_i equivale al número de bytes enviados entre dos pérdidas.
   - count = (seq_f – seq_i)/mss_size equivale a la cantidad de segmentos transmitidos entre dos pérdidas.

d) Despues que ocurre cada pérdida y se ha calculado count, seq_i adopta el último valor de seq_f, que será el menor número de secuencia enviado después de la pérdida.

Con respecto a ESTP, la variable count equivale a δ-1.

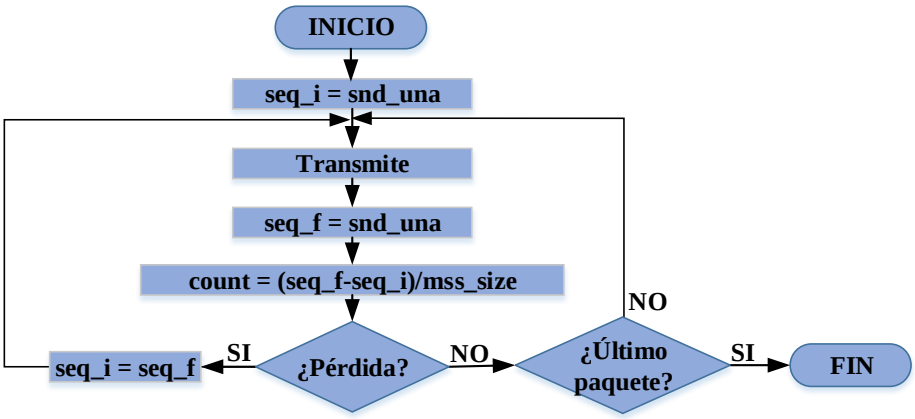


Figura 36. Contador de paquetes transmitidos satisfactoriamente después de una pérdida.

3.1.3. Construcción de una matriz de valores.

Un elemento importante a definir en el código fuente es el valor de δ, que representa la cantidad de paquetes transmitidos entre 2 pérdidas más 1, mientras β representa el factor de decremento multiplicativo que es empleado cuando existe congestión.El valor de δ tomará diferentes valores durante la transmisión. En el momento en que se produce una congestión debido a pérdida, se buscará el valor β que corresponde al valor δ.

En el archivo de extensión .c del código fuente de ESTP, se debe definir una matriz de valores δ.

3.1.3.1. Matriz de valores δ y β para un valor "τ" conocido.

En el código fuente, los valores numéricos son interpretados sólo como tipo "int" (entero). Debido a que el valor de β es inferior a 1, se requiere emplear el valor de β multiplicado por 256, y el valor obtenido es dividido por 256 en el programa al momento de ejecutar el algoritmo de decremento multiplicativo, cuando éste es empleado.

Una matriz para un valor "τ" fijo (conocido) sería de la siguiente forma.

$$estp_{values} = \begin{bmatrix} \delta_1 & \beta'_1 \\ \delta_2 & \beta'_2 \\ \vdots & \vdots \\ \delta_n & \beta'_n \end{bmatrix}$$

Donde "n" es la cantidad de valores de δ a considerar en la matriz y a cada δ_i le corresponde un valor β_i, sin embargo el valor a emplearse en la matriz es β'_i, siendo $\beta'_i = \beta_i \times 256$.

La matriz en lenguaje C para ser escrita en código fuente tendría la siguiente sintaxis:

```
static const struct <Nombre de la función>{
            unsigned int <Nombre de columna de valores δ>;
            unsigned int <Nombre de columna de valores β'ᵢ >;
} <Nombre de la matriz>[] = {
 {δ1, β'1, /* β1 */ },
 {δ2, β'2, /* β2 */ },
    ⋮  ⋮      ⋮
 {δn, β'n, /* βn */ },
}
```

La generación de una matriz en lenguaje C es posible mediante la herramienta Matlab. Las líneas de instrucción en Matlab para generar una matriz para valores de δ desde 1 a 315, considerando τ=50, son mostradas en la sección 1.1 del Anexo G.

La desventaja de emplear una matriz de éste tipo es que se requiere conocer el valor de τ, es decir operaría con una transmisión con pérdidas "determinantes", sin embargo en una conexión real, las pérdidas son aleatorias, por tal motivo no es válido. Además, para cada valor de τ, es decir para cada proporción de pérdidas distintas se tendría que hacer una nueva matriz, y por consecuencia, modificar el archivo de extensión ".c" que permite la ejecución del protocolo.

3.1.3.2. Matriz de valores para un valor "τ" no conocido.

En una conexión real, la probabilidad de pérdidas es desconocida. Una opción para programar el protocolo es construir una matriz con columnas que representen los valores de δ, τ y β, sin embargo ésta sería muy grande. Para evitar el uso de una matriz muy grande, se emplea el exponente de la función de mapeo de ESTP como una de las columnas de la matriz. Llamaremos a la variable **"exp"**:

$$\exp = (\delta - 1)/\tau \qquad \textbf{(31)}$$

El valor mínimo de "exp" es 0, debido a que el valor mínimo de δ es 1. Los valores de **exp** y **β** estarían dados en términos decimales, por tal motivo en la matriz se representa: **exp'=exp×100** y **β'=β×256**. La matriz en lenguaje C para ser escrita en código fuente tendría la siguiente sintaxis:

```
static const struct <Nombre de la función>{
            unsigned int <Nombre de columna de valores de: expᵢ' = expᵢ× 100>;
            unsigned int <Nombre de columna de valores β'ᵢ >;
} <Nombre de la matriz>[] = {
 {exp'1, β'1, /* β1 */ },
 {exp'2, β'2, /* β2 */ },
    ⋮  ⋮      ⋮
 {exp'n, β'n, /* βn */ },
}
```

Al emplear esta matriz la obtención de los valores verdaderos de δ y τ se halla dentro del programa empleando las variables **seq_i, seq_f, snd_una, mss_size y mss_cache.**

La generación de esta matriz mediante MATLAB considerando valores de "exp" desde 0 a 0.06 es dada por las líneas de instrucción que se observan en la sección 2.1 del Anexo G. La matriz generada es empleada en la programación del protocolo ESTP, apreciable en Anexo J.

3.2. Algoritmo de ESTP

La secuencia sin considerar CIR y EIR definidos (Es decir la tasa de información es de mejor esfuerzo dentro de los límites de la tasa máxima soportada por el canal), es como se explica:

3.2.1. Algoritmo de ESTP empleando τ fijo

a) El transmisor inicia el envío de paquetes. La ventana realiza un "slow start".

b) Luego de que el tamaño de la ventana alcanza el valor del "slow-start-threshold", la ventana enviada sigue un incremento aditivo, aumentando en 1 cada vez que todos los paquetes han sido reconocidos.

c) Un contador ("count") cuenta los paquetes transmitidos desde la última pérdida. "count" equivale a δ-1 (δ: Cantidad de paquetes transmitidos entre dos pérdidas más uno). Siendo count = (seq_f – seq_i)/mss_size

d) Una variable "exp" es el valor de (δ-1)/τ multiplicado por 100.

e) Por cada valor de (δ-1)/τ se obtiene un valor de β (nombrado "dec" en el programa) el cual es buscado en la matriz de valores definido previamente para realizar el decremento multiplicativo cuando ocurre una congestión.

3.2.2. Algoritmo de ESTP que calcula el valor τ automáticamente.

a) Los pasos a), b) y c) de la sección 1.4.1

b) Una variable "delta" es igual a "count" +1

c) Se emplea una variable nloss_indicator, la cual es 1 cuando ocurre una pérdida, y es 0 cuando no existe pérdida.

d) Un contador n_loss cuenta la cantidad de pérdidas producidas en la transmisión.

e) Una variable "send_packet" es la suma de los valores δ hasta el momento de la pérdida.

f) Una variable "tau" es el promedio de los valores de δ. Es decir:

g) tau = send_packet/n_loss

h) Los pasos d) y e) de la sección 1.4.1

3.3. Diagrama de flujo de ESTP

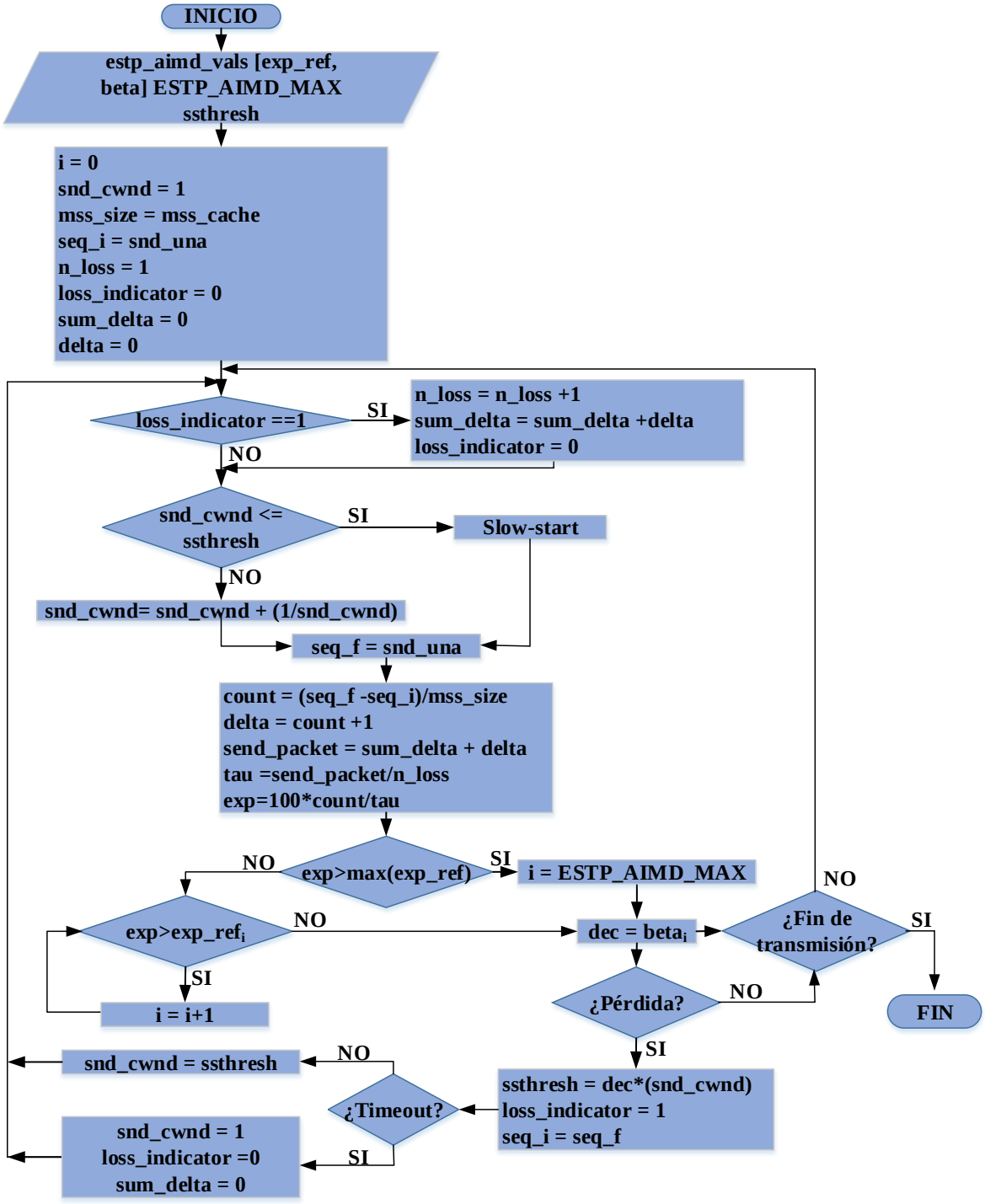


Figura 37. Diagrama de flujo de ESTP

3.4. ESTP con un valor “τ” fijo.

El programa de ESTP en lenguaje C se ubica en el Anexo J.

3.4.1. Análisis de β de ESTP empleando un valor τ fijo en una transmisión con pérdidas constantes.

a) Contraste de suposiciones

Suposición A: Existe una alta correlación positiva entre β de ejecución, β teórico y el decremento real.

Suposición B: β de ejecución ≈ β teórico.

- **Consideraciones de las pruebas a realizar.**
  - Total de pruebas: 10 transmisiones.
  - RTT = 50ms y MSS = 1448 Bytes
  - Cada transmisión dura 60 segundos aproximadamente.
  - Cada transmisión emplea diferente valor de δ.
  - Los valores de δ empleados son 20, 40, 60, 80, 100, 120, 140, 160, 180 y 200.
- **Análisis:**

- Se emplea el archivo de captura de TSHARK de cada transmisión realizada, y se seleccionan los paquetes recibidos con un ACK duplicado 3 o más veces. Estos números de ACK son los números de secuencia de los paquetes perdidos (retransmitidos posteriormente).
- En el algoritmo planteado cada paquete perdido es **seq_f**; y seq_i es el valor de seq_f del conjunto de paquetes transmitidos anteriores a la pérdida pasada.
- Se emplea MATLAB para obtener los valores de **seq_f** y **seq_i** desde el archivo de captura de TSHARK, y se divide por el valor de MSS, para obtener el valor de β experimental.

- **Valores obtenidos:**

  Los resultados son mostrados en la tabla 25.

- **Correlación entre β teórico y β de ejecución:**

  Se emplea R para hallar la correlación de los valores de β teórico y β de ejecución obtenidos por medio de R. En el Anexo "N" se observa:
  - El coeficiente de correlación (r) entre β teórico y β de ejecución es igual a 0.9999780.
  - El coeficiente de correlación (r) entre β de ejecución y el decremento real es igual a 0.9894308.
  - El coeficiente de correlación (r) entre β teórico y el decremento real es igual a 0.9887388.

  Siendo el coeficiente de correlación en los tres casos mencionados casi igual a 1: r≈ 1, entonces **se acepta la suposición A.**

N°	δ	τ	map	β teórico	β de ejecución	Decremento real	Error 1(*)	Error 2 (**)
1	20	100	1.827	0.547	0.547	0.501	0.000%	8.410%
2	40	100	1.677	0.596	0.598	0.589	0.336%	1.505%
3	60	100	1.554	0.643	0.646	0.637	0.467%	1.393%
4	80	100	1.454	0.688	0.690	0.713	0.291%	3.333%
5	100	100	1.372	0.729	0.730	0.749	0.137%	2.603%
6	120	100	1.304	0.767	0.768	0.788	0.130%	2.604%
7	140	100	1.249	0.801	0.802	0.817	0.125%	1.870%
8	160	100	1.204	0.831	0.832	0.839	0.120%	0.841%
9	180	100	1.167	0.857	0.858	0.857	0.117%	0.117%
10	200	100	1.137	0.880	0.880	0.872	0.000%	0.909%

(*) Error 1 = | [(β de ejecución) – (β teórico)] / (β teórico) |

(**) Error 2 = | [(Decremento real) – (β de ejecución)] / (β de ejecución) |

Tabla 25. Comparación de β teórico, β de ejecución y decremento real.

- **Test de hipótesis empleando R:**

  Es posible realizar el test de hipótesis empleando R, mediante la función t-test (Test de Student), empleando un intervalo de confianza de 95%.

  La suposición B (hipótesis nula) se puede expresar como:

  Hipótesis nula: β de ejecución - β teórico =0

  Hipótesis alternativa: β de ejecución - β teórico ≠ 0

  El resultado del test de hipótesis empleando R (Ver Anexo "N") consiste en:

- Si el valor de p es mayor 0.05, la hipótesis nula no se puede rechazar. Caso contrario, si el valor de p es menor o igual a 0.05, la hipótesis alternativa se acepta.
- Por lo tanto según test de Student realizado en R, que muestra un valor "p" < 0.05, se rechaza la hipótesis nula, esto es: β de ejecución ≠ β teórico**.**
- Se observan un intervalo de confianza muy "estrecho": [-0.0017755989; -0.0006754929], lo cual significa un error muy bajo en la diferencia entre medias de cada conjunto de datos.

  Si se observan los datos de la tabla 25, el error máximo de los valores de β de ejecución con respecto a β teórico es 0.467%. Siendo $\varepsilon = 0.467\% \times (\beta_{teorico})$, podemos decir entonces:

$$\beta_{ejecucion} = \beta_{teorico} \pm \varepsilon \quad \textbf{(32)}$$

  Por lo tanto, se puede decir $\beta_{ejecucion} \approx \beta_{teorico}$, siendo la **suposición B aceptada.**

b) Error del decremento real con respecto a β de ejecución.

En base a los datos de la tabla 25, se calcula lo siguiente para éste caso:

Error máximo = 8.410%, Error mínimo = 0.841%, Media del error = 2.359%.

Al ser el valor de media de error muy bajo, se puede afirmar que el **protocolo está siendo ejecutado correctamente.**

3.4.2. Análisis de β de ESTP con pérdidas constantes en un conjunto de muestras

a) Suposiciones a contrastar.

Suposición A: La relación (rendimiento)/(ventana de congestión promedio) es constante.

Suposición B: El decremento real es constante en cada muestra.

---

b) Consideraciones de las pruebas a realizar.

- 4 Muestras con: RTT=50ms y δ=100, RTT=50ms y δ=50, RTT=100ms y δ=100, RTT=100ms y δ=50.
- Total de pruebas por muestra: 10 transmisiones.
- Cada transmisión dura 60 segundos aproximadamente, y emplea MSS = 1448 Bytes.
- Cada transmisión emplea un mismo valor de δ (pérdida constante).

c) Valores obtenidos.

- **Valores obtenidos por IPERF**

  Estos valores son mostrados en la tabla 26, en la cual se observa una desviación estándar baja que indica una tendencia de aproximación del rendimiento obtenido en cada muestra al promedio para cada muestra.

	Rendimiento (Mb/s)			
N° de transmisión	**Muestra 1**	**Muestra 2**	**Muestra 3**	**Muestra 4**
1	3.350	2.390	1.740	1.290
2	3.250	2.330	1.650	1.260
3	3.240	2.320	1.700	1.240
4	3.300	2.350	1.740	1.250
5	3.330	2.310	1.690	1.290
6	3.320	2.340	1.730	1.240
7	3.210	2.310	1.710	1.240
8	3.290	2.330	1.750	1.290
9	3.190	2.360	1.690	1.240
10	3.370	2.330	1.750	1.250
Promedio	**3.285**	**2.337**	**1.715**	**1.259**
Desviación estándar	0.060	0.025	0.033	0.022

Tabla 26. Rendimiento obtenido por IPERF empleando ESTP (Pérdidas constantes).

- **Rendimiento obtenido empleando datos de captura de TCP Probe y Tshark.**

  Los datos generados empleando Matlab para el cálculo, considerando un MSS = 1448 Bytes, se muestran en la tabla 27. Se observa que la desviación estándar por cada una de las muestras es baja, lo cual demuestra que los valores de rendimiento de las 10 transmisiones en cada muestra son equivalentes.

N° de transmisión	Rendimiento (Mb/s)							
	Muestra 1		Muestra 2		Muestra 3		Muestra 4	
	TCPProbe	Tshark	TCPProbe	Tshark	TCPProbe	Tshark	TCPProbe	Tshark
1	3.452	3.177	2.504	2.309	1.703	1.679	1.276	1.251
2	3.328	3.192	2.459	2.297	1.636	1.597	1.241	1.212
3	3.322	3.191	2.448	2.297	1.670	1.651	1.243	1.202
4	3.334	3.261	2.462	2.308	1.715	1.689	1.249	1.217
5	3.353	3.267	2.425	2.295	1.673	1.653	1.254	1.229
6	3.373	3.253	2.437	2.299	1.721	1.672	1.223	1.192
7	3.288	3.183	2.440	2.297	1.684	1.655	1.220	1.198
8	3.373	3.198	2.456	2.298	1.727	1.695	1.273	1.233
9	3.292	3.178	2.467	2.314	1.673	1.625	1.234	1.211
10	3.412	3.272	2.449	2.300	1.713	1.690	1.242	1.201
Promedio	3.353	3.217	2.455	2.301	1.692	1.661	1.246	1.215
Desviación estándar	0.052	0.040	0.022	0.006	0.029	0.031	0.019	0.018

Tabla 27. Rendimiento obtenido por TCP Probe y Tshark empleando ESTP. (Pérdidas constantes)

- **Ventana de congestión promedio calculada de los datos de TCP Probe**

  Los valores de ventana de congestión promedio para cada transmisión que conforman cada una de las muestras se muestran en la tabla 28, donde existe una desviación estándar baja en cada muestra, lo cual demuestra equivalencia entre los valores dentro de cada muestra.

N° de transmisión	Ventana de congestión promedio(Paquetes)			
	Muestra 1	Muestra 2	Muestra 3	Muestra 4
1	14.901	10.810	14.705	11.013
2	14.367	10.613	14.122	10.714
3	14.337	10.567	14.419	10.731
4	14.389	10.625	14.802	10.784
5	14.474	10.466	14.446	10.826
6	14.560	10.518	14.856	10.560
7	14.191	10.531	14.541	10.534
8	14.557	10.601	14.909	10.993
9	14.210	10.650	14.444	10.654
10	14.728	10.570	14.786	10.725
Promedio	**14.471**	**10.595**	**14.603**	**10.754**
Desviación estándar	0.223	0.093	0.250	0.160

Tabla 28. Ventana de congestión promedio empleando ESTP (Pérdidas constantes)

- **Decremento real**

  La desviación estándar de los valores de decremento real dentro de cada una de las muestras es menor o igual a 0.01, lo cual permite asumir que el decremento real es constante en cada muestra (Ver tabla 29).

N° de transmisión	Decremento real			
	Muestra 1	Muestra 2	Muestra 3	Muestra 4
1	0.749	0.749	0.747	0.748
2	0.748	0.749	0.746	0.746
3	0.749	0.748	0.746	0.747
4	0.748	0.748	0.747	0.747
5	0.748	0.748	0.746	0.748
6	0.749	0.749	0.747	0.746
7	0.748	0.748	0.746	0.745
8	0.749	0.748	0.747	0.748
9	0.747	0.749	0.746	0.745
10	0.749	0.748	0.747	0.747
Promedio	0.748	0.748	0.747	0.747
Desviación estándar	0.000	0.001	0.001	0.001

Tabla 29. Decremento real promedio empleando ESTP (Pérdidas constantes)

- **Contraste de suposiciones.**

Para contrastar la suposición A se realiza un test de igualdad de proporciones. El test de igualdad de proporciones es posible realizarse mediante R. Se verifica si las proporciones:

[Rendimiento]/[cwnd] son iguales entre si dentro de una de las muestras.

Los resultados de test de proporciones son:

Herramienta de cálculo de rendimiento.	Valores de "p"			
	Muestra 1	Muestra 2	Muestra 3	Muestra 4
TCP Probe	**1**	**1**	**1**	**1**
Tshark	**1**	**1**	**1**	**1**

Tabla 30. Valor "p" de tests de proporciones (Ver Anexo "N").

Los valores obtenidos indican una relación lineal entre rendimiento y ventana de congestión promedio. Esto es: **Rendimiento = k. cwnd**, donde "k" es una constante, que varía dependiendo de los parámetros de una muestra, considerando ESTP con pérdidas constantes.

Para contrastar la suposición B, de igual forma se realiza un test de igualdad de proporciones, se compara la desviación estándar con el valor "0" en cada muestra de valores de decremento real. Esto es: si desviación estándar ≈ 0, entonces el decremento real es constante en una muestra.

En la tabla 30, se observa que las desviaciones estándares obtenidas en cada muestra son 0, 0.001, 0.001 y 0.001. Por tal motivo se puede afirmar para este análisis: desviación estándar ≈ 0, por tanto el **decremento real es constante** dentro de una muestra que emplea ESTP bajo pérdidas constantes**.**

- **Conclusión:** Se aceptan las suposiciones A y B.

3.5. ESTP con un valor "τ" autoajustable.

ESTP debe ser un protocolo capaz de ajustar el valor de "τ" dinámicamente. Bajo este esquema, ESTP hace una estimación de $\tau = \frac{1}{n}\sum_{i=1}^{n} \delta$ donde "n" es la cantidad de pérdidas por triple ACK duplicado sin que se haya producido una pérdida por "timeout". Es decir puede existir un valor "τ" diferente en cada pérdida.

La ocurrencia de "timeouts" ocasiona que por un tiempo se estime un paquete como el último reconocido, habiéndose enviado quizá muchos paquetes en el periodo de congestión. Por este motivo se estima nuevamente "τ" después de cada pérdida por "timeout" considerando que las pérdidas de este tipo ocurren mucho menos que aquellas por triple ACK duplicado.

3.5.1. Pruebas de ESTP con un valor "τ" autoajustable.

La verificación de un ESTP para un valor de "τ" autoajustable en la programación de este protocolo se verifica empleando pérdidas constantes. Se realiza las siguientes pruebas:

- Una transmisión de 60 segundos con RTT=50ms y un porcentaje de pérdidas de 1%.
- Una transmisión de 60 segundos con RTT=50ms y un porcentaje de pérdidas de 2%.
- Una transmisión de 60 segundos con RTT=100ms y un porcentaje de pérdidas de 1%.
- Una transmisión de 60 segundos con RTT=100ms y un porcentaje de pérdidas de 2%.

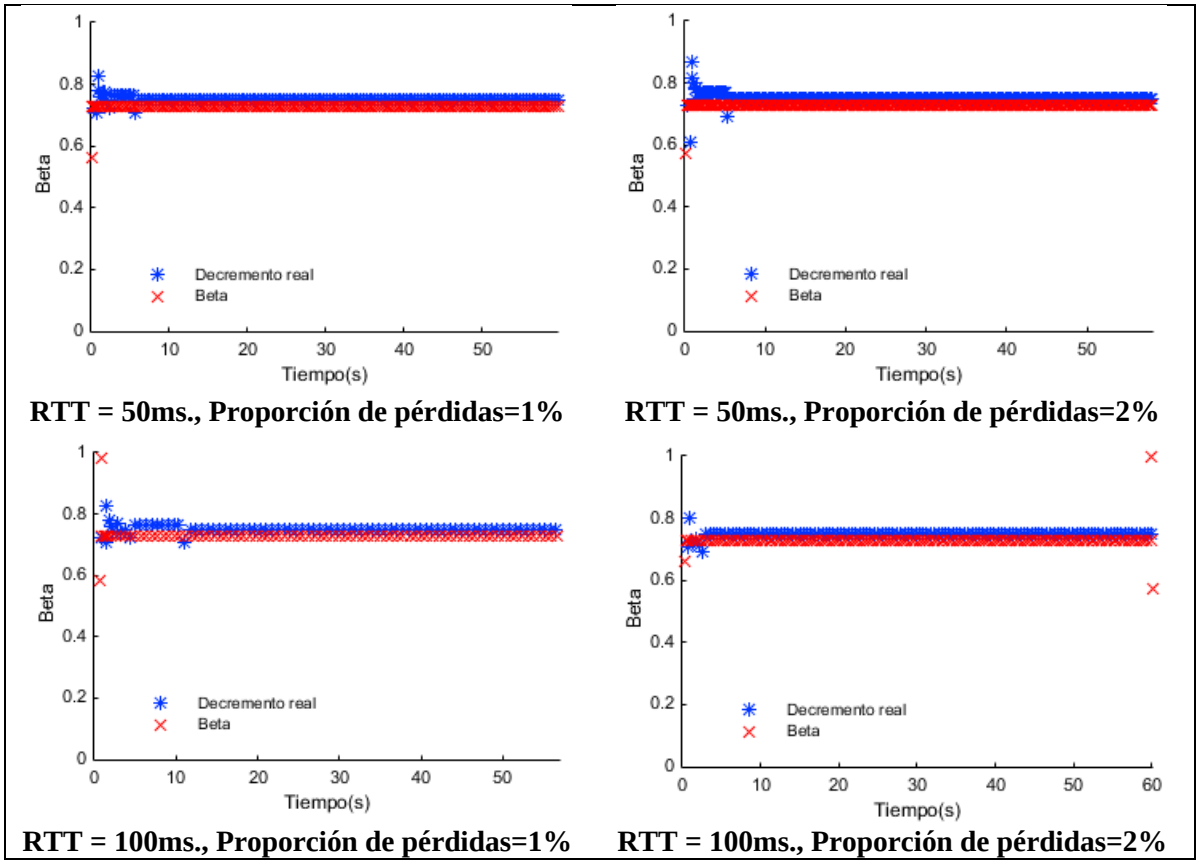
3.5.1.1. Verificación de comportamiento del decremento multiplicativo real.

Se compara el decremento multiplicativo promedio de cada caso empleando un valor de "τ" fijo (Revisar tabla 29) con el decremento multiplicativo calculado empleando un valor de "τ" autoajustable.

RTT	50ms.	50ms.	100ms.	100ms.
Porcentaje de pérdidas	**1%**	**2%**	**1%**	**2%**
Decremento real empleando "τ" fijo.	0.749	0.749	0.747	0.748
Decremento real empleando "τ" autoajustable.	0.751	0.752	0.751	0.749
Error (%)	0.267	0.401	0.535	0.134

Tabla 31. Decremento real en ESTP cuando "τ" es autoajustable.

En la tabla 31, se observa que el error es despreciable (Error promedio es 0.334%), razón por la cual se puede afirmar respecto al decremento multiplicativo (DM) en un escenario de pérdidas constantes: DM empleando un "τ" fijo en ESTP ≈ DM empleando un "τ" autoajustable en ESTP.



RTT = 50ms., Proporción de pérdidas=1% **RTT = 50ms., Proporción de pérdidas=2%**

RTT = 100ms., Proporción de pérdidas=1% **RTT = 100ms., Proporción de pérdidas=2%**

Figura 38. Decremento real de ESTP con "τ" autoajustable

3.5.2. Análisis de β de ESTP con pérdidas aleatorias en un conjunto de muestras.

3.5.2.1. Suposición a contrastar.

Suposición A: El rendimiento promedio al emplear ESTP es mayor que si se emplea TCP Reno o GAIMD.

3.5.2.2. Consideraciones de las pruebas a realizar.

- 4 Muestras con: RTT = 50ms y 1% de pérdidas, RTT = 50ms y 2% de pérdidas, RTT = 100ms y 1% de pérdidas, RTT = 100ms y 2% de pérdidas.
- Total de pruebas por muestra: 10 transmisiones.
- RTT = 50ms y MSS = 1448 Bytes
- Cada transmisión dura 60 segundos aproximadamente.

3.5.2.3. Valores obtenidos.

- **Valores obtenidos por IPERF**

Los valores obtenidos por IPERF, por cada una de las transmisiones de cada muestra son dados en la tabla 32, observándose una desviación estándar baja que indica una tendencia de aproximación del rendimiento obtenido en cada muestra al promedio para cada muestra.

N° de transmisión	Rendimiento (Mb/s)			
	Muestra 1	Muestra 2	Muestra 3	Muestra 4
1	3.640	2.600	1.770	1.270
2	3.560	2.290	1.780	1.340
3	3.530	2.450	1.780	1.360
4	3.280	2.390	1.460	1.500
5	3.400	2.570	1.730	1.270
6	3.190	2.500	1.780	1.400
7	3.090	2.470	1.870	1.380
8	3.750	2.520	1.820	1.330
9	3.250	2.640	1.720	1.220
10	3.560	2.560	2.020	1.270
Promedio	**3.425**	**2.499**	**1.773**	**1.334**
Desviación estándar	**0.205**	**0.099**	**0.132**	**0.077**

Tabla 32. Rendimiento obtenido por IPERF empleando ESTP (Pérdidas aleatorias).

- **Rendimiento obtenido empleando datos de captura de TCP Probe y Tshark.**

  Los datos generados empleando Matlab para el cálculo, considerando un MSS = 1448 Bytes, se muestran en la siguiente tabla 33. La desviación estándar por cada una de las muestras es baja, lo cual demuestra que los valores de rendimiento de las 10 transmisiones en cada muestra son equivalentes.

N° de transmisión	Rendimiento (Mb/s)							
	Muestra 1		Muestra 2		Muestra 3		Muestra 4	
	TCPProbe	Tshark	TCPProbe	Tshark	TCPProbe	Tshark	TCPProbe	Tshark
1	3.715	3.575	2.689	2.342	1.756	1.708	1.269	1.209
2	3.720	3.188	2.389	2.250	1.750	1.713	1.333	1.298
3	3.610	3.441	2.515	2.383	1.791	1.747	1.337	1.313
4	3.383	2.997	2.480	2.350	1.480	1.444	1.465	1.438
5	3.533	3.231	2.633	2.530	1.727	1.685	1.242	1.214
6	3.279	3.152	2.552	2.481	1.800	1.737	1.393	1.321
7	3.204	3.054	2.545	2.448	1.868	1.812	1.353	1.319
8	3.851	3.648	2.611	2.513	1.825	1.764	1.323	1.273
9	3.315	3.157	2.705	2.568	1.709	1.668	1.226	1.187
10	3.647	3.384	2.644	2.522	2.061	1.962	1.280	1.236
Promedio	**3.526**	**3.283**	**2.576**	**2.439**	**1.777**	**1.724**	**1.322**	**1.281**
Desviación estándar	0.219	0.219	0.099	0.103	0.145	0.129	0.072	0.074

Tabla 33. Rendimiento obtenido por TCP Probe y Tshark empleando ESTP.

(Pérdidas aleatorias)

- **Ventana de congestión promedio calculada de los datos de TCP Probe**

  Los valores de ventana de congestión promedio para cada transmisión que conforman cada una de las muestras se ven en la tabla 34, en la cual se observa una desviación estándar baja en cada muestra, lo cual demuestra, a su vez, equivalencia entre los valores dentro de cada muestra.

	Ventana de congestión promedio(Paquetes)			
N° de transmisión	**Muestra 1**	**Muestra 2**	**Muestra 3**	**Muestra 4**
1	16.037	11.608	15.157	10.956
2	16.055	10.312	15.103	11.503
3	15.580	10.855	15.457	11.545
4	14.601	10.703	12.776	12.642
5	15.247	11.366	14.908	10.722
6	14.152	11.015	15.543	12.022
7	13.829	10.985	16.129	11.682
8	16.623	11.270	15.754	11.417
9	14.307	11.674	14.757	10.581
10	15.739	11.413	17.787	11.047
Promedio	**15.217**	**11.120**	**15.337**	**11.412**
Desviación estándar	0.944	0.427	1.249	0.623

Tabla 34. Ventana de congestión promedio empleando ESTP (Pérdidas constantes).

- **Decremento real**

	Decremento real			
N° de transmisión	**Muestra 1**	**Muestra 2**	**Muestra 3**	**Muestra 4**
1	0.718	0.683	0.715	0.715
2	0.670	0.711	0.700	0.712
3	0.704	0.714	0.663	0.725
4	0.667	0.701	0.678	0.733
5	0.699	0.719	0.723	0.702
6	0.692	0.714	0.710	0.703
7	0.697	0.706	0.716	0.740
8	0.731	0.712	0.707	0.692
9	0.684	0.709	0.668	0.709
10	0.700	0.722	0.728	0.714
Promedio	0.696	0.709	0.701	0.715
Desviación estándar	0.020	0.011	0.023	0.015

Tabla 35. Decremento real promedio empleando ESTP (Pérdidas constantes).

La desviación estándar de los valores de decremento real dentro de cada una de las muestras es baja, lo cual permite asumir que el decremento real promedio es equivalente entres las pruebas (transmisiones) en un mismo escenario de RTT y pérdidas.

Sin embargo el decremento real varía dentro de cada una de las transmisiones como se puede observar en las imágenes del Anexo M.

3.5.2.4. Contraste de la suposición A.

Para contrastar la suposición A se toman los valores de rendimiento promedio en cada uno de los cuatro casos:

Muestra1: RTT=50ms, Pérdidas 1%, Muestra2: RTT=50ms, Pérdidas 2%, Muestra3: RTT=100ms, Pérdidas 1%, Muestra4: RTT=100ms, Pérdidas 2%,

Muestra	Rendimiento promedio (Mb/s)								
	iPerf			TCP Probe			Tshark		
	RENO	GAIMD	ESTP	RENO	GAIMD	ESTP	RENO	GAIMD	ESTP
1	2.470	3.254	3.425	2.530	3.315	3.526	2.353	3.144	3.526
2	1.770	2.353	2.499	1.844	2.396	2.576	1.726	2.285	2.439
3	1.310	1.663	1.773	1.301	1.693	1.777	1.301	1.649	1.724
4	0.926	1.210	1.334	0.929	1.200	1.322	0.886	1.155	1.281

Tabla 36. Rendimiento promedio por cada muestra de TCP RENO, GAIMD y ESTP.

(Muestra de 10 transmisiones por caso)

Para evaluar la suposición A, asumiremos:

x: Rendimiento empleando TCP Reno

y: Rendimiento empleando GAIMD

z: Rendimiento empleando ESTP

a) Test de Student

Se emplea el test de Student para comparar si z >x. La hipótesis nula es: z – x = 0, y la hipótesis alternativa es: z- x ≠ 0.

Se evalúa p < 0.05, entonces se acepta z-x≠ 0. Siendo esto último si el intervalo de confianza comprende números mayores a 0, se puede decir que z – x >0, que es equivalente a decir z > x.

	Origen de datos	Valor de "p"	Intervalo de confianza	Conclusión
Test 1	iPerf	0.01501	[0.2358921; 1.0416079]	z > x
Test 2	TCP Probe	0.02132	[0.1922923; 1.1702077]	z > x
Test 3	Tshark	0.02972	[0.1160445; 1.1309555]	z > x

Tabla 37. Resultados del test de Student que valida z >x

En los test 1, test 2 y test 3 se cumple las condiciones para afirmar z >x.

Ahora se aplica el test de Student para evaluar si z > y, de forma análoga a la comparación anterior.

	Origen de datos	Valor de "p"	Intervalo de confianza	Conclusión
Test 4	iPerf	0.001934	[0.09532318; 0.18017682]	z > y
Test 5	TCP Probe	0.020550	[0.05264505; 0.30785495]	z > y
Test 6	Tshark	0.09207	[-0.03693230; 0.28193230]	ND

Tabla 38. Resultados del test de Student que valida z >y

En el test 6 no se puede afirmar si z > y, debido a que p>0.05, sin embargo tampoco se puede afirmar una igualdad debido a que el intervalo de confianza incluye el valor "0". Los tests 4 y 5 cumplen las condiciones para afirmar z >y.

Se concluye:

"EL RENDIMIENTO ES MAYOR CUANDO SE EMPLEA GAIMD EN COMPARACIÓN A TCP RENO"

"EL RENDIMIENTO ES MAYOR O IGUAL CUANDO SE EMPLEA ESTP EN COMPARACIÓN A GAIMD"

b) Correlación

Los coeficientes de correlación obtenidos empleando R son:

	Test 1	Test 2	Test 3
Coeficiente de correlación entre "x" y "z"	0.9988891	0.9996327	0.9953318

Tabla 39. Coeficiente de correlación entre "x" y "z"

	Test 4	Test 5	Test 6
Coeficiente de correlación entre "y" y "z"	0.9999272	0.9995496	0.9976917

Tabla 40. Coeficiente de correlación entre "y" y "z"

El coeficiente de correlación obtenido en todos los casos es: r≈ 1. Entonces se dice:

"Existe linealidad entre el rendimiento de Tcp Reno y ESTP"

"Existe linealidad entre el rendimiento de GAIMD y ESTP"

Teniendo en cuenta los resultados del test de Student e intervalos de confianza de éste último, y los valores de coeficiente de correlación obtenidos:

SE ACEPTA LA SUPOSICIÓN A.

Los resultados del test de hipótesis en R, se observan en el Anexo N.

CONCLUSIONES

A. El rendimiento expresado en bits por segundo, en una transmisión depende del (de los) algoritmo(s) de control de congestión empleado(s).

B. El valor de RTT y porcentaje de pérdidas en una transmisión influye en el rendimiento. En los protocolos TCP, GAIMD y ESTP se observa a mayor valor de RTT y a mayor porcentaje de pérdidas, el rendimiento disminuye.

C. Modificar el valor de incremento aditivo y/o decremento multiplicativo en TCP o en algún protocolo que es variante del primero, influye en el rendimiento (bits/segundo).

D. El rendimiento de GAIMD ($\alpha = 0.31$ y $\beta = 0.875$) es proporcional al rendimiento de TCP en iguales escenarios. En base a resultados experimentales se dedujo la expresión: $T_{GAIMD} / T_{RENO} = 1.303 \pm \varepsilon$, donde ε es un valor de error despreciable. Esta expresión se aproxima a la relación calculada teóricamente: $T_{GAIMD} / T_{RENO} \approx 1.24$.

E. Cuando se emplea ESTP en un escenario de pérdidas constantes, el comportamiento de la ventana de congestión tiende a ser periódico y estacionario. Por lo tanto al hacer una predicción de la ventana para éste tipo de escenario, ésta ("ventana de predicción") será una aproximación a la ventana de congestión presentada experimentalmente.

F. Cuando se emplea ESTP en un escenario de pérdidas aleatorias, el comportamiento de la ventana de congestión dependerá de cuanta información se han transmitido entre una pérdida y otra. Si se transmiten muchos paquetes entre dos pérdidas la tendencia será que el decremento multiplicativo se aproxime a 1, caso contrario la tendencia del decremento multiplicativo es aproximarse a 0.5.

G. En base a pruebas experimentales, se puede decir que el rendimiento que ofrece una transmisión que emplea ESTP en escenarios iguales a GAIMD ($\alpha = 0.31$ y $\beta = 0.875$), es mayor o igual a éste último.

H. ESTP ofrece mejor rendimiento que TCP Reno.

I. En los protocolos de transporte TCP, GAIMD y ESTP, la ventana de congestión promedio y el rendimiento son valores altamente correlacionados.

J. El procesamiento de datos capturados por tshark y tcpprobe con Matlab, permite contrastar el valor de rendimiento que es ofrecido por la herramienta iPerf.

K. El diseño del banco de pruebas presentado usa valores ofrecidos por los archivos de captura de tcpprobe y tshark, lo que hace posible evaluar otros protocolos de transporte.

BIBLIOGRAFIA

[1] K. R. Fall y R. Stevens, TCP/IP Illustrated, Vol. 1, The Protocols, Segunda ed., Adisson Wesley, Noviembre, 2011.

[2] J. Postel, Transmission Control Protocol, RFC 793, Menlo Park, California, Setiembre, 1981.

[3] T. Sockolofsky y C. Kale, A TCP/IP Tutorial, RFC 1180, Enero, 1991.

[4] V. Jacobson, Congestion Avoidance and Control, In Proceedings of ACM SIGCOMM, Agosto, 1988.

[5] M. Allman, V. Paxson y E. Blanton, TCP Congestion Control, RFC 5681, Setiembre, 2009.

[6] Y. R. Yang y S. S. Lam, General AIMD Congestion Control, The University of Texas at Austin: Technical Report TR-2000-09, Mayo, 2000.

[7] C. Estévez, A Carrier-Ethernet oriented Transport Protocol with a Novel Congestion Control and QoS Integration: Analytical, Simulated and Experimental Validation., IEEE ICC 2012-Next-Generation Networking Symposium, 2012.

[8] J. Dugan, «iPerf - The network bandwidth measurement tool,» [En línea]. Available: https://iperf.fr/. [Último acceso: 26 Octubre 2015].

[9] Linux Foundation, «tcp_testing,» [En línea]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/tcp_testing. [Último acceso: 26 Octubre 2015].

[10] W. Shotts, «IPTABLES,» [En línea]. Available: http://www.linuxcommand.org/man_pages/iptables8.html. [Último acceso: 27 Octubre 2015].

[11] G. Combs, «Wireshark,» [En línea]. Available: https://www.wireshark.org/docs/man-pages/tshark.html. [Último acceso: 26 Octubre 2015].

[12] R. Braden, Requirements for Internet Hosts -- Communication Layers, RFC 1122, Octubre, 1989.

[13] J. T. C. ISO/IEC, Information tecnhology – Open Systems Interconnection – Basic Reference Model: The Basic Model, ISO/IEC 7498, Ginebra, Suiza, Noviembre, 1994.

[14] M. Allman y S. Floyd, Increasing TCP's Initial Window, RFC 3390, Octubre, 2002.

[15] J. Padhye, V. Firoiu, D. F. Towsley y J. F. Kurose, Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation, IEEE/ACM TRANSACTIONS ON NETWORKING, Abril, 2000.

[16] J. Bolliger, T. Gross y U. Hengartner, Bandwidth modeling for network-aware applications, Proc. INFOCOMM'99, 1999.

[17] M. Mathis, J. Semke y J. Mahdavi, The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm, SIGCOMM ACM Special Interest Group on Data Communication, Julio, 1997.

[18] D. Papadimitriou, MEF Ethernet Traffic Parameters, Abril, 2006.

[19] I. T. Union, Ethernet service activation test methodology, ITU-T Y.1564, Marzo, 2011.

[20] Linux Foundation, «netem,» [En línea]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/netem. [Último acceso: 26 Octubre 2015].

[21] «socket,» die.net, [En línea]. Available: http://linux.die.net/man/7/socket. [Último acceso: 25 01 2016].

[22] M. Allman, TCP Congestion Control with Appropriate Byte Counting (ABC), RFC 3465, Febrero, 2003.

[23] G. Kroah-Hartman, Linux Kernel Development: How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring it., Marzo, 2008.

[24] «tcptrace,» [En línea]. Available: http://www.tcptrace.org/. [Último acceso: 20 Enero 2016].

[25] V. Paxson, S. Floyd y C. Kreibich, «sock,» The ICSI Networking and Security Group, [En línea]. Available: http://www.icir.org/christian/sock.html. [Último acceso: 22 Enero 2016].

[26] «tcp,» die.net, [En línea]. Available: http://linux.die.net/man/7/tcp. [Último acceso: 17 9 2015].

[27] «The Linux Kernel Archives,» The Linux Foundation, [En línea]. Available: https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt. [Último acceso: 20 10 2015].

[28] S. Floyd, Congestion Control Principles, RFC 2914, Setiembre, 2000.

ANEXOS

ANEXO "A"

INFORMACIÓN ADICIONAL SOBRE TCP

1. Cabecera de TCP

La estructura gráfica de la cabecera TCP se muestra en la figura A1.

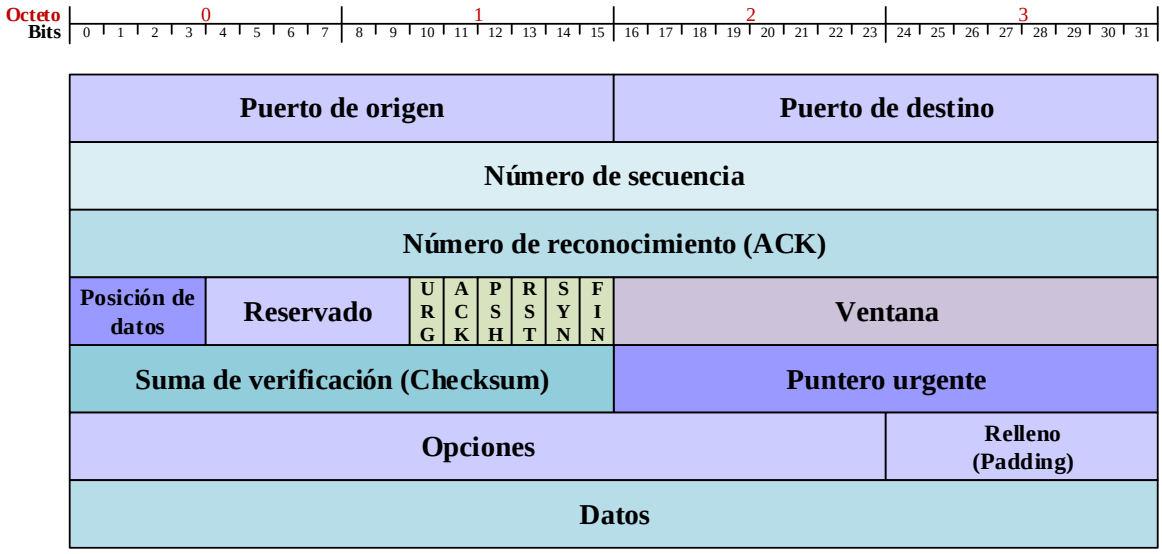


Figura A1. Cabecera TCP [2] [1].

En base a [2] y [1], se describen los campos de la cabecera de TCP:

a) Puerto de origen: Tiene una longitud de 16 bits. Es el número de puerto de origen.

b) Puerto de destino: Tiene una longitud de 16 bits. Es el número de puerto de destino.

c) Número de secuencia: Tiene una longitud de 32 bits. Si SYN está desactivado, el número de secuencia es el primer byte del flujo de datos enviado al receptor; y si SYN está activado, el campo representa el número de secuencia inicial (ISN: Initial Sequence Number). TCP enumera cada byte enviado con un número de secuencia. Éste es un número de 32 bits sin signo que retorna a 0 después de alcanzar el valor 232 - 1. [1]

d) Número de reconocimiento (ACK): Tiene una longitud de 32 bits. Este campo sólo es válido si el campo del bit de control ACK se encuentra activado. Contiene el siguiente número de secuencia de segmento que el receptor espera recibir del otro extremo. Es decir: Número de reconocimiento = Número de secuencia + 1.

e) Posición de datos: Tiene una longitud de 4 bits, y es llamado también "Longitud de cabecera" [1]. Su valor indica la posición antes de los datos, expresado en número de palabras de 32 bits. Sin incluir los campos de opciones (Opciones y relleno), este campo tiene un valor decimal de 5 (0001'b).

f) Bits de control: Cada campo bit de control tiene un valor booleano, es decir es activado y desactivado con los valores 0 y 1 respectivamente. Los bits de control son:

URG: Si su valor es 1, activa el campo Puntero urgente.
ACK: Si su valor es 1, activa el campo Número de reconocimiento
PSH: Función PUSH.
RST: Si su valor es 1, resetea la conexión.
SYN: Sincroniza números de secuencia.
Cuando su valor es 1, define el número de secuencia como ISN.
FIN: No existen más datos a ser enviados por el transmisor.

g) Ventana: Es el número de bytes, (iniciando con aquel especificado por el número ACK), que el extremo de la conexión está dispuesto a recibir. Este es un campo de 16 bits, lo que limita la ventana a 65535 bytes, y limita así el desempeño del "rendimiento" de TCP.

h) Suma de verificación (Checksum): Tiene una longitud de 16 bits. Este campo cubre la cabecera y los datos TCP, y algunos campos de la cabecera IP, utilizando un cálculo de pseudo-cabecera. Es obligatorio que el "checksum" sea calculado y almacenado por el remitente, y luego verificado por el receptor. El cálculo se realiza con el mismo algoritmo que la suma de comprobación de IP, ICMP y UDP ("Internet") [1]. Esta pseudo-cabecera contiene las direcciones de origen y de destino, el protocolo y longitud TCP (longitud de cabecera TCP + longitud de datos), dando una protección a TCP contra segmentos mal enrutados [2].

i) Puntero urgente: Es un desplazamiento positivo que debe ser añadido al número de secuencia del segmento, para obtener el último byte, cuando se trata de datos urgentes (URG = 1). Tiene una longitud de 16 bits.

j) Opciones: Es un campo de longitud variable. Su longitud es múltiplo de 8 bits. Todas las opciones son incluidas en el checksum. Una opción tal vez inicie sobre cualquier límite de octeto. Según [2], existen dos casos para el formato del campo opción: Cuanco comprende

un único octeto de “tipo de opción” ó cuando está formado por un octeto de “tipo” de opción, un octeto de “longitud” de opción y los octetos de datos de opción.

El octeto “tipo de opción” puede adquirir distinto valor según su significado. Existen 3 tipos de opción según su valor, definidas en [2], los cuales son:

- Fin de lista opción. (“tipo” = 00000000): Es usado como fin de todas las opciones. Se emplea sólo si el fin de las opciones no coincidiera con el fin de la cabecera TCP.
- No operación. (“tipo” = 00000001): Usado para alinear el inicio de una opción subsecuente sobre un límite de palabra. No existe garantía de que el transmisor use esta opción, por eso el receptor debe estar preparado para procesar opciones, incluso si éstas no inician sobre el límite de palabra.
- Máximo tamaño de segmento. (“tipo” = 00000010): Si los octetos de datos representan el máximo tamaño de segmento que el receptor está dispuesto a recibir. Este campo debe ser sólo enviado en la solicitud de conexión inicial, cuando SYN=1. Si esta opción no es usada, se adopta cualquier tamaño de segmento. Al usar este “tipo” de opción, se emplearán en total 4 bytes en el campo opciones: 1 byte de “tipo”, 1 byte de “longitud” y 2 bytes de datos, asimismo entonces el valor decimal de “longitud” es 4. El formato del campo Opciones sería: 00000010 00000100 XXXXXXXX XXXXXXXX

k) Relleno (Padding): Es de longitud variable. Es un conjunto de ceros, que se usan para completar el formato de palabras de 32 bits de la cabecera TCP.

2. Número de secuencia inicial.

Al iniciar una conexión, cada extremo envía un segmento que lleva activo el bit de control SYN, denominado “Segmento SYN”. Este segmento es identificado con un número de secuencia inicial (ISN: Initial Sequence Number), el cual es denominado “secuencia inicial de envío” (ISS: Initial send sequence) y “secuencia inicial de recepción” (ISR: Initial receive sequence), según corresponda al transmisor y receptor, respectivamente [1] [3]. Cada ISN tiene una longitud de 32 bits, y este es incrementado en 1 cada 4μs, con el objetivo de que los números de secuencia de segmentos en una conexión no se solapen con los números de secuencia de una conexión anterior, tal y como se especifica en [2], de tal forma que cada ISN rota cada 4.55 horas. También, para evitar este solapamiento, en [2] se expone el concepto “vida máxima de un segmento” (MSL: Maximum segment lifetime), que es el tiempo máximo que se permite a un segmento permanecer

en la red. Ante la caída o reinicio de una conexión se espera este tiempo MSL, para enviar un nuevo segmento; conocido como "tiempo en silencio". Por último, se sugiere emplear una suma de comprobación a nivel de capa de aplicación para asegurar la transferencia de datos sin errores [1]. En caso de Linux, adicionalmente se implementa la selección semi-aleatoria del ISN, basado en reloj, siendo el desplazamiento aleatorio elegido por una función criptográfica de "hash", en el identificador de conexión (dos direcciones IP y dos números de puerto), obteniendo que de los 32 bits del ISN, una cantidad de bits superiores son un número de secuencia secreto, y los bits restantes son generados por el hash [1]. Esto produce un ISN que es difícil de imaginar.

3. Establecimiento de una conexión TCP

El establecimiento de la conexión es un procedimiento de 3 pasos, conocido como el "handshake" (saludo) de tres vías. Siendo los extremos de la conexión TCP A y TCP B, en base a [1], para establecer la conexión se emplea tres segmentos:

a) Segmento 1: TCP A envía un segmento SYN, que especifica el número de puerto destino al que se quiere conectar, y su número de secuencia inicial, al que denominaremos ISN(A).

b) Segmento 2: TCP B responde con su propio segmento SYN, que contiene el número de secuencia inicial ISN(B), y el número de reconocimiento ISN(A) +1.

c) Segmento 3: TCP A reconoce el segmento SYN enviado por TCP B, enviando un segmento sin datos con número de reconocimiento ISN(B) +1.

Los tres pasos del "handshake" serían: Transición de un estado de cierre a envío de SYN, transición de envío de SYN a reconocimiento y transición de esto último al establecimiento de la conexión. En la figura A2, se da un ejemplo del procedimiento de 3 pasos para establecer una conexión TCP.

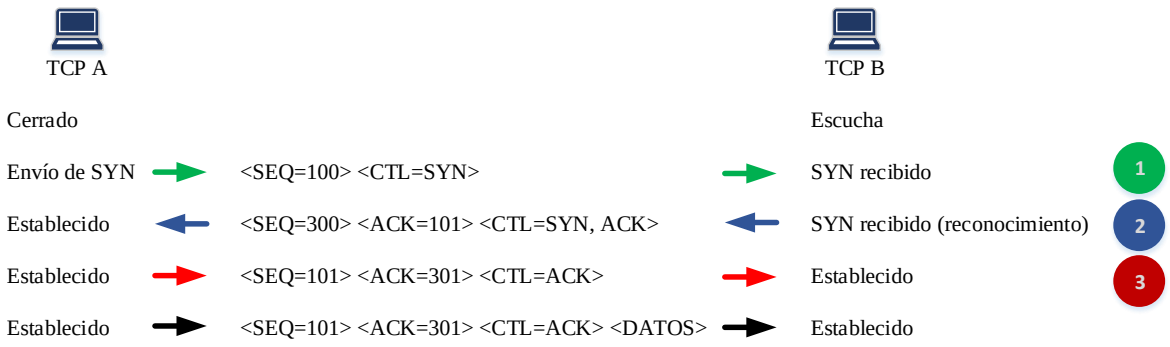


Figura A2. Establecimiento de conexión TCP: Procedimiento de 3 pasos [2].

El procedimiento, también funciona si ambos extremos TCP simultáneamente inician el establecimiento de la conexión . Cada extremo recibe un segmento SYN, que no lleva acuse de

recibo tras haber enviado su segmento inicial, es decir la sincronización es simultánea [2] [1]. La figura A3, ejemplifica y explica este caso.

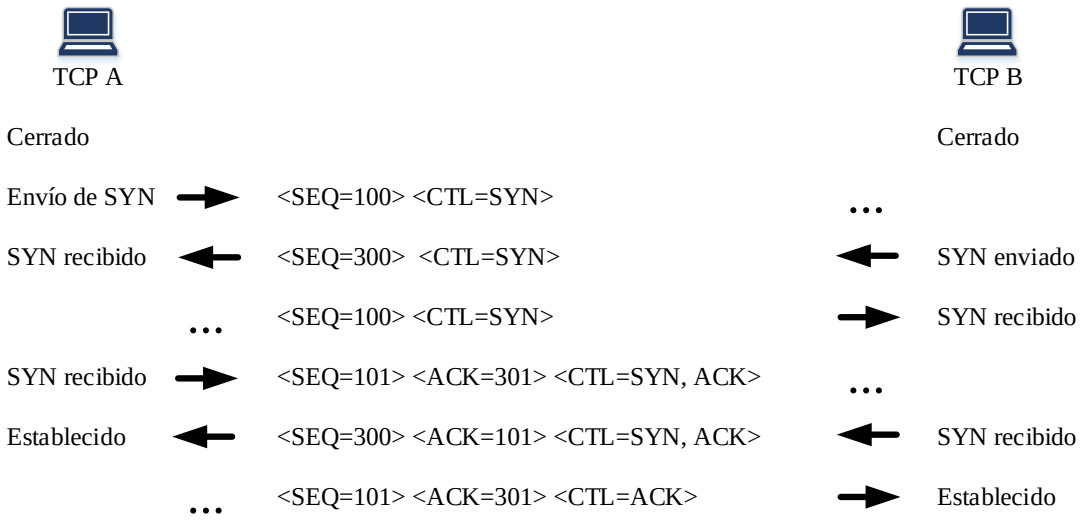


Figura A3. Establecimiento simultáneo de la conexión [2].

Un caso particular se da cuando en una conexión ocurre la llegada de un segmento SYN anterior duplicado: Si el TCP receptor está en un estado no sincronizado,vuelve a su estado de “escucha” tras la recepción de un segmento "RESET" desde el transmisor. Más, si el TCP receptor está en uno de los estados sincronizados (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), corta abruptamente la conexión e informa al usuario [2].

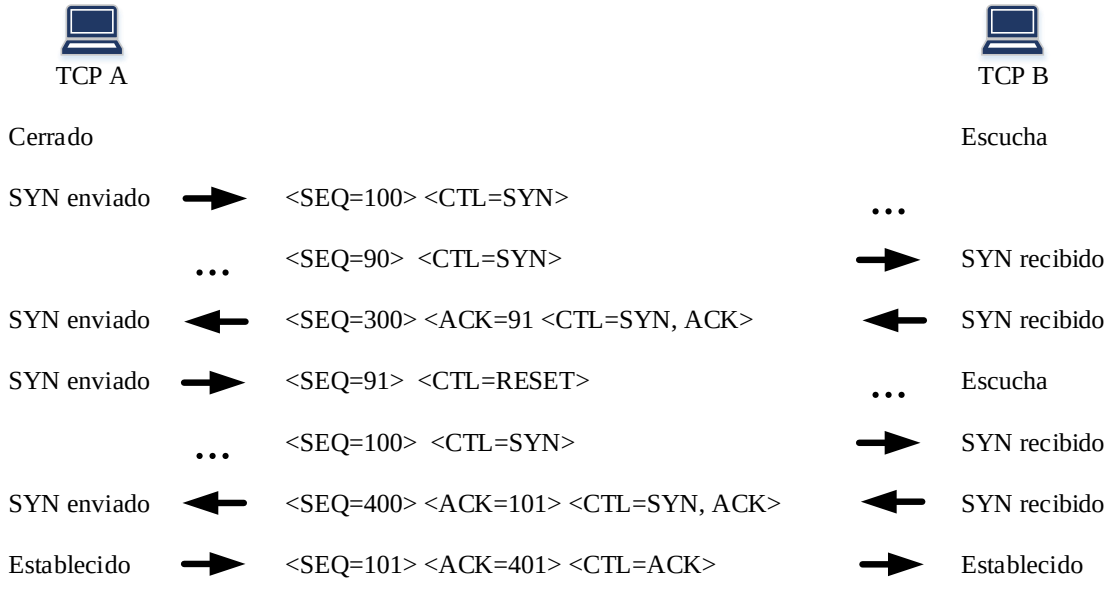


Figura A4. Reinicio ante un SYN duplicado (Uso de bit de control RESET) [2].

En la figura A4, se muestra que TCP A envía un segmento SYN para establecer la conexión, sin embargo TCP B recibe un segmento SYN duplicado anterior, y responde con un segmento de número de ACK no esperado por TCP A, involucrando el uso de "RESET".

El uso del bit RESET también es empleado en conexiones medio-abiertas (half open). Estas conexiones ocurren cuando uno de los extremos de la conexión ha cerrado o ha interrumpido la conexión sin avisar a la otra parte, quedando no sincronizados [2].

4. Opciones de TCP

4.1. Tamaño máximo de segmento (MSS)

Es ubicado en el campo "Opciones" de la cabecera de cada segmento SYN, y es implementado en el par correspondiente al extremo que transmitirá los datos [12] [1] [2]. Si tenemos dos extremos, denominados TCP A y TCP B, la opción MSS es implementada en TCP B, la cual indica la cantidad máxima de bytes de datos por segmento, que está dispuesto a recibir de TCP A.

El campo opciones de TCP, permite emplear 16 bits para un valor de MSS, siendo el valor mínimo de un MSS = 536 bytes, el cual es asumido en caso de no ser especificado el MSS. A partir de ello se tiene un tamaño de datagrama de 576 bytes. Esto es: 20 bytes de cabecera IPv4 + 20 bytes de cabecera TCP + 536 bytes de datos (MSS) [12]. Para el caso de Ethernet, el cual tiene importancia en el presente documento, la unidad máxima de transmisión (MTU: Maximum Transmission Unit) es 1500 bytes, que es el datagrama resultante al sumar un MSS de 1460 bytes, 20 bytes de cabecera TCP y 20 bytes de cabecera IPv4. Cuando se emplea IPv6, el MSS sería de 1460 bytes (Considerando una cabecera IPv6 de 40 bytes) [1].

5. ABC ("Appropiate Byte Counting")

El conteo de bytes apropiado (ABC), es una técnica propuesta en [22], que mejora los algoritmos de "Slow-start" y evitación de la congestión, debido a que el incremento de la ventana de congestión ("cwnd") se realiza cuando se reconoce el número de bytes que han sido reconocidos para cada ACK que llega al transmisor, a diferencia del comportamiento típico, en el cual la cantidad de ACKs recibidos (sin distinguir la cantidad de bytes) basta para realizar el incremento de "cwnd". Con ABC, durante la fase de evitación de la congestión, se emplea una variable "bytes_acked". Cuándo su valor es mayor o igual al valor de "cwnd", será reducido a "bytes_acked - cwnd", y la ventana es incrementado en un valor "L", siendo el valor por defecto de "L" un segmento de tamaño completo de envío ("SMSS": Sender MSS).

Durante la evitación de la congestión, "cwnd" es incrementada en 1 SMSS aproximadamente en por cada RTT [4] [5], sin embargo cuando se emplea el algoritmo "ACK retardado" (*ACK delayed*), "cwnd" se incrementa en 1 SMSS por cada segundo RTT durante la evitación de la congestión. En este último caso, el crecimiento lento sería mitigado al emplear ABC con un valor L = 2 SMSS. Por otro lado si se aplica ABC con L = 2 SMSS, en un entorno donde no existe "ACK retardado", la ventana tendrá un incremento agresivo de 2 segmentos por cada RTT.

Debido a una mayor tendencia de ocurrencia de pérdidas durante la fase de "slow-start", al emplear ABC con L=2SMSS, en [22] se recomienda emplear un valor de L = 1 SMSS, y no emplear "ACK retardado", de tal modo de no emplear un protocolo agresivo que genere una mayor cantidad de pérdidas, aunque este sea compensando con un mejor rendimiento total según entornos simulados.

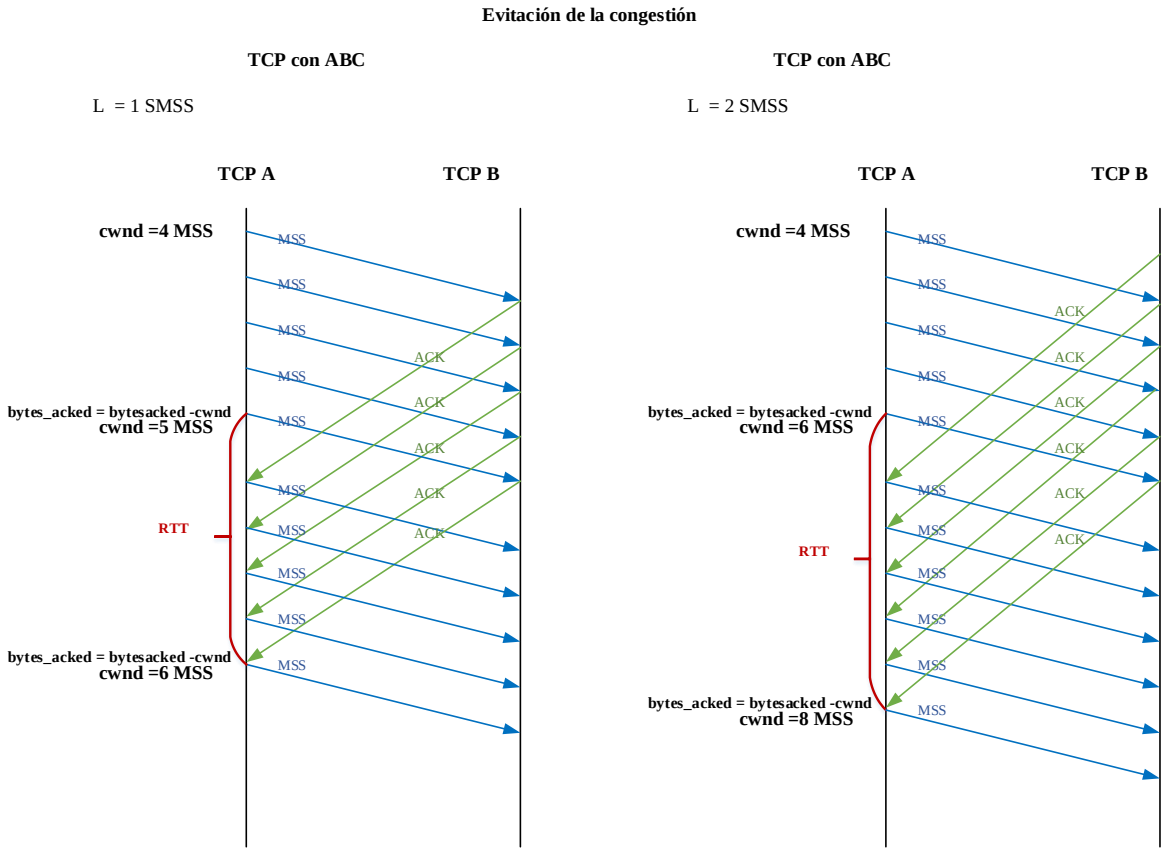


Figura A5. Uso de ABC en la fase de evitación de la congestión de TCP

Si hacemos una comparación simple, asumiendo que no existen pérdidas ni un comportamiento con "ACK retardado", y el receptor emitió los ACK correctamente, se tiene un comportamiento como el mostrado en la figura A4, cuando se emplea ABC con L = 1 SMSS y L = 2 SMSS.

ANEXO "B"

CÓDIGO FUENTE EN LINUX

1. KERNEL Y CÓDIGO FUENTE

El kernel o llamado "núcleo" es el nivel más bajo de software ejecutándose sobre un sistema Linux y determina como el sistema de Linux trabajará, siendo su pieza más confiable [23]. La versión de kernel es obtenida mediante la línea de comando: "**uname –r**" en la consola de un sistema operativo Linux. Ejemplo: 3.13.0-32-generic, 3.2.0-23-generic, etc.

El código fuente es el conjunto de directorios y archivos que contienen el conjunto de instrucciones que emplea el kernel para realizar la ejecución de un sistema de Linux. Para obtener el código fuente es necesario descargarlo de su sitio web. El directorio de descarga por defecto es /usr/src/, entonces: **<Directorio de código fuente>=/usr/src/<linux-X.X.X>.** Por ejemplo: El código fuente es ubicado en /usr/src/linux-3.13.0, cuando el kernel es 3.13.0-32-generic.

2. PROCESO DE INSTALACIÓN Y CONFIGURACIÓN DE CÓDIGO FUENTE.

a) Instalación de cabeceras

Se realiza mediante la línea de comando:

```
sudo apt-get install linux-headers-$(uname –r) linux-libc-dev kernel-package
```

b) Instalación de código fuente

- **Descarga desde el sitio web de kernel:** En el sitio https://www.kernel.org/pub/linux/kernel/, se descarga el archivo de extensión "tar.gz" que corresponda a la versión de kernel de interés. Por ejemplo, para la versión de kernel 3.13.0-32 se descarga el archivo linux-3.0.13.tar.gz. Se recomienda que el archivo sea copiado a la ubicación /usr/src/.
- **Para distribuciones debían UBUNTU y LUBUNTU:** Se descarga el archivo nombrado linux-source-x.x.x_x.x.x-x.x_all.deb correspondiente a la versión de kernel deseado, desde la URL: http://security.ubuntu.com/ubuntu/pool/main/l/linux/. La instalación del código fuente se realiza mediante la línea de comando: dpkg -i <*.deb>. Para un sistema operativo con kernel 3.13.0-32 la línea de comandos es:

```
dpkg –i linux-source-3.13.0_3.13.0-32.57_all.deb
```

Un archivo de extensión tar.bz2 que indica la versión de kernel es creado, y luego se extrae.

c) Preparación del equipo para que opere código fuente.

Se prepara el equipo para operar con módulos a través de la herramienta **module-assistant.** Se descarga y ejecuta escribiendo en consola:

```
sudo apt-get install module-assistant
sudo module-assistant
```

Al ejecutar la herramienta, se selecciona la opción PREPARE, que preparará el equipo para compilar módulos. Esto descargará build-essential, necesario para utilizar los módulos del código fuente.

3. DIRECTORIOS DE INTERÉS EN CODIGO FUENTE.

a) **<Directorio de código fuente>/net/ipv4:** Aloja los archivos de extensión ".c" entre ellos los programas escritos en lenguaje C que permiten ejecutar las variantes de protocolos de transporte TCP: tcp_bic.c, tcp_cubic.c, tcp_highspeed.c, entre otros.

b) **<Directorio de código fuente>/include/linux:** Aquí se encuentran un archivos importante:

- **tcp.h :** Archivo que muestra los nombres de las principales variables que permiten la modificación del comportamiento del protocolo TCP, al ser empleadas en la programación de archivos de extensión ".c" ubicados en <Directorio de código fuente>/net/ipv4.

  Las variables más importantes son aquellas que llaman a la ventana de congestión enviada, "slow-start-threshold" y contador de paquetes.

c) **<Directorio de código fuente>/include/net:** Aquí se encuentran los archivos importantes:

- **tcp.h :** Define las opciones del módulo de TCP, entre ellas: MAX_TCP_HEADER (Máximo tamaño de encabezado TCP), MAX_TCP_WINDOW (Máximo tamaño de ventana TCP que se puede recibir), TCP_INIT_CWND (Ventana inicial de congestión), etc.

  En tcp.h se enumera eventos de control de la congestión, y también define la función **tcp_congestion_ops**, que será explicada en el capítulo siguiente.
- **tcp_states.h:** Enumera estados de TCP.

d) **<Directorio de código fuente>/include/uapi/linux:** Aquí se encuentran el archivo:

- **tcp.h :** Enumera los estados de evitación de la congestión.

ANEXO "C"

OTRAS HERRAMIENTAS PARA EXTRAER DATOS EN UNA CONEXIÓN TCP.

1. TCPTRACE

TCPTRACE es una utilidad que permite el análisis de archivos TCP. Este puede tomar como entrada los archivos producidos por varios programas de captura de paquetes. Esta herramienta puede producir varios diferentes de tipos de información de contenido de salida, tal como cada conexión dada, el tiempo transcurrido de ésta, segmentos enviados y recibidos, retransmisiones, RTT, ventana anunciada, rendimiento y otros [24]. Con "tcptrace" se puede analizar un archivo de captura (de extensión .pcap) obtenido empleando tshark. La herramienta se encuentra disponible para sistemas operativos Linux y Windows en [24].

Por ejemplo al emular una tranmisión que emplea TCP Reno con iPerf con un valor de MSS=2896 Bytes, RTT máximo de 100ms y un porcentaje de pérdidas aleatorias de 1%, siendo el transmisor identificado con dirección IP 192.168.1.2 y el receptor con la dirección IP 192.168.1.12, se obtiene un archivo de captura con tshark llamado "capreno02.cap":

- La sentencia en linux que brinda información resumida de la conexión capturada en el archivo de extensión .cap sería la siguiente:

```
tcptrace capreno02.cap
```

En la siguiente figura se aprecia un ejemplo de la salida de este comando:

```
1 arg remaining, starting with 'capreno02.cap'
Ostermann's tcptrace -- version 6.6.7 -- Thu Nov  4, 2004

7496 packets seen, 7494 TCP packets traced
elapsed wallclock time: 0:00:00.308859, 24269 pkts/sec analyzed
trace file elapsed time: 0:00:59.881419
TCP connection info:
  1: 192.168.5.2:37661 - 192.168.5.12:5001 (a2b) 3484> 4010<
```

Figura C1. Salida empleando el comando tcptrace

Se observa en la figura anterior un resumen de la conexión, siendo el host "a" identificado con dirección IP 192.168.5.2 y el host "b" con dirección IP 192.168.5.112. La cantidad de paquetes transmitidos de "a" hacia "b" es 3484. El puerto TCP destino de la conexión es 5001 (Puerto empleado por iPerf)

- Añadiendo la opción "l" al comando anterior, se obtiene mayor información de la conexión grabada en el archivo de captura: La línea de comando en linux es:

```
tcptrace –l capreno02.cap
```

Mediante la opción “l” del comando tcptrace, se muestra más detalles como: Bytes enviados, máximo tamaño del segmento, valor de ventana inicial, entre otros.

En el ejemplo dado, se transmitieron 9523520 bytes, el valor de MSS es de 2896 bytes, la ventana inicial de envío es 6 paquetes y el rendimiento es de 159040 bytes/segundo (Ver Figura C2).

```
1 arg remaining, starting with 'capreno02.cap'
Ostermann's tcptrace -- version 6.6.7 -- Thu Nov  4, 2004

7496 packets seen, 7494 TCP packets traced
elapsed wallclock time: 0:00:00.056863, 131825 pkts/sec analyzed
trace file elapsed time: 0:00:59.881419
TCP connection info:
1 TCP connection traced:
TCP connection 1:
        host a:        192.168.5.2:37661
        host b:        192.168.5.12:5001
        complete conn: no       (SYNs: 0)  (FINs: 0)
        first packet:  Sat Jan 23 17:36:57.125486 2016
        last packet:   Sat Jan 23 17:37:57.006905 2016
        elapsed time:  0:00:59.881419
        total packets: 7494
        filename:      capreno02.cap
   a->b:                              b->a:
     total packets:          3484           total packets:          4010
     ack pkts sent:          3484           ack pkts sent:          4010
     pure acks sent:            1           pure acks sent:         4010
     sack pkts sent:            0           sack pkts sent:          618
     dsack pkts sent:           0           dsack pkts sent:           0
     max sack blks/ack:         0           max sack blks/ack:         2
     unique bytes sent:   9523520           unique bytes sent:         0
     actual data pkts:       3483           actual data pkts:          0
     actual data bytes:   9523520           actual data bytes:         0
     rexmt data pkts:           0           rexmt data pkts:           0
     rexmt data bytes:          0           rexmt data bytes:          0
     zwnd probe pkts:           0           zwnd probe pkts:           0
     zwnd probe bytes:          0           zwnd probe bytes:          0
     outoforder pkts:          80           outoforder pkts:           0
     pushed data pkts:        126           pushed data pkts:          0
     SYN/FIN pkts sent:       0/0           SYN/FIN pkts sent:       0/0
     req 1323 ws/ts:          N/Y           req 1323 ws/ts:          N/Y
     urgent data pkts:          0 pkts      urgent data pkts:          0 pkts
     urgent data bytes:         0 bytes     urgent data bytes:         0 bytes
     mss requested:             0 bytes     mss requested:             0 bytes
     max segm size:          2896 bytes     max segm size:             0 bytes
     min segm size:            24 bytes     min segm size:             0 bytes
     avg segm size:          2734 bytes     avg segm size:             0 bytes
     max win adv:             115 bytes     max win adv:            4059 bytes
     min win adv:             115 bytes     min win adv:              57 bytes
     zero win adv:              0 times     zero win adv:              0 times
     avg win adv:             115 bytes     avg win adv:            3831 bytes
     initial window:        13056 bytes     initial window:            0 bytes
     initial window:            6 pkts      initial window:            0 pkts
     ttl stream length:        NA           ttl stream length:        NA
     missed data:              NA           missed data:              NA
     truncated data:            0 bytes     truncated data:            0 bytes
     truncated packets:         0 pkts      truncated packets:         0 pkts
     data xmit time:       59.881 secs      data xmit time:        0.000 secs
     idletime max:          101.9 ms        idletime max:          101.8 ms
     throughput:           159040 Bps       throughput:                0 Bps
```

Figura C2. Salida empleando el comando tcptrace con la opción “-l”

- Además al añadir la opción "r" al comando anterior, se obtiene información referente al delay (RTT). La línea de comando en linux es:

```
tcptrace –lr capreno02.cap
```

Mediante el comando dado, se muestra una salida con más detalles como: RTT máximo, ACKs de pérdidas de paquetes, triple ACK duplicados, entre otros.

En el ejemplo dado, el RTT máximo es 100ms, durante la transmisión existen 41 ACKs que indican pérdida, de los cuales 40 fueron triple ACK duplicados (Ver Figura 25).

```
RTT samples:            2804          RTT samples:              0
RTT min:                 0.1 ms       RTT min:                0.0 ms
RTT max:                99.8 ms       RTT max:                0.0 ms
RTT avg:                 0.5 ms       RTT avg:                0.0 ms
RTT stdev:               2.7 ms       RTT stdev:              0.0 ms

RTT from 3WHS:           0.0 ms       RTT from 3WHS:          0.0 ms

RTT full_sz smpls:      2530          RTT full_sz smpls:        0
RTT full_sz min:         0.1 ms       RTT full_sz min:        0.0 ms
RTT full_sz max:        85.0 ms       RTT full_sz max:        0.0 ms
RTT full_sz avg:         0.4 ms       RTT full_sz avg:        0.0 ms
RTT full_sz stdev:       2.1 ms       RTT full_sz stdev:      0.0 ms

post-loss acks:           41          post-loss acks:           0
segs cum acked:          637          segs cum acked:           0
duplicate acks:          577          duplicate acks:           0
triple dupacks:           40          triple dupacks:           0
max # retrans:             0          max # retrans:            0
min retr time:           0.0 ms       min retr time:          0.0 ms
max retr time:           0.0 ms       max retr time:          0.0 ms
avg retr time:           0.0 ms       avg retr time:          0.0 ms
sdv retr time:           0.0 ms       sdv retr time:          0.0 ms
```

Figura C3. Salida empleando el comando tcptrace con la opción "-lr"

2. SOCK

"SOCK" es una herramienta que permite demostrar las propiedades de TCP/IP, disponible para sistema operativo linux en [25]. La utilidad más sencilla de "sock" es establecer una conexión cliente-servidor indicando los números de puertos TCP en ambos extremos, y luego verificando el estado de los puertos con la herramienta "netstat", incluida en sistemas operativos linux. A la vez "sock" muestra el valor de MSS del transmisor (cliente).

Por ejemplo: Para establecer una conexión con sock entre el cliente con dirección IP 192.168.1.37 que emplea un puerto 2091 (Puerto TCP de ejemplo) y el servidor con dirección IP 192.168.1.34, que escucha en el puerto TCP 5001 (Número de puerto empleado por iPerf), se realiza mediante las siguientes líneas de comando:

- En el servidor se ejecuta la línea de comando:

```
sock –b 2091 –v 192.168.1.34 5001
```

- En el cliente se ejecuta la línea de comando:

```
sock –s –v 5001
```

Al emplear “sock”, se muestra una conexión con un MSS de 1448 bytes que emplea el transmisor.

Si se ejecuta la línea de comando “netstat –n –t” en una nueva ventana de consola en el servidor (“sock server”), se mostrará la conexión como “ESTABLECIDO” en el puerto local 5001, desde un puerto TCP 2091 de la dirección IP del cliente.

Si se emplea el mismo comando “netstat –n –t” inmediatamente al momento que se cierra la conexión en la consola, aún se muestra el estado “TIMEWAIT” de la conexión, que es un estado de espera antes de cerrar completamente la conexión.

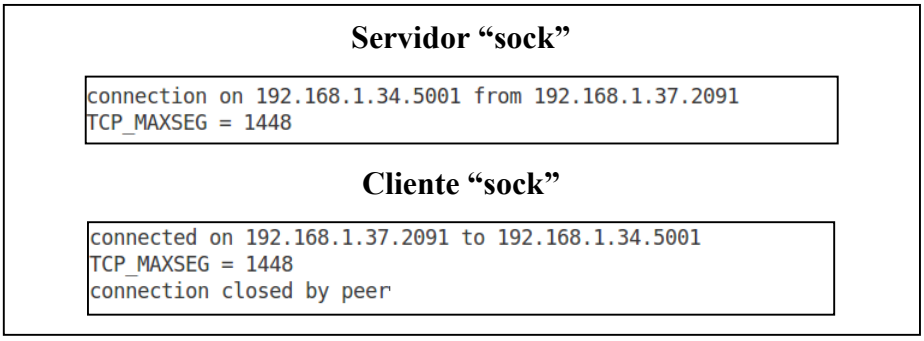


Figura C3. Ejemplo de uso de “sock”

Esta utilidad permite conocer qué puertos TCP están disponibles, al establecerse la conexión, caso contrario mostraría un mensaje indicando la conexión rechazada.

ANEXO "D"

VARIABLES DE TCP

1. DESCRIPCIÓN DE VARIABLES DE TCP

Variable TCP	Valor	Significado [26] [27]
tcp_abort_on_overflow	1	La variable es booleana. El valor "1" se emplea para indicar que si existe sobreflujo de datos debido a una sobreactividad la conexión se aborta.
tcp_adv_win_scale	2	Permite indicar la fracción del espacio de búfer total (aplicación más kernel) a ser empleado para la aplicación en TCP, que incluye el tamaño de la ventana de recepción anunciada al otro extremo. El valor "2" indica que el búfer de aplicación ocupa 1/2 del espacio total.
tcp_app_win	31	Este valor define cuántos bytes de la ventana TCP son reservados para sobrecarga de almacenamiento. El valor por defecto es "31".
tcp_base_mss	512	El MSS mínimo aceptable en el descubrimiento de MTU.
tcp_dsack	1	Variable booleana, que permite a TCP enviar sack's duplicados. Su valor por defecto es 1
tcp_ecn	0	Deshabilita ECN. La función ECN (notificación de congestión implícita) es evitar las pérdidas debido a la congestión, al permitir que los routers de apoyo indiquen la congestión antes de tener que descartar los paquetes.
tcp_fack	0	Se deshabilita la evitación de la congestión de tipo FACK.
tcp_fin_timeout	60	Especifica cuantos segundos esperar por un paquete FIN final antes que el socket en el otro extremo sea forzadamente cerrado. Esto evita ataques de denegación de servicio.
tcp_frto	0	Deshabilita el algoritmo FRTO, cuya función era mejorar la recuperación para "timeouts" de retransmisión en TCP.
tcp_frto_response	0	El valor "0" normalmente indica que la ventana de congestión y "slow-start threshold" son reducidos a la mitad de la ventana de congestión, después de un RTT cuando ocurre un "timeout" de retransmisión.
tcp_keepalive_intvl	75	Es el tiempo entre dos pruebas "keepalive". Una prueba "keepalive" es un mensaje enviado por un dispositivo al otro extremo para comprobar que el enlace está en funcionamiento.
tcp_keepalive_probes	9	Cantidad máxima de mensajes "keepalive" a enviar al otro extremo, si no se tiene una respuesta de éste.
tcp_keepalive_time	7200	Cantidad de segundos que una conexión necesita estar vacía antes que TCP inicie el envío de pruebas "keepalive".
tcp_low_latency	0	Deshabilitar una baja latencia implica obtener un rendimiento más alto.
tcp_max_ssthresh	2147483647	Indica el máximo valor del umbral (Slow-start threshold). *Nota:* En la versión de kernel empleado: 3.13.0-32 no aparece esta opción y su valor es dado por defecto.
tcp_max_orphans	8192	El número máximo de sockets TCP huérfanos (no conectado a ningún identificador de archivo de usuario) permitidos en el sistema.
tcp_max_syn_backlog	1	TCP realiza el autoajuste del búfer de recepción, tratando de dimensionar automáticamente la memoria intermedia (no mayor de tcp_rmem [2° valor]) para que coincida con el tamaño requerido por la ruta para completar el rendimiento.

tcp_men	'65535 1048575 16777215'	Cuando la cantidad de memoria asignada por TCP está debajo de 65535 no ejecuta ningún ajuste. Cuando la cantidad de memoria asignada por TCP supera 1048575, TCP lo regula hasta que sea menor a 65535. La cantidad máxima de memoria asignada por TCP es 16777215. Las cantidades son medibles en unidades páginas de sistema.
tcp_mtu_probing	0	Descubrimiento de MTU deshabilitado.
tcp_no_metrics_save	1	TCP no almacena en la caché métricas sobre las conexiones ya cerradas.
tcp_orphan_retries	0	El número máximo de intentos realizados para probar el otro extremo de una conexión que ha sido cerrado por nuestro lado.
tcp_reordering	3	El número máximo de paquetes que pueden ser reordenados en un flujo de paquetes TCP sin asumir que sea una pérdida de paquete es 3.
tcp_retrans_collapse	0	No se prueba enviar paquetes de tamaño completado durante la retransmisión.
tcp_rmen	'65535 1048575 16777215'	Las cantidades son expresadas en páginas de sistema. 65535 es el tamaño mínimo de búfer de recepción usado por un socket TCP. 1048575 es el tamaño por defecto de búfer de recepción usado por un socket TCP. 16777215 es el tamaño máximo de búfer de recepción usado por un socket TCP.
tcp_sack	1	Habilita SACK
tcp_slow_start_after_idle	1	Ejecuta un slow-start después de un periodo vacío (“timeout” de retransmisión).
tcp_syncookies	0	El valor “0” desactiva el envío de syncookies, que es una violación del protocolo TCP, y los conflictos con otras áreas de la TCP, como extensiones TCP. De ser activado (Valor “1”) puede causar problemas para los clientes y redes, razón por la cual no se recomienda como un mecanismo de ajuste para servidores con mucha carga para ayudar con las condiciones de sobrecarga o mal configurados.
tcp_syn_retries	5	El segmento SYN será retransmitido máximo 5 veces para una conexión TCP activa.
tcp_synack_retries	5	Es el número máximo de veces que un segmento SYN/ACK será retransmitido para una conexión TCP pasiva.
tcp_window_scaling	1	Permite el uso de ventana de paquetes mayor a 64 KB.
tcp_wmem	'65535 1048575 16777215'	Las cantidades son expresadas en páginas de sistema. 65535 es el tamaño mínimo de búfer de envío usado por un socket TCP. 1048575 es el tamaño por defecto de búfer de envío usado por un socket TCP. 16777215 es el tamaño máximo de búfer de envío usado por un socket TCP.
tcp_workaround_signed_windows	0	Deshabilita usar un tamaño de ventana máximo fijo.

Tabla D1. Variables de TCP.

2. SCRIPT EN LINUX PARA CONFIGURACIÓN DE VARIABLES DE TCP

```
# Las opciones de TCP para las variabkes son definidas al ejecutar este script
#! /bin/sh
sudo sysctl -w net.ipv4.tcp_adv_win_scale=2                    #valor por defecto es 1
sudo sysctl -w net.ipv4.tcp_abort_on_overflow=1                #valor por defecto es 0
sudo sysctl -w net.ipv4.tcp_app_win=31                         #valor por defecto
sudo sysctl -w net.ipv4.tcp_base_mss=512                       #valor por defecto
sudo sysctl -w net.ipv4.tcp_dsack=1                            #valor por defecto
sudo sysctl -w net.ipv4.tcp_ecn=0                              #valor por defecto es 2
sudo sysctl -w net.ipv4.tcp_fack=0                             #valor por defecto es 1
sudo sysctl -w net.ipv4.tcp_fin_timeout=60                     #valor por defecto
sudo sysctl -w net.ipv4.tcp_frto=0                             #valor por defecto
sudo sysctl -w net.ipv4.tcp_frto_response=0                    #valor por defecto
sudo sysctl -w net.ipv4.tcp_keepalive_intvl=75                 #valor por defecto
sudo sysctl -w net.ipv4.tcp_keepalive_probes=9                 #valor por defecto
sudo sysctl -w net.ipv4.tcp_keepalive_time=7200                #valor por defecto
sudo sysctl -w net.ipv4.tcp_low_latency=0                      #valor que permite el rendimiento mas alto
sudo sysctl -w net.ipv4.tcp_max_orphans=8192                   #valor por defecto
sudo sysctl -w net.ipv4.tcp_moderate_rcvbuf=1                  #valor por defecto. Permite que iperf
obtenga datos.
sudo sysctl -w net.ipv4.tcp_mtu_probing=0                      #valor por defecto
sudo sysctl -w net.ipv4.tcp_no_metrics_save=1                  #valor que permite la realización de
pruebas.
sudo sysctl -w net.ipv4.tcp_orphan_retries=0                   #valor por defecto es 8
sudo sysctl -w net.ipv4.tcp_reordering=3                       #valor por defecto
sudo sysctl -w net.ipv4.tcp_retrans_collapse=0                 #valor por defecto es 1
sudo sysctl -w net.ipv4.tcp_sack=1                             #valor por defecto
sudo sysctl -w net.ipv4.tcp_slow_start_after_idle=1            #valor por defecto
sudo sysctl -w net.ipv4.tcp_syn_retries=5                      #valor por defecto
sudo sysctl -w net.ipv4.tcp_synack_retries=5                   #valor por defecto
sudo sysctl -w net.ipv4.tcp_syncookies=0                       #valor por defecto es 1
sudo sysctl -w net.ipv4.tcp_timestamps=1                       #valor por defecto
sudo sysctl -w net.ipv4.tcp_window_scaling=1                   #valor por defecto
sudo sysctl -w net.ipv4.tcp_workaround_signed_windows=0        #valor por defecto
#
# Variables TCP relacionadas con búferes
sudo sysctl -w net.ipv4.tcp_mem='65535 1048575 16777215'
sudo sysctl -w net.ipv4.tcp_wmem='65535 1048575 16777215'
sudo sysctl -w net.ipv4.tcp_rmem='65535 1048575 16777215'
```

Figura D1. Archivo script ~/opciones_tcp

ANEXO “E”

PROGRAMA EN MATLAB PARA ANÁLISIS DE DATOS DE UNA CONEXIÓN PUNTO A PUNTO

1. /home/owl-wkstn-tcp/SIMULACION /tcp_analyzer.m

```
%% ANALIZADOR DE COMPORTAMIENTO DE LA VENTANA DE CONGESTION EN PROTOCOLOS DE TRANSPORTE
PARA UNA CONEXIÓN UNICA
% JESUS MARTIN BRAVO SUCLUPE

%% Clean up
close all
clear all
clc

load_prev = input('¿Desea cargar datos utilizados la última vez? s/n [n]: ', 's');
if isempty(load_prev)
    load_prev = 'n';
end

if load_prev == 's'
    fprintf('Cargando datos ... ')
    % Inicia reloj
    tic
    load tcp_data
    % dur_t: cantidad de segundos transcurridos en reloj
    duracion = toc;
    fprintf('Hecho! Duracion: %f segundos\n', duracion)
else

prompt = 'Seleccione el protocolo a cargar: 1 [Reno], 2 [GAIMD], 3 [ESTP], 4 [otro]:';
load_protocol = input(prompt);
protocol = load_protocol;
ESTP = 0;

%% Valor de RTT en segundos
    rtt = 0.10;% RTT en segundos

%% Valor de MSS
    mss = 1448; % MSS en bytes

%% Valor de tau para ESTP
    tau = 50;

%% Obtiene archivos tcpprobe y tshark desde usuario
    [pbfile path] = uigetfile('tcpprobe*.*','Archivo TCPprobe');
    [fshark path] = uigetfile('wire*.*','Archivo shark');

%% Leer archivo tcpprobe
    fprintf('Importando datos desde tcpprobe  ... ')
    tic
    if ischar(pbfile)
        [pbtime_i, pbcwnd_i, ssthresh] = tcpprobe_reader([path, pbfile], rtt);
         %pbtime_i = pbtime_i - min(pbtime_i);
    else
        fprintf('--> Ningún archivo seleccionado\n\n')
    end
    duracion = toc;
    fprintf('Hecho! Duración: %f segundos\n', duracion)

%% Leyendo archivo tshark
    fprintf('Importando datos desde archivo tshark ... ')
    tic
```

```
    if ischar(fshark)
          [time_i, ACK, SEQ, LEN, REORDER, RTX] = shark_reader([path, fshark]);
    else
        fprintf('--> Ningún archivo seleccionado\n\n')
    end

    fprintf('Convirtiendo datos a vectores. Espere por favor ...\n')
    for n=1:50
        fprintf('*')
    end
    fprintf('\n')
    dur_t = toc;
    fprintf('*\nHecho! Duración: %f segundos\n', dur_t)

%% Calculando nuevos vectores de tiempo, cwnd y pendiente(incremento)/ decremento
multiplicativo de TCP Probe

[pbtime, pbcwnd, pbslope, pbdec, pbthrput_total, beta_reno, beta_gaimd] =
tcpprobe_values(pbtime_i, pbcwnd_i, ssthresh, mss, rtt);

pbdec_prom = mean(pbdec(~isnan(pbdec)));

%%  Cálculo de nuevos vectores de tiempo y cwnd de tshark

[time, wscwnd, pktsend, thrput_total, beta_estp] = shark_values(time_i, SEQ, LEN, ACK,
REORDER, RTX, mss, rtt, tau);

%% Seleccionando el valor Beta correcto
% Los valores de beta se obtienen empleando tshark para calcular
% beta_estp y se emplea tcpprobe para calcular beta_reno y beta_gaimd

    if protocol ==1             %Si el protocolo empleado es RENO
        beta = beta_reno;
    else
        if protocol == 2        %Si el protocolo empleado es GAIMD
            beta = beta_gaimd;
        else
            if protocol == 3    %Si el protocolo empleado es ESTP
                beta = beta_estp;
            end
        end
    end

%% Cálculo de rendimiento (throughput)

% Empleando datos de tcp probe
    thrput_pb = zeros(1,length(pbcwnd));

    for i=1:length(pbcwnd)
        thrput_pb(i) = mean(pbcwnd(1:i))*8*mss/rtt; % Rendimiento en bits/s (TCP Probe)
    end
% Empleando datos de Tshark

    thrput = 8*mss*(pktsend./time);

    Ltcp=min(length(time),length(pbtime));
    dur_t = toc;
    fprintf('*\nHecho! DuraciÃ³n: %f segundos\n', dur_t)

    fprintf('Guardando espacio de trabajo ... ')
    tic
    save tcp_data
    dur_t = toc;
    fprintf('Hecho! DuraciÃ³n: %f segundos\n', dur_t)
end
```

```
%% GRAFICANDO EL COMPORTAMIENTO DE LA VENTANA DE CONGESTION, RENDIMIENTO, ALFA Y BETA

%% Inicializar variables Plot
p_loss = 0.001;
b = 1;
TDP = sqrt(2*b/(3*p_loss))*rtt;
plot_range = TDP*5; % 5 Periodos triple ACK duplicado
txt = zeros(1,Ltcp);
slope = zeros(1,Ltcp);
last_SEQ = 0;
last_ACK = 0;
win = 1;
Ecwnd = mean(pbcwnd(1:length(pbcwnd))); % Promedio de la ventana de congestión (TCP
Probe)
x = 1;

%% Inicilizar Area Grafica
h_fig = figure(1);
subplot(5,3,[1,2])
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
set(h_fig, 'Color', [1 1 1])

subplot(5,3,[4,5,7,8])
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
subplot(5,3,[10,13])
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
subplot(5,3,[11,14])
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
subplot(5,3,[12,15])
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
subplot(5,3,[3,6,9])
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
axis([0,10,0,100]) % Rango de ejes para el texto
drawnow

%% Plotear cwnd basado en tcprobe (Superior izquierdo))
h_fig = figure(1);
subplot(5,3,[1,2])
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
fill_cwnd = area(pbtime,pbcwnd);
line_cwnd = plot(pbtime,pbcwnd,'Color', [0 0.3 0]);
set(fill_cwnd, 'EdgeColor' , 'none' , 'FaceColor' , [1 1 1])
axis([-plot_range/2,max(pbtime),0,4*Ecwnd+plot_range/2])
h_left_bar = plot(-plot_range/2*[1,1],[0,4*Ecwnd],'Color', [0.8 0.5 0.5]);
h_right_bar = plot(plot_range/2*[1,1],[0,4*Ecwnd],'Color', [0.8 0.5 0.5]);
h = title('Eje X: Tiempo (s) - Eje Y: Ventana de congestión (Paquetes)');
set(h, 'Color','k')

%% Plotear version zoom de cwnd basado en tcpprobe (Lado intermedio izquierdo: Linea
verde)
subplot(5,3,[4,5,7,8])
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
xmax = plot_range/2;
xmin = -plot_range/2;
ymin = 0;
ymax = max(pbcwnd);
```

```
axis([xmin,xmax,ymin,ymax])
window = 0.1; % Ventana
margin = 0; % Margen de ploteo in seconds
fill_cwnd = area(pbtime,pbcwnd);
set(fill_cwnd, 'EdgeColor' , 'none' , 'FaceColor' , [0.9 0.9 0.9])
line_cwnd = plot(pbtime,pbcwnd,'Color', [0 0.3 0]);
line_cwnd_ws = plot(0,0,'Color', [0 0 1], 'LineWidth', 2);
pbcwnd_circle = plot(0,0,'oc');

%% Plotear slope (pendiente) basado en tcpprobe (Lado izquierdo inferior)
% Cuando cwnd > ssthresh: Incremento de ventana
subplot(5,3,[10,13])
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
h_slope = plot(0,0,'x', 'Color', [0.7 0.1 0.1]);
h_marker = plot(0,0, 'Color', [0.8 0.5 0.5]);
xlabel('Tiempo(s)','Color','k')
ylabel('Incremento','Color','k')

%% Plotear throughput based on wireshark y tcpprobe (Lado inferior central)
subplot(5,3,[12,15])
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
h_thrput = plot(0,0,'Color', [0 0.2 1]);
h_thrput_pb = plot(pbtime,thrput_pb,'m');
xlabel('Tiempo(s)','Color','k')
ylabel('Rendimiento (bits/s)','Color','k')
hleg1 = legend('Datos de Tshark','Datos de TCPprobe');
set(hleg1,'Color', [0 0 0], 'Location','NorthWest', 'TextColor', 'k', 'Box', 'off')

%% Plotear decremento multiplicativo de la ventana de congestion.
subplot(5,3,[11,14])
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
h_decrease = plot(0,0,'*', 'Color', [0 0.2 1]);
h_beta = plot(0,0,'rx');
xlabel('Tiempo(s)','Color','k')
ylabel('Beta','Color','k')
hleg_beta = legend('Decremento real','Beta');
set(hleg_beta,'Color', [0 0 0], 'Location','SouthWest', 'TextColor', 'k', 'Box', 'off')

%% Mostrar información de la transmisión (Lado derecho)
subplot(5,3,[3,6,9])
set(gca, 'Color', [1 1 1], 'XColor', [1 1 1], 'YColor', [1 1 1])
set(gcf, 'Color', [1 1 1])
axis([0,10,0,100])

      txta1 = text(1,106,'-');
      set(txta1, 'Color', [0 0.3 0], 'FontSize', 24)
      txta2 = text(2.5,105,'Ventana de congestion (TCPProbe)');
      set(txta2, 'Color', [0 0 0], 'FontSize', 9)

      txtb1 = text(1,101,'-');
      set(txtb1, 'Color', [0 0 1], 'FontSize', 24, 'FontWeight', 'bold')
      txtb2 = text(2.5,100,'Aproximacion de ventana (Tshark)');
      set(txtb2, 'Color', [0 0 0], 'FontSize', 9)

      txt1a = text(1,90,'Maximo tamano de segmento (bytes):');
      set(txt1a, 'Color', [0 0 0], 'FontSize', 9)
      txt1b = text(8,90,num2str(mss));
      set(txt1b, 'Color', [0 0 0], 'FontSize', 9)          %Muestra tamaño de segmento
      txt1c = text(1,85,'RTT (segundos):');
      set(txt1c, 'Color', [0 0 0], 'FontSize', 9)
      txt1d = text(8,85,num2str(rtt));                      % RTT en segundos
      set(txt1d, 'Color', [0 0 0], 'FontSize', 9)
```

```
      txt1e0 = text(1,75,'Datos de TCPPROBE');
      set(txt1e0, 'Color', [0 0 0], 'FontSize', 9, 'FontWeight', 'bold')

      txt1e = text(1,70,'Ventana de congestion [Unidades: MSS]:');
      set(txt1e, 'Color', [0 0 0], 'FontSize', 9)
      txt1f = text(1,65,'Ventana de congestion promedio:');
      set(txt1f, 'Color', [0 0 0], 'FontSize', 9)

      txt1 = text(1,50,'Rendimiento (TSHARK): T1 [Mb/s]');
      set(txt1, 'Color' , [0 0 0] , 'FontSize' , 9)
      txt2 = text(1,45,'T1 = Transmitidos[Mb]/Tiempo-transcurrido[s]');
      set(txt2, 'Color' , [0 0 0] , 'FontSize' , 9)
      txt3 = text(1,30,'Rendimiento (TCPPROBE): T2 [Mb/s]');
      set(txt3, 'Color' , [0 0 0] , 'FontSize' , 9)
      txt4 = text(1,25,'T2=Ventana-promedio*MSS[Mb]/RTT[s]');
      set(txt4, 'Color' , [0 0 0] , 'FontSize' , 9)

      if (protocol==1 || protocol==2)||(protocol == 3)
            txt_beta = text(1,10,'Beta (PROMEDIO):');
            set(txt_beta,'Color', [0 0 0], 'FontSize', 9)
      end

      txt_decrease = text(1,5,'Decremento real (PROMEDIO):');
      set(txt_decrease,'Color', [0 0 0], 'FontSize', 9)
% drawnow

mult=200;
k=0;
q=0;
for n=1:Ltcp
    %% Mostrar información de cwnd y throughput (rendimiento).  (Lado derecho de la
figura)

      subplot(5,3,[3,6,9])

      if ((n == mult*q+1) || (n == Ltcp))
        info_cwnd = num2str(pbcwnd(n));
        txt_cwnd = text(9,70,info_cwnd);
        set(txt_cwnd, 'Color' , [0 0 0] , 'FontSize' , 10)     %Muestra la ventana de
congestión activa cada 200 muestras
        q = q+1;
      end

      if (n == Ltcp)
            info_cwndprom = num2str(mean(pbcwnd));
            txt_cwndprom = text(9,65,info_cwndprom);
            set(txt_cwndprom, 'Color' , [0 0 0] , 'FontSize' , 10, 'FontWeight',
'bold')      % Ventana de congestión promedio
      end

      if (n == mult*k+1)
        info1 = num2str((thrput(n)/1000000));
        info2 = num2str((thrput_pb(n)/1000000));
        txt5 = text(3,40,info1);
        set(txt5, 'Color' , [0 0 0] , 'FontSize' , 10, 'FontWeight', 'bold')
%Muestra rendimiento logrado hasta el tiempo "n" (tshark)
        txt6 = text(3,20,info2);
        set(txt6, 'Color' , [0 0 0] , 'FontSize' , 10, 'FontWeight', 'bold')
%Muestra rendimiento logrado hasta el tiempo "n" (tcpprobe)
        k = k+1;
      end

      if (n == Ltcp)
        info1 = num2str((thrput_total/1000000));
```

```
        info2 = num2str((pbthrput_total/1000000));
        beta1 = num2str(mean(beta(~isnan(beta))));
        beta2 = num2str(mode(beta(~isnan(beta))));
        txt5 = text(3,40,info1);
        set(txt5, 'Color' , [0 0 0] , 'FontSize' , 10, 'FontWeight', 'bold')
%Muestra rendimiento logrado hasta el tiempo "n" (tshark)
        txt6 = text(3,20,info2);
        set(txt6, 'Color' , [0 0 0] , 'FontSize' , 10, 'FontWeight', 'bold')
%Muestra rendimiento logrado hasta el tiempo "n" (tcpprobe)
            if (protocol == 3)
                txt7 = text(8,10,beta1);
                set(txt7, 'Color' , [0 0 0] , 'FontSize' , 10, 'FontWeight', 'bold')
%Beta (promedio)
            end

            if (protocol==1 ||protocol == 2)
                txt7 = text(8,10,beta2);
                set(txt7, 'Color' , [0 0 0] , 'FontSize' , 10, 'FontWeight', 'bold')
%Beta (promedio)
            end

        info_decrease = num2str(pbdec_prom);
        txt_decrease = text(8,5,info_decrease);
        set(txt_decrease, 'Color' , [0 0 0] , 'FontSize' , 10, 'FontWeight', 'bold')
%Decremento real (promedio)
     end

      if ((n == mult*k) || (n == Ltcp-1))
        pause(1)
        delete(txt5)    %Limpia el valor de rendimiento (tshark)
        delete(txt6)    %Limpia el valor de rendimiento (tcpprobe)
        delete(txt_cwnd)%Limpia el valor cwnd para que se actualice cada 200 muestras
      end

    %% Plotear cwnd de tcpprobe y Tshark
    subplot(5,3,[4,5,7,8])
    circle_y = interp1(pbtime, pbcwnd, time(n));
    near_i = stable_time(time(n), pbtime, 'time');
    near_y = pbcwnd(near_i);
    near_x = pbtime(near_i);

    %set(pbcwnd_circle, 'XData' , time(n), 'YData' , circle_y)
    set(pbcwnd_circle, 'XData' , time(n), 'YData' , wscwnd(n))

    %set(line_cwnd_ws, 'XData', time(1:n) , 'YData' , wscwnd(1:n))
    set(line_cwnd_ws, 'XData', time(1:n) , 'YData' , wscwnd(1:n))

    % tcpprobe
    xmax = time(n) + plot_range/2;
    xmin = time(n) - plot_range/2;
    ymin = 0;
    ymax_lb = stable_time(mean([time(n),xmin]),pbtime,'time');
    ymax_ub = stable_time(mean([time(n),xmax]),pbtime,'time');
    if isempty(ymax_lb)
        ymax_lb = last_ymax_lb;
    else
        last_ymax_lb = ymax_lb;
    end
    if isempty(ymax_ub)
        ymax_ub = last_ymax_ub;
    else
        last_ymax_ub = ymax_ub;
    end
    ymax = max(pbcwnd(ymax_lb:ymax_ub))*1.5;
    axis([xmin-1,xmax,ymin,ymax])
```

```
    subplot(5,3,[1,2])
    set(h_left_bar, 'XData' , xmin*[1,1])
    set(h_right_bar, 'XData' , xmax*[1,1])

    subplot(5,3,[10,13])
    set(h_slope, 'XData' , time(1:n), 'YData' , pbslope(1:n))
    set(h_marker, 'XData' , [time(1),time(n)], 'YData' , [1,1])
    axis([0,max(time(n),rtt),0,4])

    if (n == Ltcp)

        txt_increase = text(8,3.5,'Incremento (ALFA)=');
        set(txt_increase,'Color', 'k', 'FontSize', 10)

        alfa = num2str(mean(pbslope(~isnan(pbslope))));
        txt_alfa = text(35,3.5,alfa);
        set(txt_alfa, 'Color' , 'k' , 'FontSize' , 10, 'FontWeight', 'bold')
    end

    subplot(5,3,[12,15])
    set(h_thrput, 'XData' , time(1:n), 'YData' , thrput(1:n) )
    axis([0,max(time(n),rtt),0,max(mean(thrput(1:n))*10,1)])

    subplot(5,3,[11,14])
    axis([0,max(time(n),rtt),0,1])
    set(h_decrease,'XData',time(1:n),'YData',pbdec(1:n))

    if (protocol == 1 || protocol == 2) || (protocol == 3)
        set(h_beta,'XData',time(1:n),'YData',beta(1:n))
    end

    drawnow
end
```

2. /home/owl-wkstn-tcp/SIMULACION /tcpprobe_reader

```
function [time, cwnd, ssthresh] = tcpprobe_reader(fname, rtt)

fileID = fopen(fname);
C = textscan(fileID, '%s', 'delimiter','\n');
fclose(fileID);

L = length(C{1});
time = zeros(1,L);
cwnd = zeros(1,L);
ssthresh = zeros(1,L);


for n=1:L
    tmp = cell2mat(C{1}(n));
    blnk = strfind(tmp,' ');
    time(n) = str2double(tmp(1:blnk(1)-1));
    cwnd(n) = str2double(tmp(blnk(6)+1:blnk(7)-1));
    ssthresh(n) = str2double(tmp(blnk(7)+1:blnk(8)-1));
end

time= time - min(time)+rtt;

ssthresh_limit = 500;
ssthresh = ssthresh.*(ssthresh < ssthresh_limit) + ssthresh_limit*(ssthresh >=
ssthresh_limit);
```

3. /home/owl-wkstn-tcp/SIMULACION/shark_reader

```
function [TIME, ACK, SEQ, LEN, REORDER, RTX ] = shark_reader(fname)

%fprintf('Importando datos desde archivo Tshark... ');
fileID = fopen(fname);
C = textscan(fileID, '%s', 'delimiter','\n');
fclose(fileID);
% fprintf('done!\n\n');

L = length(C{1});
TIME = zeros(1,L);    %vector de tiempo
ACK = zeros(1,L);       %vector de ACK
SEQ = zeros(1,L);       %vector de número de secuencia
LEN = zeros(1,L);       %vector que indica la longitud de segmento
REORDER = zeros(1,L);   %vector que indica paquetes capturados fuera de orden
(probablemente paquetes retransmitidos: se asume que son paquetes retransmitidos)
RTX = zeros(1,L);
m = 1;


    for n=1:L
        %% Inicializar variables
        p = n/L; % Porcentaje transcurrido
        tmp = cell2mat(C{1}(n)); % convirtiendo datos a un "array" temporal
        isTCP = ~isempty(strfind(tmp,'TCP')); % Busca si contiene la palabra TCP
(Filtra datos TCP). Si contiene TCP es 1, caso contrario es 0.
        %strfind ofrece la posiciÃ³n de la palabra buscada.
        blnk = strfind(tmp,' '); % Busca las posiciones donde existen espacios en
blanco
        isREORDER = ~isempty(strfind(tmp,'Out-Of-Order'));  %Busca segmentos marcados
como capturados fuera de orden (Asumiremos éstos son segmentos retransmitidos)
        isRETRANS = ~isempty(strfind(tmp,'TCP Retransmission'));
        isFAST = ~isempty(strfind(tmp,'Fast Retransmission'));

        %% Filtrado de segmentos TCP
        if isTCP
            punto = min(strfind(tmp,'.'));
            time_blnk = find(blnk < punto, 1, 'last');  %Busca la n-esima posicion en
blanco menor que la posicion del primer punto
            pos_i = blnk(time_blnk)+1;                  %Posicion inicial anterior a
punto (no en blanco)
            pos_f = blnk(time_blnk+1)-1;                %Posicion no en blanco
posterior a punto
            TIME(n) = str2double(tmp(pos_i:pos_f));     %Convierte a un valor "double"

            %% Información específica de TCP (Seq, Ack, Len)
            pos_Ack = strfind(tmp,'Ack=');              %Posicion donde inicia la
palabra "Ack="
            if isempty(pos_Ack)
                continue
            else
                ACK_blnk = find(blnk<pos_Ack,1,'last'); %Busca la ultima posicion de
blnk cuyo valor sea menor a pos_Ack
                SEQ(n) = str2double(tmp(blnk(ACK_blnk-1)+5:blnk(ACK_blnk)-1));
                ACK(n) = str2double(tmp(blnk(ACK_blnk)+5:blnk(ACK_blnk+1)-1));
                LEN(n) = str2double(tmp(blnk(ACK_blnk+2)+5:blnk(ACK_blnk+3)-1));

                if isREORDER
                    REORDER(n) =1;
                end
                if isRETRANS
                    RTX(n) = 1;
```

```
                end
                if isFAST
                    RTX(n) = 1;
                end

                %% Contador de segmento
                m=m+1;
            end
        end
    end
```

4. /home/owl-wkstn-tcp/SIMULACION /stable_time

```
function index = stable_time(target, time_data_stream, type)
if strcmp(type,'percent')
    target_time = max(time_data_stream)*target; % window in seconds
elseif strcmp(type,'time')
    target_time = target;
end

index = find(time_data_stream >= target_time, 1); % index corresponding to upper bound
of window
```

5. /home/owl-wkstn-tcp/SIMULACION/tcpprobe_values

```
function [time, cwnd, slope, decrease, thrput, beta_reno, beta_gaimd] =
tcpprobe_values(time_i, cwnd_i, ssthresh, mss, step)

     L=length(time_i);      % Longitud de vector inicial de tiempo

    % Cálculo de vector inicial de valores de pendiente
    % Cuando ventana de congestión > ssthresh (incremento aditivo)

    slope_i = NaN(1,L);     % Será el vector inicial de valores de pendiente
    d = NaN(1,L);
    d(1)=1;
    idx = 1;
    for i=1:L-1
        if (cwnd_i(i) ~= cwnd_i(i+1))
            if (cwnd_i(i) < cwnd_i(i+1))
                if (cwnd_i(i) > ssthresh(i) && cwnd_i(i)>1)
                    d(idx+1:i) = i-idx;       % Vector de divisor

                end
            end
            idx = i;
        end
    end

    for k=2:L
        if d(k)~=0
            slope_i(k)=cwnd_i(k)/d(k); % Vector inicial que comprende los valores de
pendiente.
        end
    end

     % Calculo de vector inicial de valores de decremento real = ventana
     % siguiente a la congestion/ventana cuando ocurre la congestión.
```

```
dec_i = zeros(1,L);
beta_reno_i = zeros(1,L);
beta_gaimd_i = zeros(1,L);
timedec_i = zeros(1,L);
i = 2;
c = 1;
for q=2:L
    if q == i
        if (cwnd_i(q) < cwnd_i(q-1)) && (cwnd_i(q) >= ssthresh(q))&&cwnd_i(q)>1
            n = q-1;
            reduce = ssthresh(q);
            while ((cwnd_i(n) ~= reduce) && n<L)
                    n = n+1;
            end

            dec1 = reduce/cwnd_i(q-1);
            timedec_i(c) = time_i(q-1);
            dec_i(c) = dec1;
            dec2 = reduce/(2*round(cwnd_i(q-1)/2));
            beta_reno_i(c) = dec2;
            dec3 = reduce/(8*round(7*cwnd_i(q-1)/8)/7);
            beta_gaimd_i(c) = dec3;
            c = c+1;
            i = n+1;
             if i <= q || i > L
                 i = q+1;
             end
        else
            i = q+1;
        end
    end

end


% Cálculo de valores aproximadamente cada rtt

time_pr=zeros(1,L);    % Vector temporal de tiempo
cwnd_pr=zeros(1,L);    % Vector temporal de cwnd
slope_pr=zeros(1,L);   % Vector temporal de pendiente
time_pr(1)=time_i(1);
cwnd_pr(1)=cwnd_i(1);
slope_pr(1)=slope_i(1);

hop=0;
j=0;
for i=1:L
    if time_i(i)>hop
           j=j+1;
           cwnd_pr(j)=cwnd_i(i);
           time_pr(j)=time_i(i);
           slope_pr(j)=slope_i(i);
           hop = hop + step;
    end
end

time=zeros(1,j);
cwnd=zeros(1,j);
slope=zeros(1,j);

for i=1:j
time(i)=time_pr(i);     % Vector de tiempo para graficar
cwnd(i)=cwnd_pr(i);     % Vector de cwnd para graficar
slope(i)=slope_pr(i);   % Vector de slope para graficar
```

```
    end
    thrput = mean(cwnd)*8*mss/step; % Tasa de transferencia aproximada

    % Filtrado de valores de beta (decremento multiplicativo)

    dec_pr = dec_i(dec_i~=0);
    beta_reno_pr = beta_reno_i(beta_reno_i~=0);
    beta_gaimd_pr = beta_gaimd_i(beta_gaimd_i~=0);
    timedec_pr = timedec_i(dec_i~=0);
    decrease=NaN(1,j);
    beta_reno=NaN(1,j);
    beta_gaimd=NaN(1,j);
    k=1;
    for i=1:j
        if k<=length(timedec_pr)
            if time(i)>=timedec_pr(k)
               d1 = dec_pr(k);
               decrease(i)=d1;
               d2 = beta_reno_pr(k);
               beta_reno(i)=d2;
               d3 = beta_gaimd_pr(k);
               beta_gaimd(i) = d3;
               k = k+1;
            end
        end
    end
```

6. /home/owl-wkstn-tcp/SIMULACION/shark_values

```
function [time, wscwnd, pktsend, thrput, beta_estp] = shark_values(time_i, SEQ, LEN,
ACK, REORDER, RTX, MSS, step, tau)

%% Nuevos vectores
    SEQtx = SEQ(LEN~=0);          % Números de secuencia de segmentos enviados
    REORDER_i = REORDER(LEN~=0); % Posiciones de los números de secuencia enviados
etiquetados con "OUT-OF-ORDER"
    RTX_i = RTX(LEN~=0);          % Posiciones de los números de secuencia enviados
etiquetados con "FAST-RETRANSMIT" o "TCP RETRANSMIT"
    time_itx = time_i(LEN~=0);    % Tiempo en que cada segmento es enviado
    time_itx=time_itx-time_itx(1)+step;
    Ltcp = length(time_itx);

% Paquetes perdidos.
ACK_RX = ACK(LEN==0);
ACK_3DUP_I = zeros(1,length(ACK_RX));
order = 1;

for m=1:length(ACK_RX)
        repit = length(find(ACK_RX==ACK_RX(m)));
        if repit>=3
            ACK_3DUP_I(order)=ACK_RX(m);
        order = order+1;
        end
end

ACK_3DUP=unique(ACK_3DUP_I(ACK_3DUP_I~=0));

% Ubicación de posiciones del vector SEQtx donde se retransmitió un
% paquete perdido por 3DUP-ACK

RTX_3DUP = zeros(1,length(SEQ(LEN~=0)));
```

```
for m=1:length(ACK_3DUP)
        pos_seq =find(SEQ(LEN~=0)==ACK_3DUP(m),1,'last');
        if ~isempty(pos_seq)==1
            RTX_3DUP(pos_seq) = 1;
        end
end

RTX_TOTAL = or(or(RTX_3DUP,RTX_i),REORDER_i); %Vector que indica las posiciones de
paquetes retransmitidos = 1

%Vectores que indican la cantidad de paquetes transmitidos

    pkt_i = ((SEQtx - SEQtx(1))/MSS)+1; % Número de paquete
    pktsend_i = pkt_i;                  % Cantidad de paquetes enviados (No discrimina
si el paquete es retransmitido)

    for n=2:Ltcp
        if RTX_TOTAL(n) ==1
            pktsend_i(n) = pktsend_i(n-1)+1;
            pktsend_i(n+1:Ltcp)=pktsend_i(n+1:Ltcp)+1;
        end
    end

    send_loss_i = zeros(1,Ltcp);

    for z=1:Ltcp
        send_loss_i(z) = pktsend_i(z)-sum(RTX_TOTAL(1:z));
    end

    packetsend = pktsend_i(Ltcp);    %   Paquetes enviados
    timetotal = max(time_i);         %   Tiempo total de transmisión

    thrput = (packetsend)*MSS*8/timetotal;

%%   Para graficar
    ws_n=length(time_itx);
    wstime_pr=zeros(1,ws_n);
    pktsend_pr=zeros(1,ws_n);
    send_loss_pr=zeros(1,ws_n);
    pkt_pr=zeros(1,ws_n);
    wstime_pr(1)=time_itx(1);

    hop_ws=0;
    z=0;
    for i=1:ws_n
        if time_itx(i)>hop_ws
               z=z+1;
               wstime_pr(z)=time_itx(i);
               pktsend_pr(z)=pktsend_i(i);
               pkt_pr(z)=pkt_i(i);
               send_loss_pr(z)=send_loss_i(i);
               hop_ws = hop_ws + step;
        end
    end

time=zeros(1,z);
wscwnd=zeros(1,z);
pkt=zeros(1,z);
pktsend=zeros(1,z);
send_loss=zeros(1,z);

for i=1:z
    time(i)=wstime_pr(i);           % Vector de tiempo para graficar (Fuente: tshark)
    pkt(i)=pkt_pr(i);               % Vector que indica el orden de paquetes enviados
    pktsend(i)=pktsend_pr(i);
```

```
    send_loss(i)=send_loss_pr(i);    % Vector de paquetes transmitidos correctamente:
enviados - perdidos
end

for i=2:z
    wscwnd(i)=pktsend(i)-pktsend(i-1);  % Vector de cwnd para graficar (Fuente: wscwnd)
end

%% Cálculo de BETA según ESTP

SEQtx = SEQ(LEN~=0);
time_itx = time_i(LEN~=0);

    ISN = min(SEQ(LEN~=0));
    SEQ_loss_i=zeros(1,length(SEQtx));
    time_loss_i=zeros(1,length(SEQtx));
    c=1;
    for m=1:length(ACK_3DUP)
        pos=find(SEQ(LEN~=0)==ACK_3DUP(m),1,'last');
        if ~isempty(pos)==1
            p=pos;
            SEQ_loss_i(c) = SEQtx(p);
            time_loss_i(c) = time_itx(p);
        c=c+1;
        end
    end
    time_loss=time_loss_i(SEQ_loss_i~=0);
    SEQ_loss=SEQ_loss_i(SEQ_loss_i~=0);
    L = length(time_loss);
    delta=zeros(1,L);
    delta(1)=1+(SEQ_loss(1)-ISN)/1448;
    delta(2:L)=(SEQ_loss(2:L)-SEQ_loss(1:L-1))./1448 + 1;
    beta_loss=(exp(-(delta-1)/tau)+1).^-1;

%Vector de beta(estp) para graficar.
%beta_estp es igual a 1, si no ocurre pérdida
    beta_estp=NaN(1,length(time));
    k=1;
    for i=1:length(time)
        if k<=length(time_loss)
            if time(i)>=time_loss(k)
               d = beta_loss(k);
               beta_estp(i)=d;
               k = k+1;
            end
        end
    end
```

ANEXO “F”

PROGRAMA EN MATLAB PARA ANÁLISIS DE DATOS DE VARIAS CONEXIONES QUE EMPLEAN LOS MISMOS PARÁMETROS

```
%% Clean up
close all
clear all
clc
load_prev = input('Desea cargar datos utilizados la ultima vez? s/n [n]: ', 's');
if isempty(load_prev)
    load_prev = 'n';
end

if load_prev == 's'
    fprintf('Cargando datos ... ')
    % Inicia reloj
    tic
    load tcp_data
    % dur_t: cantidad de segundos transcurridos en reloj
    duracion = toc;
    fprintf('Hecho! Duracion: %f segundos\n', duracion)
else

prompt = 'Seleccione el protocolo a cargar: 1 [Reno], 2 [GAIMD], 3 [ESTP (tau fijo)], 4
[genérico]:';
load_protocol = input(prompt);
protocol = load_protocol;

    %% Valor de RTT en segundos
    rtt = 0.05;% RTT en segundos

    %% Valor de MSS en bytes
    mss = 1448;

    %% Valor de TAU
    tau = 50;

    %% Obtiene archivos tcpprobe
    [pbfile1 path] = uigetfile('tcpprobe*.*','Archivo TCPprobe 1');
    [pbfile2 path] = uigetfile('tcpprobe*.*','Archivo TCPprobe 2');
    [pbfile3 path] = uigetfile('tcpprobe*.*','Archivo TCPprobe 3');
    [pbfile4 path] = uigetfile('tcpprobe*.*','Archivo TCPprobe 4');
    [pbfile5 path] = uigetfile('tcpprobe*.*','Archivo TCPprobe 5');
    [pbfile6 path] = uigetfile('tcpprobe*.*','Archivo TCPprobe 6');
    [pbfile7 path] = uigetfile('tcpprobe*.*','Archivo TCPprobe 7');
    [pbfile8 path] = uigetfile('tcpprobe*.*','Archivo TCPprobe 8');
    [pbfile9 path] = uigetfile('tcpprobe*.*','Archivo TCPprobe 9');
    [pbfile10 path] = uigetfile('tcpprobe*.*','Archivo TCPprobe 10');


    %% Obtiene archivos de TSHARK
    [fshark1 path] = uigetfile('wire*.*','Archivo shark 1');
    [fshark2 path] = uigetfile('wire*.*','Archivo shark 2');
    [fshark3 path] = uigetfile('wire*.*','Archivo shark 3');
    [fshark4 path] = uigetfile('wire*.*','Archivo shark 4');
    [fshark5 path] = uigetfile('wire*.*','Archivo shark 5');
    [fshark6 path] = uigetfile('wire*.*','Archivo shark 6');
```

```
[fshark7 path] = uigetfile('wire*.*','Archivo shark 7');
[fshark8 path] = uigetfile('wire*.*','Archivo shark 8');
[fshark9 path] = uigetfile('wire*.*','Archivo shark 9');
[fshark10 path] = uigetfile('wire*.*','Archivo shark 10');

%% Leer archivos tcpprobe
fprintf('Importando datos desde tcpprobe  ... ')
tic
if ischar(pbfile1)
    [pbtime_i1, pbcwnd_i1, ssthresh1] = tcpprobe_reader([path, pbfile1], rtt);
    %pbtime_i1 = pbtime_i1 - min(pbtime_i1);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
if ischar(pbfile2)
    [pbtime_i2, pbcwnd_i2, ssthresh2] = tcpprobe_reader([path, pbfile2], rtt);
    %pbtime_i2 = pbtime_i2 - min(pbtime_i2);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
if ischar(pbfile3)
    [pbtime_i3, pbcwnd_i3, ssthresh3] = tcpprobe_reader([path, pbfile3], rtt);
    %pbtime_i3 = pbtime_i3 - min(pbtime_i3);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
if ischar(pbfile4)
    [pbtime_i4, pbcwnd_i4, ssthresh4] = tcpprobe_reader([path, pbfile4], rtt);
    %pbtime_i4 = pbtime_i4 - min(pbtime_i4);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
if ischar(pbfile5)
    [pbtime_i5, pbcwnd_i5, ssthresh5] = tcpprobe_reader([path, pbfile5], rtt);
    %pbtime_i5 = pbtime_i5 - min(pbtime_i5);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
if ischar(pbfile6)
    [pbtime_i6, pbcwnd_i6, ssthresh6] = tcpprobe_reader([path, pbfile6], rtt);
    %pbtime_i6 = pbtime_i6 - min(pbtime_i6);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
if ischar(pbfile7)
    [pbtime_i7, pbcwnd_i7, ssthresh7] = tcpprobe_reader([path, pbfile7], rtt);
    %pbtime_i7 = pbtime_i7 - min(pbtime_i7);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
if ischar(pbfile8)
    [pbtime_i8, pbcwnd_i8, ssthresh8] = tcpprobe_reader([path, pbfile8], rtt);
    %pbtime_i8 = pbtime_i8 - min(pbtime_i8);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
if ischar(pbfile9)
    [pbtime_i9, pbcwnd_i9, ssthresh9] = tcpprobe_reader([path, pbfile9], rtt);
    %pbtime_i9 = pbtime_i9 - min(pbtime_i9);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
if ischar(pbfile10)
    [pbtime_i10, pbcwnd_i10, ssthresh10] = tcpprobe_reader([path, pbfile10], rtt);
    %pbtime_i10 = pbtime_i10 - min(pbtime_i10);
```

```
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end

duracion = toc;
fprintf('Hecho! Duracion: %f segundos\n', duracion)

%% Leyendo archivo tshark
fprintf('Importando datos desde primer archivo tshark ... ')
tic
if ischar(fshark1)
      [time_i1, ACK1, SEQ1, LEN1, REORDER1, RTX1] = shark_reader([path, fshark1]);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
fprintf('Importando datos desde segundo archivo tshark ... ')
if ischar(fshark2)
      [time_i2, ACK2, SEQ2, LEN2, REORDER2, RTX2] = shark_reader([path, fshark2]);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
fprintf('Importando datos desde tercer archivo tshark ... ')
if ischar(fshark3)
      [time_i3, ACK3, SEQ3, LEN3, REORDER3, RTX3] = shark_reader([path, fshark3]);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
fprintf('Importando datos desde cuarto archivo tshark ... ')
if ischar(fshark4)
      [time_i4, ACK4, SEQ4, LEN4, REORDER4, RTX4] = shark_reader([path, fshark4]);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
fprintf('Importando datos desde quinto archivo tshark ... ')
if ischar(fshark5)
      [time_i5, ACK5, SEQ5, LEN5, REORDER5, RTX5] = shark_reader([path, fshark5]);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
fprintf('Importando datos desde sexto archivo tshark ... ')
if ischar(fshark6)
      [time_i6, ACK6, SEQ6, LEN6, REORDER6, RTX6] = shark_reader([path, fshark6]);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
fprintf('Importando datos desde septimo archivo tshark ... ')
if ischar(fshark7)
      [time_i7, ACK7, SEQ7, LEN7, REORDER7, RTX7] = shark_reader([path, fshark7]);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
fprintf('Importando datos desde octavo archivo tshark ... ')
if ischar(fshark8)
      [time_i8, ACK8, SEQ8, LEN8, REORDER8, RTX8] = shark_reader([path, fshark8]);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
fprintf('Importando datos desde noveno archivo tshark ... ')
if ischar(fshark9)
      [time_i9, ACK9, SEQ9, LEN9, REORDER9, RTX9] = shark_reader([path, fshark9]);
else
    fprintf('--> Ningun archivo seleccionado\n\n')
end
fprintf('Importando datos desde decimo archivo tshark ... ')
```

```
    if ischar(fshark10)
          [time_i10, ACK10, SEQ10, LEN10, REORDER10, RTX10] = shark_reader([path,
fshark10]);
    else
        fprintf('--> Ningun archivo seleccionado\n\n')
    end

    [pbtime1, pbcwnd1, pbslope1, pbdec1, pbthrput1, beta_reno1, beta_gaimd1] =
tcpprobe_values(pbtime_i1, pbcwnd_i1, ssthresh1, mss, rtt);
    [pbtime2, pbcwnd2, pbslope2, pbdec2, pbthrput2, beta_reno2, beta_gaimd2] =
tcpprobe_values(pbtime_i2, pbcwnd_i2, ssthresh2, mss, rtt);
    [pbtime3, pbcwnd3, pbslope3, pbdec3, pbthrput3, beta_reno3, beta_gaimd3] =
tcpprobe_values(pbtime_i3, pbcwnd_i3, ssthresh3, mss, rtt);
    [pbtime4, pbcwnd4, pbslope4, pbdec4, pbthrput4, beta_reno4, beta_gaimd4] =
tcpprobe_values(pbtime_i4, pbcwnd_i4, ssthresh4, mss, rtt);
    [pbtime5, pbcwnd5, pbslope5, pbdec5, pbthrput5, beta_reno5, beta_gaimd5] =
tcpprobe_values(pbtime_i5, pbcwnd_i5, ssthresh5, mss, rtt);
    [pbtime6, pbcwnd6, pbslope6, pbdec6, pbthrput6, beta_reno6, beta_gaimd6] =
tcpprobe_values(pbtime_i6, pbcwnd_i6, ssthresh6, mss, rtt);
    [pbtime7, pbcwnd7, pbslope7, pbdec7, pbthrput7, beta_reno7, beta_gaimd7] =
tcpprobe_values(pbtime_i7, pbcwnd_i7, ssthresh7, mss, rtt);
    [pbtime8, pbcwnd8, pbslope8, pbdec8, pbthrput8, beta_reno8, beta_gaimd8] =
tcpprobe_values(pbtime_i8, pbcwnd_i8, ssthresh8, mss, rtt);
    [pbtime9, pbcwnd9, pbslope9, pbdec9, pbthrput9, beta_reno9, beta_gaimd9] =
tcpprobe_values(pbtime_i9, pbcwnd_i9, ssthresh9, mss, rtt);
    [pbtime10, pbcwnd10, pbslope10, pbdec10, pbthrput10, beta_reno10, beta_gaimd10] =
tcpprobe_values(pbtime_i10, pbcwnd_i10, ssthresh10, mss, rtt);

    [time1, wscwnd1, pktsend1, thrput1, beta_estp1] = shark_values(time_i1, SEQ1, LEN1,
ACK1, REORDER1, RTX1, mss, rtt, tau);
    [time2, wscwnd2, pktsend2, thrput2, beta_estp2] = shark_values(time_i2, SEQ2, LEN2,
ACK2, REORDER2, RTX2, mss, rtt, tau);
    [time3, wscwnd3, pktsend3, thrput3, beta_estp3] = shark_values(time_i3, SEQ3, LEN3,
ACK3, REORDER3, RTX3, mss, rtt, tau);
    [time4, wscwnd4, pktsend4, thrput4, beta_estp4] = shark_values(time_i4, SEQ4, LEN4,
ACK4, REORDER4, RTX4, mss, rtt, tau);
    [time5, wscwnd5, pktsend5, thrput5, beta_estp5] = shark_values(time_i5, SEQ5, LEN5,
ACK5, REORDER5, RTX5, mss, rtt, tau);
    [time6, wscwnd6, pktsend6, thrput6, beta_estp6] = shark_values(time_i6, SEQ6, LEN6,
ACK6, REORDER6, RTX6, mss, rtt, tau);
    [time7, wscwnd7, pktsend7, thrput7, beta_estp7] = shark_values(time_i7, SEQ7, LEN7,
ACK7, REORDER7, RTX7, mss, rtt, tau);
    [time8, wscwnd8, pktsend8, thrput8, beta_estp8] = shark_values(time_i8, SEQ8, LEN8,
ACK8, REORDER8, RTX8, mss, rtt, tau);
    [time9, wscwnd9, pktsend9, thrput9, beta_estp9] = shark_values(time_i9, SEQ9, LEN9,
ACK9, REORDER9, RTX9, mss, rtt, tau);
    [time10, wscwnd10, pktsend10, thrput10, beta_estp10] = shark_values(time_i10,
SEQ10, LEN10, ACK10, REORDER10, RTX10, mss, rtt, tau);

    thrput_totales =
[thrput1,thrput2,thrput3,thrput4,thrput5,thrput6,thrput7,thrput8,thrput9,thrput10];
    pbthrput_totales =
[pbthrput1,pbthrput2,pbthrput3,pbthrput4,pbthrput5,pbthrput6,pbthrput7,pbthrput8,pbthrp
ut9,pbthrput10];
    muestras = (1:10);

    CWND1=mean(pbcwnd1);
    CWND2=mean(pbcwnd2);
    CWND3=mean(pbcwnd3);
    CWND4=mean(pbcwnd4);
    CWND5=mean(pbcwnd5);
    CWND6=mean(pbcwnd6);
    CWND7=mean(pbcwnd7);
    CWND8=mean(pbcwnd8);
```

```
    CWND9=mean(pbcwnd9);
    CWND10=mean(pbcwnd10);

    CWND_totales = [CWND1,CWND2,CWND3,CWND4,CWND5,CWND6,CWND7,CWND8,CWND9,CWND10];

    pbdec1_prom = mean(pbdec1(~isnan(pbdec1)));
    pbdec2_prom = mean(pbdec2(~isnan(pbdec2)));
    pbdec3_prom = mean(pbdec3(~isnan(pbdec3)));
    pbdec4_prom = mean(pbdec4(~isnan(pbdec4)));
    pbdec5_prom = mean(pbdec5(~isnan(pbdec5)));
    pbdec6_prom = mean(pbdec6(~isnan(pbdec6)));
    pbdec7_prom = mean(pbdec7(~isnan(pbdec7)));
    pbdec8_prom = mean(pbdec8(~isnan(pbdec8)));
    pbdec9_prom = mean(pbdec9(~isnan(pbdec9)));
    pbdec10_prom = mean(pbdec10(~isnan(pbdec10)));

    pbdec_totales = [pbdec1_prom, pbdec2_prom, pbdec3_prom, pbdec4_prom, pbdec5_prom,
pbdec6_prom, pbdec7_prom, pbdec8_prom, pbdec9_prom, pbdec10_prom];

    if protocol ==1             %Si el protocolo empleado es RENO
        beta1 = beta_reno1;
        beta2 = beta_reno2;
        beta3 = beta_reno3;
        beta4 = beta_reno4;
        beta5 = beta_reno5;
        beta6 = beta_reno6;
        beta7 = beta_reno7;
        beta8 = beta_reno8;
        beta9 = beta_reno9;
        beta10 = beta_reno10;

    else
        if protocol == 2        %Si el protocolo empleado es GAIMD
            beta1 = beta_gaimd1;
            beta2 = beta_gaimd2;
            beta3 = beta_gaimd3;
            beta4 = beta_gaimd4;
            beta5 = beta_gaimd5;
            beta6 = beta_gaimd6;
            beta7 = beta_gaimd7;
            beta8 = beta_gaimd8;
            beta9 = beta_gaimd9;
            beta10 = beta_gaimd10;
        else
            if protocol == 3    %Si el protocolo empleado es ESTP
                beta1 = beta_estp1;
                beta2 = beta_estp2;
                beta3 = beta_estp3;
                beta4 = beta_estp4;
                beta5 = beta_estp5;
                beta6 = beta_estp6;
                beta7 = beta_estp7;
                beta8 = beta_estp8;
                beta9 = beta_estp9;
                beta10 = beta_estp10;
            end
        end
    end

    if ((protocol==1 || protocol ==2) || protocol==3)
    beta1_prom=mean(beta1(~isnan(beta1)));
    beta2_prom=mean(beta2(~isnan(beta2)));
    beta3_prom=mean(beta3(~isnan(beta3)));
```

```
    beta4_prom=mean(beta4(~isnan(beta4)));
    beta5_prom=mean(beta5(~isnan(beta5)));
    beta6_prom=mean(beta6(~isnan(beta6)));
    beta7_prom=mean(beta7(~isnan(beta7)));
    beta8_prom=mean(beta8(~isnan(beta8)));
    beta9_prom=mean(beta9(~isnan(beta9)));
    beta10_prom=mean(beta10(~isnan(beta10)));

    beta1_mode=mode(beta1(~isnan(beta1)));
    beta2_mode=mode(beta2(~isnan(beta2)));
    beta3_mode=mode(beta3(~isnan(beta3)));
    beta4_mode=mode(beta4(~isnan(beta4)));
    beta5_mode=mode(beta5(~isnan(beta5)));
    beta6_mode=mode(beta6(~isnan(beta6)));
    beta7_mode=mode(beta7(~isnan(beta7)));
    beta8_mode=mode(beta8(~isnan(beta8)));
    beta9_mode=mode(beta9(~isnan(beta9)));
    beta10_mode=mode(beta10(~isnan(beta10)));
    end

    if(protocol == 1 || protocol ==2)
        beta_totales = [beta1_mode, beta2_mode, beta3_mode, beta4_mode, beta5_mode,
beta6_mode, beta7_mode, beta8_mode, beta9_mode, beta10_mode];
    end
    if protocol == 3
        beta_totales = [beta1_prom, beta2_prom, beta3_prom, beta4_prom, beta5_prom,
beta6_prom, beta7_prom, beta8_prom, beta9_prom, beta10_prom];
    end
    if protocol == 4
        beta_totales = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
    end

    fprintf('Convirtiendo datos a vectores. Espere por favor ...\n')
    for n=1:50
        fprintf('*')
    end
    fprintf('\n')
    dur_t = toc;
    fprintf('*\nHecho! Duración: %f segundos\n', dur_t)

    data_tcp1 = zeros(10,6);
    data_tcp1(:,1) = muestras;
    data_tcp1(:,2) = pbthrput_totales/1000000;
    data_tcp1(:,3) = thrput_totales/1000000;
    data_tcp1(:,4) = CWND_totales;
    data_tcp1(:,5) = beta_totales;
    data_tcp1(:,6) = pbdec_totales;

    data_tcp2 = zeros(2,6);
    data_tcp2(1,2) = mean(pbthrput_totales)/1000000;
    data_tcp2(1,3) = mean(thrput_totales)/1000000;
    data_tcp2(1,4) = mean(CWND_totales);
    data_tcp2(1,5) = mean(beta_totales);
    data_tcp2(1,6) = mean(pbdec_totales);
    data_tcp2(2,2) = std(pbthrput_totales)/1000000;
    data_tcp2(2,3) = std(thrput_totales)/1000000;
    data_tcp2(2,4) = std(CWND_totales);
    data_tcp2(2,5) = std(beta_totales);
    data_tcp2(2,6) = std(pbdec_totales);

    data_tcp = [data_tcp1; data_tcp2];

    dlmwrite('datatcp.csv',data_tcp,'precision','%.4f');
```

```
end

%% Gráfica de ventana de congestión y rendimiento

    h_fig = figure(1);
    set(h_fig, 'Color', [1 1 1])
    subplot(5,3,[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15])
    set(gca, 'Color', [1 1 1])
    hold on

    subplot(5,3,1)
    set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
    hold on
    plot(pbtime_i1,pbcwnd_i1)
    axis([0 60 0 50])
    xlabel('Tiempo(s.)','FontSize',6,'Color','k')
    h = title('[1]: Ventana de congestion en el tiempo');
    set(h, 'Color','k')

    subplot(5,3,2)
    set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
    hold on
    plot(pbtime_i2,pbcwnd_i2)
    axis([0 60 0 50])
    xlabel('Tiempo(s.)','FontSize',6,'Color','k')
    h=title('[2]: Ventana de congestion en el tiempo');
    set(h,'Color','k')

    subplot(5,3,4)
    set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
    hold on
    plot(pbtime_i3,pbcwnd_i3)
    axis([0 60 0 50])
    xlabel('Tiempo(s.)','FontSize',6,'Color','k')
    h = title('[3]: Ventana de congestion en el tiempo');
    set(h,'Color','k')

    subplot(5,3,5)
    set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
    hold on
    plot(pbtime_i4,pbcwnd_i4)
    axis([0 60 0 50])
    xlabel('Tiempo(s.)','FontSize',6,'Color','k')
    h = title('[4]: Ventana de congestion en el tiempo');
    set(h,'Color','k')

    subplot(5,3,7)
    set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
    hold on
    plot(pbtime_i5,pbcwnd_i5)
    axis([0 60 0 50])
    xlabel('Tiempo(s.)','FontSize',6,'Color','k')
    h=title('[5]: Ventana de congestion en el tiempo');
    set(h,'Color','k');

    subplot(5,3,8)
    set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
    hold on
    plot(pbtime_i6,pbcwnd_i6)
    axis([0 60 0 50])
    xlabel('Tiempo(s.)','FontSize',6,'Color','k')
```

```
h=title('[6]: Ventana de congestion en el tiempo');
set(h,'Color','k')

subplot(5,3,10)
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
plot(pbtime_i7,pbcwnd_i7)
axis([0 60 0 50])
xlabel('Tiempo(s.)','FontSize',6,'Color','k')
h=title('[7]: Ventana de congestion en el tiempo');
set(h,'Color','k')

subplot(5,3,11)
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
plot(pbtime_i8,pbcwnd_i8)
axis([0 60 0 50])
xlabel('Tiempo(s.)','FontSize',6,'Color','k')
h=title('[8]: Ventana de congestion en el tiempo');
set(h,'Color','k')

subplot(5,3,13)
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
plot(pbtime_i9,pbcwnd_i9)
axis([0 60 0 50])
xlabel('Tiempo(s.)','FontSize',6,'Color','k')
h=title('[9]: Ventana de congestion en el tiempo');
set(h,'Color','k')

subplot(5,3,14)
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
plot(pbtime_i10,pbcwnd_i10)
axis([0 60 0 50])
xlabel('Tiempo(s.)','FontSize',6,'Color','k')
h=title('[10]: Ventana de congestion en el tiempo');
set(h,'Color','k')

subplot(5,3,[3 6])
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
plot(muestras,thrput_totales,'b+',muestras,pbthrput_totales,'rx')
axis([0 11 0 10000000])
xlabel('Transmisión en la muestra','Color','k')
ylabel('Bits/s','Color','k')
h = title('Rendimiento (Bits/s)');
set(h, 'Color','k')
hleg1 = legend('Datos de Tshark','Datos de TCPprobe');
set(hleg1,'Color', [0 0 0], 'Location','NorthWest', 'TextColor', 'k', 'Box', 'off')

subplot(5,3,[9 12])
set(gca, 'Color', [1 1 1], 'XColor', [1 1 1], 'YColor', [1 1 1])
hold on
axis([0,25,0,50])

txta = text(5,44,'Rendimiento(Mb/s)');
set(txta, 'Color' , 'k' , 'FontSize' , 9)
txtb = text(20,44,'Ventana(Paquetes)');
set(txtb, 'Color' , 'k' , 'FontSize' , 9)

txt1 = text(5,40,'TCPProbe');
set(txt1, 'Color' , 'r' , 'FontSize' , 9)
```

```
    txt2 = text(12,40,'Tshark');
    set(txt2, 'Color' , 'b' , 'FontSize' , 9)

    for n=1:10

        subplot(5,3,[9 12])

        info1 = num2str(round(pbthrput_totales(n)/1000)/1000);
        info2 = num2str(round(thrput_totales(n)/1000)/1000);
        info3 = num2str(round(1000*CWND_totales(n))/1000);
        number = num2str(n);

        txt3 = text(1,(40-4*n),number);
        set(txt3, 'Color' , 'k' , 'FontSize' , 9)
        txt4 = text(5,(40-4*n),info1);
        set(txt4, 'Color' , 'k' , 'FontSize' , 9)
        txt5 = text(12,(40-4*n),info2);
        set(txt5, 'Color' , 'k' , 'FontSize' , 9)
        txt6 = text(20,(40-4*n),info3);
        set(txt6, 'Color' , 'k' , 'FontSize' , 9)

    end


    subplot(5,3,15)
    set(gca, 'Color', [1 1 1], 'XColor', [1 1 1], 'YColor', [1 1 1])
    hold on
    axis([0,25,0,25])

    promedio = text(5,20,'Promedios');
    set(promedio, 'Color' , 'k' , 'FontSize' , 9)

    info4 = num2str(round(mean(pbthrput_totales)/1000)/1000);
    info5 = num2str(round(mean(thrput_totales)/1000)/1000);
    info6 = num2str(round(mean(CWND_totales)*1000)/1000);


        txt11 = text(5,15,info4);
        set(txt11, 'Color' , 'k' , 'FontSize' , 9)
        txt12 = text(12,15,info5);
        set(txt12, 'Color' , 'k' , 'FontSize' , 9)
        txt13 = text(20,15,info6);
        set(txt13, 'Color' , 'k' , 'FontSize' , 9)

        desviacion = text(5,5,'Desviacion estandar');
        set(desviacion, 'Color' , 'k' , 'FontSize' , 9)

     info7 = num2str(round(std(pbthrput_totales)/1000)/1000);
     info8 = num2str(round(std(thrput_totales)/1000)/1000);
     info9 = num2str(round(std(CWND_totales)*1000)/1000);

        txt14 = text(5,0,info7);
        set(txt14, 'Color' , 'k' , 'FontSize' , 9)
        txt15 = text(12,0,info8);
        set(txt15, 'Color' , 'k' , 'FontSize' , 9)
        txt16 = text(20,0,info9);
        set(txt16, 'Color' , 'k' , 'FontSize' , 9)

%% Gráfica de beta y decremento real

    h_fig2 = figure(2);
```

```
set(h_fig2, 'Color', [1 1 1])
subplot(5,3,[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15])
set(gca, 'Color', [1 1 1])
hold on

subplot(5,3,1)
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
plot(pbtime1,pbdec1, 'x', 'Color', [0 0.1 1])
if(protocol == 1 || protocol ==2)
    plot(pbtime1,beta1, 'r+')
end
if protocol == 3
    plot(time1,beta1, 'r+')
end
axis([0 60 0 1])
xlabel('Tiempo(s.)','FontSize',6,'Color','k')
h = title('[1]: Decremento multiplicativo (Beta)');
set(h, 'Color','k')

subplot(5,3,2)
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
plot(pbtime2,pbdec2, 'x', 'Color', [0 0.1 1])
if(protocol == 1 || protocol ==2)
    plot(pbtime2,beta2, 'r+')
end
if protocol == 3
    plot(time2,beta2, 'r+')
end
axis([0 60 0 1])
xlabel('Tiempo(s.)','FontSize',6,'Color','k')
h=title('[2]: Decremento multiplicativo (Beta)');
set(h,'Color','k')

subplot(5,3,4)
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
plot(pbtime3,pbdec3, 'x', 'Color', [0 0.1 1])
if(protocol == 1 || protocol ==2)
    plot(pbtime3,beta3, 'r+')
end
if protocol == 3
    plot(time3,beta3, 'r+')
end
axis([0 60 0 1])
xlabel('Tiempo(s.)','FontSize',6,'Color','k')
h = title('[3]: Decremento multiplicativo (Beta)');
set(h,'Color','k')

subplot(5,3,5)
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
plot(pbtime4,pbdec4, 'x', 'Color', [0 0.1 1])
if(protocol == 1 || protocol ==2)
    plot(pbtime4,beta4, 'r+')
end
if protocol == 3
    plot(time4,beta4, 'r+')
end
axis([0 60 0 1])
xlabel('Tiempo(s.)','FontSize',6,'Color','k')
h = title('[4]: Decremento multiplicativo (Beta)');
set(h,'Color','k')
```

```
subplot(5,3,7)
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
plot(pbtime5,pbdec5, 'x', 'Color', [0 0.1 1])
if(protocol == 1 || protocol ==2)
    plot(pbtime5,beta5, 'r+')
end
if protocol == 3
    plot(time5,beta5, 'r+')
end
axis([0 60 0 1])
xlabel('Tiempo(s.)','FontSize',6,'Color','k')
h=title('[5]: Decremento multiplicativo (Beta)');
set(h,'Color','k');

subplot(5,3,8)
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
plot(pbtime6,pbdec6, 'x', 'Color', [0 0.1 1])
if(protocol == 1 || protocol ==2)
    plot(pbtime6,beta6, 'r+')
end
if protocol == 3
    plot(time6,beta6, 'r+')
end
axis([0 60 0 1])
xlabel('Tiempo(s.)','FontSize',6,'Color','k')
h=title('[6]: Decremento multiplicativo (Beta)');
set(h,'Color','k')

subplot(5,3,10)
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
plot(pbtime7,pbdec7, 'x', 'Color', [0 0.1 1])
if(protocol == 1 || protocol ==2)
    plot(pbtime7,beta7, 'r+')
end
if protocol == 3
    plot(time7,beta7, 'r+')
end
axis([0 60 0 1])
xlabel('Tiempo(s.)','FontSize',6,'Color','k')
h=title('[7]: Decremento multiplicativo (Beta)');
set(h,'Color','k')

subplot(5,3,11)
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
plot(pbtime8,pbdec8, 'x', 'Color', [0 0.1 1])
if(protocol == 1 || protocol ==2)
    plot(pbtime8,beta8, 'r+')
end
if protocol == 3
    plot(time8,beta8, 'r+')
end
axis([0 60 0 1])
xlabel('Tiempo(s.)','FontSize',6,'Color','k')
h=title('[8]: Decremento multiplicativo (Beta)');
set(h,'Color','k')

subplot(5,3,13)
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
```

```
plot(pbtime9,pbdec9, 'x', 'Color', [0 0.1 1])
if(protocol == 1 || protocol ==2)
    plot(pbtime9,beta9, 'r+')
end
if protocol == 3
    plot(time9,beta9, 'r+')
end
axis([0 60 0 1])
xlabel('Tiempo(s.)','FontSize',6,'Color','k')
h=title('[9]: Decremento multiplicativo (Beta)');
set(h,'Color','k')

subplot(5,3,14)
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
plot(pbtime10,pbdec10, 'x', 'Color', [0 0.1 1])
 if(protocol == 1 || protocol ==2)
    plot(pbtime10,beta10, 'r+')
end
if protocol == 3
    plot(time10,beta10, 'r+')
end
axis([0 60 0 1])
xlabel('Tiempo(s.)','FontSize',6,'Color','k')
h=title('[10]: Decremento multiplicativo (Beta)');
set(h,'Color','k')

subplot(5,3,[3 6])
set(gca, 'Color', [1 1 1], 'XColor', [0 0 0], 'YColor', [0 0 0])
hold on
plot(muestras,pbdec_totales,'x', 'Color', [0 0.1 1])
if protocol ~=4
    plot(muestras,beta_totales,'r+')
end
axis([0 10 0 1])
xlabel('Transmisión en la muestra','Color','k')
ylabel('Beta','Color','k')
h = title('Decremento multiplicativo');
set(h, 'Color','k')

txt_legen1a = text(3,0.2,'x');
txt_legen1b = text(4,0.2,'Decremento real');
set(txt_legen1a, 'Color' , [0 0.1 1], 'FontSize' , 11 )
set(txt_legen1b, 'Color' , 'k' , 'FontSize' , 9)
if protocol ~=4
    txt_legen2a = text(3,0.1,'+');
    txt_legen2b = text(4,0.1,'Beta');
    set(txt_legen2a, 'Color' , 'r', 'FontSize' , 11)
    set(txt_legen2b, 'Color' , 'k' , 'FontSize' , 9)
end

subplot(5,3,[9 12])
set(gca, 'Color', [1 1 1], 'XColor', [1 1 1], 'YColor', [1 1 1])
hold on
axis([0,25,0,50])

txt_01 = text(5,40,'Decremento');
set(txt_01, 'Color' , 'k' , 'FontSize' , 9)
if protocol ~=4
    txt_11 = text(15,40,'Beta');
    set(txt_11, 'Color' , 'k' , 'FontSize' , 9)
end
```

```
for n=1:10

    subplot(5,3,[9 12])

    infodec = num2str(round(1000*pbdec_totales(n))/1000);
    infobeta = num2str(round(1000*beta_totales(n))/1000);
    id = num2str(n);

    txt_02 = text(1,(40-4*n),id);
    set(txt_02, 'Color' , 'k' , 'FontSize' , 9)
    txt_03 = text(5,(40-4*n),infodec);
    set(txt_03, 'Color' , 'k' , 'FontSize' , 9)
    if protocol ~=4
        txt_13 = text(15,(40-4*n),infobeta);
        set(txt_13, 'Color' , 'k' , 'FontSize' , 9)
    end
end

subplot(5,3,15)
set(gca, 'Color', [1 1 1], 'XColor', [1 1 1], 'YColor', [1 1 1])
hold on
axis([0,25,0,25])

promediobeta = text(5,25,'Promedio');
set(promediobeta, 'Color' , 'k' , 'FontSize' , 9)

infodecprom = num2str(round(mean(pbdec_totales)*1000)/1000);
infobetaprom = num2str(round(mean(beta_totales)*1000)/1000);


    txt_prom = text(5,20,infodecprom);
    set(txt_prom, 'Color' , 'k' , 'FontSize' , 9)

    if protocol ~=4

    txt_promb = text(15,20,infobetaprom);
    set(txt_promb, 'Color' , 'k' , 'FontSize' , 9)
    end

    txt_stdbeta = text(5,10,'Desviacion estandar');
    set(txt_stdbeta, 'Color' , 'k' , 'FontSize' , 9)

infodecstd = num2str(round(std(pbdec_totales)*1000)/1000);
infobetastd = num2str(round(std(beta_totales)*1000)/1000);


    txt_std = text(5,5,infodecstd);
    set(txt_std, 'Color' , 'k' , 'FontSize' , 9)

    if protocol ~=4
    txt_stdb = text(15,5,infobetastd);
    set(txt_stdb, 'Color' , 'k' , 'FontSize' , 9)
    end
```

ANEXO "G"
PROGRAMA EN MATLAB PARA GENERACIÓN DE MATRIZ EMPLEADA EN ESTP

1. Matriz de valores δ y β para un valor τ=50.

1.1. Archivo de MATLAB

Nombre de archivo: /home/owl-wksnt-tcp/SIMULACION/matriz_estp_tau_fijo.m

```
tau = 50;               %Valor de "Tau" en protocolo ESTP
step = 1;
delta = 1:step:315;     %Vector con valores de "delta" según protocolo ESTP
beta = (exp(-(delta-1)/tau)+1).^-1;     %Vector de beta: decremento multiplicativo
beta_matrix = round(beta*256);          %Vector de (beta)*256
%Matriz de valores en formato empleado por lenguaje C
for n=1:length(delta)
    fprintf(' {%d, %d, /* %1.3f */ },\n', delta(n), beta_matrix(n), beta(n))
end
```

Figura D1. /home/owl-wksnt-tcp/SIMULACION /matriz_estp_tau_fijo.m

2. Matriz ESTP

2.1. Archivo de MATLAB

Nombre de archivo: /home/owl-wksnt-tcp/SIMULACION /matriz_estp_tau_variable.m

```
step = 0.01;
exponent = 0:step:6;                %Vector con valores de "exp" según protocolo ESTP
beta = (exp(-exponent)+1).^-1;      %Vector de beta: decremento multiplicativo
exp_matrix = exponent.*100;         %Vector de (exp)*100
beta_matrix = round(beta*256);      %Vector de (beta)*256
%Matriz de valores en formato empleado por lenguaje C
for n=1:length(exponent)
    fprintf(' {%.0f, %d, /* %1.3f */ },\n', exp_matrix(n), beta_matrix(n), beta(n))
end
```

Figura D2. /home/owl-wksnt-tcp/SIMULACION /matriz_estp_tau_variable.m

2.2. Resultado en MATLAB

Se muestran las primeras 10 líneas de la matriz generada.

```
Hecho! Duración: 14.243666 segundos
>> matriz_ESTP_tau_variable
 {0, 128, /* 0.500 */ },
 {1, 129, /* 0.502 */ },
 {2, 129, /* 0.505 */ },
 {3, 130, /* 0.507 */ },
 {4, 131, /* 0.510 */ },
 {5, 131, /* 0.512 */ },
 {6, 132, /* 0.515 */ },
 {7, 132, /* 0.517 */ },
 {8, 133, /* 0.520 */ },
 {9, 134, /* 0.522 */ },
 {10, 134, /* 0.525 */ },
```

Para la programación del protocolo se emplean 520 líneas de la matriz (Ver Anexo J)

ANEXO "H"

PROGRAMA DE TCP RENO

Programa de TCP Reno en lenguaje C en código fuente de Línux.

Archivo: <Directorio de código fuente>/net/ipv4/tcp_reno2.c Versión de kernel: 3.13.0-32

```
/* Control de congestión: TCP Reno
 * Programado por: Jesus Bravo (basado en tcp_cong.c)
 */
#include <linux/module.h>
#include <net/tcp.h>

static void tcp_reno2_init(struct sock *sk)
{
    struct tcp_sock *tp = tcp_sk(sk);
    tp->snd_cwnd = 1;
/* Asegurar que la ventana de congestión no alcance un
valor muy alto*/
/* snd_cwnd_clamp tomará como mínimo valor 0xffffffff/128 */
    tp->snd_cwnd_clamp = min_t(u32, tp->snd_cwnd_clamp,
0xffffffff/128);
}
static void tcp_reno2_cong_avoid(struct sock *sk, u32 ack,
u32 acked, u32 in_flight)
{
    struct tcp_sock *tp = tcp_sk(sk);

    if (!tcp_is_cwnd_limited(sk, in_flight))
         return;
// Ejecuta slow-start si: snd_cwnd <= snd_ssthresh
    if (tp->snd_cwnd <= tp->snd_ssthresh)
         tcp_slow_start(tp, acked);
    else {
         /* Realiza incremento aditivo */
         if (tp->snd_cwnd < tp->snd_cwnd_clamp) {
                /* Contador snd_cwnd_cnt incrementa en 1 */
                tp->snd_cwnd_cnt ++;
                if (tp->snd_cwnd_cnt >= tp->snd_cwnd) {
         /* Contador snd_cwnd_cnt retorna a 0 */
         /* cuando alcanza el valor de snd_cwnd */
                        tp->snd_cwnd_cnt =0;
                        tp->snd_cwnd++;
                }
         }
    }
}
static u32 tcp_reno2_ssthresh(struct sock *sk)
{
    const struct tcp_sock *tp = tcp_sk(sk);
    /* Realiza decremento multiplicativo*/
    return max(tp->snd_cwnd - (tp->snd_cwnd >> 1), 2U);
}
static u32 tcp_reno2_min_cwnd(const struct sock *sk)
{
    return 1U;
}

static struct tcp_congestion_ops tcp_reno2_reg
__read_mostly = {
    .init       = tcp_reno2_init,
    .ssthresh   = tcp_reno2_ssthresh,
    .cong_avoid = tcp_reno2_cong_avoid,
    .min_cwnd   = tcp_reno2_min_cwnd,
    .owner              = THIS_MODULE,
    .name       = "reno2"
};
static int __init tcp_reno2_register(void)
{
//  BUILD_BUG_ON(sizeof(struct tcp_reno2) >
ICSK_CA_PRIV_SIZE);
    return tcp_register_congestion_control(&tcp_reno2_reg);
}
static void __exit tcp_reno2_unregister(void)
{
    tcp_unregister_congestion_control(&tcp_reno2_reg);
}
module_init(tcp_reno2_register);
module_exit(tcp_reno2_unregister);

MODULE_AUTHOR("JMBS");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("TCP Reno2");
```

ANEXO "I"

PROGRAMA DE GAIMD

Programa de GAIMD en lenguaje C en código fuente de Línux.

Archivo: <Directorio de código fuente>/net/ipv4/tcp_gaimd.c Versión de kernel: 3.13.0-32

```
/*
TCP GAIMD  con parametros alfa y beta
Programado por Jesus Bravo
*/

#include <linux/module.h>
#include <net/tcp.h>

struct gaimd {
	u32	dec;		//Decremento multiplicativo=dec/8
};

static void gaimd_init(struct sock *sk)
{
	struct tcp_sock *tp = tcp_sk(sk);
	struct gaimd *ca = inet_csk_ca(sk);

	tp->snd_cwnd = 1;
	ca->dec = 4;
	/* Asegurar que la ventana de congestiÃ²n no alcance
un valor muy alto*/
	/*snd_cwnd_clamp tomarÃ  como minimo valor
0xffffffff/128*/
	tp->snd_cwnd_clamp = min_t(u32, tp->snd_cwnd_clamp,
0xffffffff/128);
}

static void gaimd_cong_avoid(struct sock *sk, u32 ack, u32
acked, u32 in_flight)
{
	struct tcp_sock *tp = tcp_sk(sk);
	struct gaimd *ca = inet_csk_ca(sk);

	if (!tcp_is_cwnd_limited(sk, in_flight))
		return;

	if (tp->snd_cwnd <= tp->snd_ssthresh){
		tcp_slow_start(tp, acked);
		ca->dec = 4;
	/*Si ocurre una pÃ©rdida durante slow-start el
decremento multiplicativo es:
	dec/8 = 4/8 = 0.5*/
	}
	else {
		/* Realiza el incremento aditivo */
		if (tp->snd_cwnd < tp->snd_cwnd_clamp) {
			tp->snd_cwnd_cnt++;
			ca->dec = 7;
			if ((tp->snd_cwnd_cnt * 5) >= (tp-
>snd_cwnd * 16)){
			/*Incremento aditivo = 5/16 = 0.3125*/
				tp->snd_cwnd_cnt = 0;
				tp->snd_cwnd++;
			}
		}
	}
}

static u32 gaimd_ssthresh(struct sock *sk)
{
	const struct tcp_sock *tp = tcp_sk(sk);
	const struct gaimd *ca = inet_csk_ca(sk);

	/* Decremento multiplicativo */
	return max(((tp->snd_cwnd*ca->dec+4) >> 3U), 2U);
	/* beta = dec/8, beta=7/8 cuando la perdida es
durante el incremento aditivo*/
}

static u32 tcp_gaimd_min_cwnd(const struct sock *sk)
```

```
{
	return 1U;
}

static void  gaimd_state(struct sock *sk, u8 new_state)
{
	struct tcp_sock *tp = tcp_sk(sk);
//	struct gaimd *ca = inet_csk_ca(sk);
	/*En caso la perdida sea por timeout*/
	if (new_state == TCP_CA_Loss){
		tp->snd_ssthresh = (tp->snd_cwnd >> 1U);
		tp->snd_cwnd =  1U;
	}
}


static struct tcp_congestion_ops tcp_gaimd __read_mostly =
{
	.init		= gaimd_init,
	.ssthresh	= gaimd_ssthresh,
	.cong_avoid	= gaimd_cong_avoid,
	.min_cwnd	= tcp_gaimd_min_cwnd,
	.set_state	= gaimd_state,
	.owner		= THIS_MODULE,
	.name		= "gaimd"
};

static int __init gaimd_register(void)
{
	BUILD_BUG_ON(sizeof(struct gaimd) >
ICSK_CA_PRIV_SIZE);
	return tcp_register_congestion_control(&tcp_gaimd);
}

static void __exit gaimd_unregister(void)
{
	tcp_unregister_congestion_control(&tcp_gaimd);
}

module_init(gaimd_register);
module_exit(gaimd_unregister);

MODULE_AUTHOR("Gaimd");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Gaimd TCP");
```

ANEXO "J"

PROGRAMA DE ESTP

Programa de ESTP en lenguaje C en código fuente de Línux

Archivo: <Directorio de código fuente>/net/ipv4/tcp_estp.c Versión de kernel: 3.13.0-32

```
/* Ethernet Service Transport Protcol
 * Autor: Claudio Estevez
 * Programador: Jesus Bravo
 */
#include <linux/module.h>
#include <net/tcp.h>

/* Valores de beta para ESTP */
static const struct estp_md_val {
      unsigned int exp_ref;     //Numero que se compara
con el valor de "exp" , exp = 100*(alfa-1)/(tau)
      unsigned int beta;
} estp_md_vals[] = {
 {0, 128, /* 0.500 */ },
 {1, 129, /* 0.502 */ },
 {2, 129, /* 0.505 */ },
 {3, 130, /* 0.507 */ },
 {4, 131, /* 0.510 */ },
 {5, 131, /* 0.512 */ },
 {6, 132, /* 0.515 */ },
 {7, 132, /* 0.517 */ },
 {8, 133, /* 0.520 */ },
 {9, 134, /* 0.522 */ },
 {10, 134, /* 0.525 */ },
 {11, 135, /* 0.527 */ },
 {12, 136, /* 0.530 */ },
 {13, 136, /* 0.532 */ },
 {14, 137, /* 0.535 */ },
 {15, 138, /* 0.537 */ },
 {16, 138, /* 0.540 */ },
 {17, 139, /* 0.542 */ },
 {18, 139, /* 0.545 */ },
 {19, 140, /* 0.547 */ },
 {20, 141, /* 0.550 */ },
 {21, 141, /* 0.552 */ },
 {22, 142, /* 0.555 */ },
 {23, 143, /* 0.557 */ },
 {24, 143, /* 0.560 */ },
 {25, 144, /* 0.562 */ },
 {26, 145, /* 0.565 */ },
 {27, 145, /* 0.567 */ },
 {28, 146, /* 0.570 */ },
 {29, 146, /* 0.572 */ },
 {30, 147, /* 0.574 */ },
 {31, 148, /* 0.577 */ },
 {32, 148, /* 0.579 */ },
 {33, 149, /* 0.582 */ },
 {34, 150, /* 0.584 */ },
 {35, 150, /* 0.587 */ },
 {36, 151, /* 0.589 */ },
 {37, 151, /* 0.591 */ },
 {38, 152, /* 0.594 */ },
 {39, 153, /* 0.596 */ },
 {40, 153, /* 0.599 */ },
 {41, 154, /* 0.601 */ },
 {42, 154, /* 0.603 */ },
 {43, 155, /* 0.606 */ },
 {44, 156, /* 0.608 */ },
 {45, 156, /* 0.611 */ },
 {46, 157, /* 0.613 */ },
 {47, 158, /* 0.615 */ },
 {48, 158, /* 0.618 */ },
 {49, 159, /* 0.620 */ },
 {50, 159, /* 0.622 */ },
 {51, 160, /* 0.625 */ },
 {52, 161, /* 0.627 */ },
 {53, 161, /* 0.629 */ },
 {54, 162, /* 0.632 */ },
 {55, 162, /* 0.634 */ },
 {56, 163, /* 0.636 */ },
 {57, 164, /* 0.639 */ },
 {58, 164, /* 0.641 */ },
 {59, 165, /* 0.643 */ },
 {60, 165, /* 0.646 */ },
```

```
{61, 166, /* 0.648 */ },
{62, 166, /* 0.650 */ },
{63, 167, /* 0.652 */ },
{64, 168, /* 0.655 */ },
{65, 168, /* 0.657 */ },
{66, 169, /* 0.659 */ },
{67, 169, /* 0.662 */ },
{68, 170, /* 0.664 */ },
{69, 170, /* 0.666 */ },
{70, 171, /* 0.668 */ },
{71, 172, /* 0.670 */ },
{72, 172, /* 0.673 */ },
{73, 173, /* 0.675 */ },
{74, 173, /* 0.677 */ },
{75, 174, /* 0.679 */ },
{76, 174, /* 0.681 */ },
{77, 175, /* 0.684 */ },
{78, 176, /* 0.686 */ },
{79, 176, /* 0.688 */ },
{80, 177, /* 0.690 */ },
{81, 177, /* 0.692 */ },
{82, 178, /* 0.694 */ },
{83, 178, /* 0.696 */ },
{84, 179, /* 0.698 */ },
{85, 179, /* 0.701 */ },
{86, 180, /* 0.703 */ },
{87, 180, /* 0.705 */ },
{88, 181, /* 0.707 */ },
{89, 181, /* 0.709 */ },
{90, 182, /* 0.711 */ },
{91, 183, /* 0.713 */ },
{92, 183, /* 0.715 */ },
{93, 184, /* 0.717 */ },
{94, 184, /* 0.719 */ },
{95, 185, /* 0.721 */ },
{96, 185, /* 0.723 */ },
{97, 186, /* 0.725 */ },
{98, 186, /* 0.727 */ },
{99, 187, /* 0.729 */ },
{100, 187, /* 0.731 */ },
{101, 188, /* 0.733 */ },
{102, 188, /* 0.735 */ },
{103, 189, /* 0.737 */ },
{104, 189, /* 0.739 */ },
{105, 190, /* 0.741 */ },
{106, 190, /* 0.743 */ },
{107, 191, /* 0.745 */ },
{108, 191, /* 0.746 */ },
{109, 192, /* 0.748 */ },
{110, 192, /* 0.750 */ },
{111, 193, /* 0.752 */ },
{112, 193, /* 0.754 */ },
{113, 193, /* 0.756 */ },
{114, 194, /* 0.758 */ },
{115, 194, /* 0.760 */ },
{116, 195, /* 0.761 */ },
{117, 195, /* 0.763 */ },
{118, 196, /* 0.765 */ },
{119, 196, /* 0.767 */ },
{120, 197, /* 0.769 */ },
{121, 197, /* 0.770 */ },
{122, 198, /* 0.772 */ },
{123, 198, /* 0.774 */ },
{124, 199, /* 0.776 */ },
{125, 199, /* 0.777 */ },
{126, 199, /* 0.779 */ },
{127, 200, /* 0.781 */ },
{128, 200, /* 0.782 */ },
{129, 201, /* 0.784 */ },
{130, 201, /* 0.786 */ },
{131, 202, /* 0.788 */ },
{132, 202, /* 0.789 */ },
{133, 202, /* 0.791 */ },
{134, 203, /* 0.792 */ },
{135, 203, /* 0.794 */ },
{136, 204, /* 0.796 */ },
{137, 204, /* 0.797 */ },
{138, 205, /* 0.799 */ },
{139, 205, /* 0.801 */ },
{140, 205, /* 0.802 */ },
{141, 206, /* 0.804 */ },
{142, 206, /* 0.805 */ },
{143, 207, /* 0.807 */ },
{144, 207, /* 0.808 */ },
{145, 207, /* 0.810 */ },
{146, 208, /* 0.812 */ },
{147, 208, /* 0.813 */ },
{148, 209, /* 0.815 */ },
{149, 209, /* 0.816 */ },
{150, 209, /* 0.818 */ },
```

```
{151, 210, /* 0.819 */ },
{152, 210, /* 0.821 */ },
{153, 210, /* 0.822 */ },
{154, 211, /* 0.823 */ },
{155, 211, /* 0.825 */ },
{156, 212, /* 0.826 */ },
{157, 212, /* 0.828 */ },
{158, 212, /* 0.829 */ },
{159, 213, /* 0.831 */ },
{160, 213, /* 0.832 */ },
{161, 213, /* 0.833 */ },
{162, 214, /* 0.835 */ },
{163, 214, /* 0.836 */ },
{164, 214, /* 0.838 */ },
{165, 215, /* 0.839 */ },
{166, 215, /* 0.840 */ },
{167, 215, /* 0.842 */ },
{168, 216, /* 0.843 */ },
{169, 216, /* 0.844 */ },
{170, 216, /* 0.846 */ },
{171, 217, /* 0.847 */ },
{172, 217, /* 0.848 */ },
{173, 217, /* 0.849 */ },
{174, 218, /* 0.851 */ },
{175, 218, /* 0.852 */ },
{176, 218, /* 0.853 */ },
{177, 219, /* 0.854 */ },
{178, 219, /* 0.856 */ },
{179, 219, /* 0.857 */ },
{180, 220, /* 0.858 */ },
{181, 220, /* 0.859 */ },
{182, 220, /* 0.861 */ },
{183, 221, /* 0.862 */ },
{184, 221, /* 0.863 */ },
{185, 221, /* 0.864 */ },
{186, 222, /* 0.865 */ },
{187, 222, /* 0.866 */ },
{188, 222, /* 0.868 */ },
{189, 222, /* 0.869 */ },
{190, 223, /* 0.870 */ },
{191, 223, /* 0.871 */ },
{192, 223, /* 0.872 */ },
{193, 224, /* 0.873 */ },
{194, 224, /* 0.874 */ },
{195, 224, /* 0.875 */ },
{196, 224, /* 0.877 */ },
{197, 225, /* 0.878 */ },
{198, 225, /* 0.879 */ },
{199, 225, /* 0.880 */ },
{200, 225, /* 0.881 */ },
{201, 226, /* 0.882 */ },
{202, 226, /* 0.883 */ },
{203, 226, /* 0.884 */ },
{204, 227, /* 0.885 */ },
{205, 227, /* 0.886 */ },
{206, 227, /* 0.887 */ },
{207, 227, /* 0.888 */ },
{208, 228, /* 0.889 */ },
{209, 228, /* 0.890 */ },
{210, 228, /* 0.891 */ },
{211, 228, /* 0.892 */ },
{212, 229, /* 0.893 */ },
{213, 229, /* 0.894 */ },
{214, 229, /* 0.895 */ },
{215, 229, /* 0.896 */ },
{216, 230, /* 0.897 */ },
{217, 230, /* 0.898 */ },
{218, 230, /* 0.898 */ },
{219, 230, /* 0.899 */ },
{220, 230, /* 0.900 */ },
{221, 231, /* 0.901 */ },
{222, 231, /* 0.902 */ },
{223, 231, /* 0.903 */ },
{224, 231, /* 0.904 */ },
{225, 232, /* 0.905 */ },
{226, 232, /* 0.906 */ },
{227, 232, /* 0.906 */ },
{228, 232, /* 0.907 */ },
{229, 232, /* 0.908 */ },
{230, 233, /* 0.909 */ },
{231, 233, /* 0.910 */ },
{232, 233, /* 0.911 */ },
{233, 233, /* 0.911 */ },
{234, 234, /* 0.912 */ },
{235, 234, /* 0.913 */ },
{236, 234, /* 0.914 */ },
{237, 234, /* 0.915 */ },
{238, 234, /* 0.915 */ },
{239, 235, /* 0.916 */ },
{240, 235, /* 0.917 */ },
```

```
{241, 235, /* 0.918 */ },
{242, 235, /* 0.918 */ },
{243, 235, /* 0.919 */ },
{244, 235, /* 0.920 */ },
{245, 236, /* 0.921 */ },
{246, 236, /* 0.921 */ },
{247, 236, /* 0.922 */ },
{248, 236, /* 0.923 */ },
{249, 236, /* 0.923 */ },
{250, 237, /* 0.924 */ },
{251, 237, /* 0.925 */ },
{252, 237, /* 0.926 */ },
{253, 237, /* 0.926 */ },
{254, 237, /* 0.927 */ },
{255, 237, /* 0.928 */ },
{256, 238, /* 0.928 */ },
{257, 238, /* 0.929 */ },
{258, 238, /* 0.930 */ },
{259, 238, /* 0.930 */ },
{260, 238, /* 0.931 */ },
{261, 238, /* 0.932 */ },
{262, 239, /* 0.932 */ },
{263, 239, /* 0.933 */ },
{264, 239, /* 0.933 */ },
{265, 239, /* 0.934 */ },
{266, 239, /* 0.935 */ },
{267, 239, /* 0.935 */ },
{268, 240, /* 0.936 */ },
{269, 240, /* 0.936 */ },
{270, 240, /* 0.937 */ },
{271, 240, /* 0.938 */ },
{272, 240, /* 0.938 */ },
{273, 240, /* 0.939 */ },
{274, 240, /* 0.939 */ },
{275, 241, /* 0.940 */ },
{276, 241, /* 0.940 */ },
{277, 241, /* 0.941 */ },
{278, 241, /* 0.942 */ },
{279, 241, /* 0.942 */ },
{280, 241, /* 0.943 */ },
{281, 241, /* 0.943 */ },
{282, 242, /* 0.944 */ },
{283, 242, /* 0.944 */ },
{284, 242, /* 0.945 */ },
{285, 242, /* 0.945 */ },
{286, 242, /* 0.946 */ },
{287, 242, /* 0.946 */ },
{288, 242, /* 0.947 */ },
{289, 243, /* 0.947 */ },
{290, 243, /* 0.948 */ },
{291, 243, /* 0.948 */ },
{292, 243, /* 0.949 */ },
{293, 243, /* 0.949 */ },
{294, 243, /* 0.950 */ },
{295, 243, /* 0.950 */ },
{296, 243, /* 0.951 */ },
{297, 244, /* 0.951 */ },
{298, 244, /* 0.952 */ },
{299, 244, /* 0.952 */ },
{300, 244, /* 0.953 */ },
{301, 244, /* 0.953 */ },
{302, 244, /* 0.953 */ },
{303, 244, /* 0.954 */ },
{304, 244, /* 0.954 */ },
{305, 244, /* 0.955 */ },
{306, 245, /* 0.955 */ },
{307, 245, /* 0.956 */ },
{308, 245, /* 0.956 */ },
{309, 245, /* 0.956 */ },
{310, 245, /* 0.957 */ },
{311, 245, /* 0.957 */ },
{312, 245, /* 0.958 */ },
{313, 245, /* 0.958 */ },
{314, 245, /* 0.959 */ },
{315, 245, /* 0.959 */ },
{316, 246, /* 0.959 */ },
{317, 246, /* 0.960 */ },
{318, 246, /* 0.960 */ },
{319, 246, /* 0.960 */ },
{320, 246, /* 0.961 */ },
{321, 246, /* 0.961 */ },
{322, 246, /* 0.962 */ },
{323, 246, /* 0.962 */ },
{324, 246, /* 0.962 */ },
{325, 246, /* 0.963 */ },
{326, 247, /* 0.963 */ },
{327, 247, /* 0.963 */ },
{328, 247, /* 0.964 */ },
{329, 247, /* 0.964 */ },
{330, 247, /* 0.964 */ },
```

```
{331, 247, /* 0.965 */ },
{332, 247, /* 0.965 */ },
{333, 247, /* 0.965 */ },
{334, 247, /* 0.966 */ },
{335, 247, /* 0.966 */ },
{336, 247, /* 0.966 */ },
{337, 247, /* 0.967 */ },
{338, 248, /* 0.967 */ },
{339, 248, /* 0.967 */ },
{340, 248, /* 0.968 */ },
{341, 248, /* 0.968 */ },
{342, 248, /* 0.968 */ },
{343, 248, /* 0.969 */ },
{344, 248, /* 0.969 */ },
{345, 248, /* 0.969 */ },
{346, 248, /* 0.970 */ },
{347, 248, /* 0.970 */ },
{348, 248, /* 0.970 */ },
{349, 248, /* 0.970 */ },
{350, 248, /* 0.971 */ },
{351, 249, /* 0.971 */ },
{352, 249, /* 0.971 */ },
{353, 249, /* 0.972 */ },
{354, 249, /* 0.972 */ },
{355, 249, /* 0.972 */ },
{356, 249, /* 0.972 */ },
{357, 249, /* 0.973 */ },
{358, 249, /* 0.973 */ },
{359, 249, /* 0.973 */ },
{360, 249, /* 0.973 */ },
{361, 249, /* 0.974 */ },
{362, 249, /* 0.974 */ },
{363, 249, /* 0.974 */ },
{364, 249, /* 0.974 */ },
{365, 250, /* 0.975 */ },
{366, 250, /* 0.975 */ },
{367, 250, /* 0.975 */ },
{368, 250, /* 0.975 */ },
{369, 250, /* 0.976 */ },
{370, 250, /* 0.976 */ },
{371, 250, /* 0.976 */ },
{372, 250, /* 0.976 */ },
{373, 250, /* 0.977 */ },
{374, 250, /* 0.977 */ },
{375, 250, /* 0.977 */ },
{376, 250, /* 0.977 */ },
{377, 250, /* 0.977 */ },
{378, 250, /* 0.978 */ },
{379, 250, /* 0.978 */ },
{380, 250, /* 0.978 */ },
{381, 250, /* 0.978 */ },
{382, 251, /* 0.979 */ },
{383, 251, /* 0.979 */ },
{384, 251, /* 0.979 */ },
{385, 251, /* 0.979 */ },
{386, 251, /* 0.979 */ },
{387, 251, /* 0.980 */ },
{388, 251, /* 0.980 */ },
{389, 251, /* 0.980 */ },
{390, 251, /* 0.980 */ },
{391, 251, /* 0.980 */ },
{392, 251, /* 0.981 */ },
{393, 251, /* 0.981 */ },
{394, 251, /* 0.981 */ },
{395, 251, /* 0.981 */ },
{396, 251, /* 0.981 */ },
{397, 251, /* 0.981 */ },
{398, 251, /* 0.982 */ },
{399, 251, /* 0.982 */ },
{400, 251, /* 0.982 */ },
{401, 251, /* 0.982 */ },
{402, 251, /* 0.982 */ },
{403, 252, /* 0.983 */ },
{404, 252, /* 0.983 */ },
{405, 252, /* 0.983 */ },
{406, 252, /* 0.983 */ },
{407, 252, /* 0.983 */ },
{408, 252, /* 0.983 */ },
{409, 252, /* 0.984 */ },
{410, 252, /* 0.984 */ },
{411, 252, /* 0.984 */ },
{412, 252, /* 0.984 */ },
{413, 252, /* 0.984 */ },
{414, 252, /* 0.984 */ },
{415, 252, /* 0.984 */ },
{416, 252, /* 0.985 */ },
{417, 252, /* 0.985 */ },
{418, 252, /* 0.985 */ },
{419, 252, /* 0.985 */ },
{420, 252, /* 0.985 */ },
```

```
{421, 252, /* 0.985 */ },
{422, 252, /* 0.986 */ },
{423, 252, /* 0.986 */ },
{424, 252, /* 0.986 */ },
{425, 252, /* 0.986 */ },
{426, 252, /* 0.986 */ },
{427, 252, /* 0.986 */ },
{428, 253, /* 0.986 */ },
{429, 253, /* 0.986 */ },
{430, 253, /* 0.987 */ },
{431, 253, /* 0.987 */ },
{432, 253, /* 0.987 */ },
{433, 253, /* 0.987 */ },
{434, 253, /* 0.987 */ },
{435, 253, /* 0.987 */ },
{436, 253, /* 0.987 */ },
{437, 253, /* 0.988 */ },
{438, 253, /* 0.988 */ },
{439, 253, /* 0.988 */ },
{440, 253, /* 0.988 */ },
{441, 253, /* 0.988 */ },
{442, 253, /* 0.988 */ },
{443, 253, /* 0.988 */ },
{444, 253, /* 0.988 */ },
{445, 253, /* 0.988 */ },
{446, 253, /* 0.989 */ },
{447, 253, /* 0.989 */ },
{448, 253, /* 0.989 */ },
{449, 253, /* 0.989 */ },
{450, 253, /* 0.989 */ },
{451, 253, /* 0.989 */ },
{452, 253, /* 0.989 */ },
{453, 253, /* 0.989 */ },
{454, 253, /* 0.989 */ },
{455, 253, /* 0.990 */ },
{456, 253, /* 0.990 */ },
{457, 253, /* 0.990 */ },
{458, 253, /* 0.990 */ },
{459, 253, /* 0.990 */ },
{460, 253, /* 0.990 */ },
{461, 253, /* 0.990 */ },
{462, 254, /* 0.990 */ },
{463, 254, /* 0.990 */ },
{464, 254, /* 0.990 */ },
{465, 254, /* 0.991 */ },
{466, 254, /* 0.991 */ },
{467, 254, /* 0.991 */ },
{468, 254, /* 0.991 */ },
{469, 254, /* 0.991 */ },
{470, 254, /* 0.991 */ },
{471, 254, /* 0.991 */ },
{472, 254, /* 0.991 */ },
{473, 254, /* 0.991 */ },
{474, 254, /* 0.991 */ },
{475, 254, /* 0.991 */ },
{476, 254, /* 0.992 */ },
{477, 254, /* 0.992 */ },
{478, 254, /* 0.992 */ },
{479, 254, /* 0.992 */ },
{480, 254, /* 0.992 */ },
{481, 254, /* 0.992 */ },
{482, 254, /* 0.992 */ },
{483, 254, /* 0.992 */ },
{484, 254, /* 0.992 */ },
{485, 254, /* 0.992 */ },
{486, 254, /* 0.992 */ },
{487, 254, /* 0.992 */ },
{488, 254, /* 0.992 */ },
{489, 254, /* 0.993 */ },
{490, 254, /* 0.993 */ },
{491, 254, /* 0.993 */ },
{492, 254, /* 0.993 */ },
{493, 254, /* 0.993 */ },
{494, 254, /* 0.993 */ },
{495, 254, /* 0.993 */ },
{496, 254, /* 0.993 */ },
{497, 254, /* 0.993 */ },
{498, 254, /* 0.993 */ },
{499, 254, /* 0.993 */ },
{500, 254, /* 0.993 */ },
{501, 254, /* 0.993 */ },
{502, 254, /* 0.993 */ },
{503, 254, /* 0.994 */ },
{504, 254, /* 0.994 */ },
{505, 254, /* 0.994 */ },
{506, 254, /* 0.994 */ },
{507, 254, /* 0.994 */ },
{508, 254, /* 0.994 */ },
{509, 254, /* 0.994 */ },
{510, 254, /* 0.994 */ },
```

```
 {511, 254, /* 0.994 */ },
 {512, 254, /* 0.994 */ },
 {513, 254, /* 0.994 */ },
 {514, 255, /* 0.994 */ },
 {515, 255, /* 0.994 */ },
 {516, 255, /* 0.994 */ },
 {517, 255, /* 0.994 */ },
 {518, 255, /* 0.994 */ },
 {519, 255, /* 0.994 */ },
 {520, 255, /* 0.995 */ },
};

#define ESTP_MD_MAX       ARRAY_SIZE(estp_md_vals)

struct estp {
    u32  ai;       //indice fila de matriz estp_md_vals
    u32  count;    //contador de paquetes transmitidos
satisfactoriamente entre dos perdidas
    u32  tau;      // "tau"
    u32  exp;      //equivalente a 100*(alfa-1)/(tau)
    u32  dec;      //256*beta
    u32  delta;    //valor del delta actual (count +1)
    u32  seq_i;    //numero de secuencia inicial entre dos
perdidas
    u32  seq_f;    //numero de secuencia final entre dos
perdidas
    u32  packet_size;     //tamano de segmento
    u32  to_indicator;    //indicador de <<timeout>>
    u32  n_loss;          //numero de perdidas
    u32  sum_delta;       //suma de deltas acumulativa
    u32  loss_indicator;  //indicador de perdida
    u32  send_packet;     //suma de deltas
    u32  tmp_seq;
};

static void estp_init(struct sock *sk)
{
    struct tcp_sock *tp = tcp_sk(sk);
    struct estp *ca = inet_csk_ca(sk);
    tp->snd_cwnd = 1;
    ca->ai = 0;
    ca->delta = 0;
    ca->tau = 100;
    ca->count=0;
    ca->seq_i= tp->snd_una;
    ca->to_indicator=0;
    ca->send_packet=0;
    ca->n_loss=1;
    ca->sum_delta=0;
    ca->loss_indicator=0;
    /* Asegurar la fase de decremento multiplicativo,
estableciendo snd_cwnd_clamp como un valor muy grande*/
    tp->snd_cwnd_clamp = min_t(u32, tp->snd_cwnd_clamp,
0xffffffff/128);
}

static void estp_cong_avoid(struct sock *sk, u32 ack, u32
acked, u32 in_flight)
{
    struct tcp_sock *tp = tcp_sk(sk);
    struct estp *ca = inet_csk_ca(sk);

    if (ca->loss_indicator == 1){
       ca->n_loss++;
       ca->sum_delta = ca->sum_delta + ca->delta;
       ca->loss_indicator=0;
    }
    if (!tcp_is_cwnd_limited(sk, in_flight))
       return;
    if (tp->snd_cwnd <= tp->snd_ssthresh){
       tcp_slow_start(tp, acked);
    }
    else {

       /* Incremento aditivo */
       if (tp->snd_cwnd < tp->snd_cwnd_clamp) {
              tp->snd_cwnd_cnt++;
              if (tp->snd_cwnd_cnt >= tp->snd_cwnd) {
                     tp->snd_cwnd_cnt -= tp->snd_cwnd;
                     tp->snd_cwnd++;
              }
       }
    }
    ca->seq_f = tp->snd_una;
    ca->packet_size = tp->mss_cache;
    /* Parametro delta */
    ca->count = ((ca->seq_f - ca->seq_i)/ca->packet_size);
    ca->delta = ca->count +1 ;
    /*Suma de valores de delta*/
    ca->send_packet = ca->sum_delta + ca->delta;
```

```
    /* Valor de  tau */
    ca->tau = (ca->send_packet/ca->n_loss)+1;   //omitir
para emplear tau fijo, y utilizar siguiente linea
    //ca->tau = 100;
    /* Calculo de exponente */
    ca->exp = (100 * ca->count)/ (ca->tau);

    //Determinar el ca->ai
    if (ca->exp > estp_md_vals[ca->ai].exp_ref && ca->ai <
ESTP_MD_MAX - 1){
       while (ca->exp > estp_md_vals[ca->ai].exp_ref)
             ca->ai++;
    }
    if (ca->exp < estp_md_vals[ca->ai].exp_ref && ca->ai <
ESTP_MD_MAX -1){
       while (ca->exp < estp_md_vals[ca->ai].exp_ref)
             ca->ai--;
    }
    if (ca->exp > 520)
       ca->ai = ESTP_MD_MAX -1;

    ca->dec = estp_md_vals[ca->ai].beta;
}

static u32 estp_ssthresh(struct sock *sk)
{
    struct tcp_sock *tp = tcp_sk(sk);
    struct estp *ca = inet_csk_ca(sk);

    ca->seq_i=ca->seq_f;
    ca->count=0;
    ca->loss_indicator = 1;
    ca->to_indicator = 0;
    /* Hace decremento multiplicativo */
    return max(((tp->snd_cwnd * ca->dec + 128) >> 8), 2U);
}

static void estp_state(struct sock *sk, u8 new_state)
{
    struct tcp_sock *tp = tcp_sk(sk);
    struct estp *ca = inet_csk_ca(sk);
//Si ocurre una perdida por <<timeout>>:ca->to_indicator=1
         if (new_state == TCP_CA_Loss){
              ca->count = 0;
       ca->loss_indicator = 0;
       ca->n_loss = 1;
       ca->delta = 0;
       ca->sum_delta = 0;
       ca->to_indicator = 1;
        }
    if (ca->to_indicator == 1){
       ca->seq_i = tp->snd_una;
    }
    if (new_state == TCP_CA_Open){
       ca->to_indicator = 0;
    }
}

static struct tcp_congestion_ops tcp_estp __read_mostly = {
    .init           = estp_init,
    .ssthresh= estp_ssthresh,
    .cong_avoid     = estp_cong_avoid,
    .min_cwnd= tcp_reno_min_cwnd,
    .set_state      = estp_state,

    .owner          = THIS_MODULE,
    .name           = "estp"
};

static int __init estp_register(void)
{
    BUILD_BUG_ON(sizeof(struct estp) > ICSK_CA_PRIV_SIZE);
    return tcp_register_congestion_control(&tcp_estp);
}
static void __exit estp_unregister(void)
{
    tcp_unregister_congestion_control(&tcp_estp);
}

module_init(estp_register);
module_exit(estp_unregister);

MODULE_AUTHOR("ESTP");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("ESTP");
```

ANEXO “K”

ANÁLISIS DE TCP RENO CON MATLAB

1. TRANSMISIÓN PUNTO A PUNTO

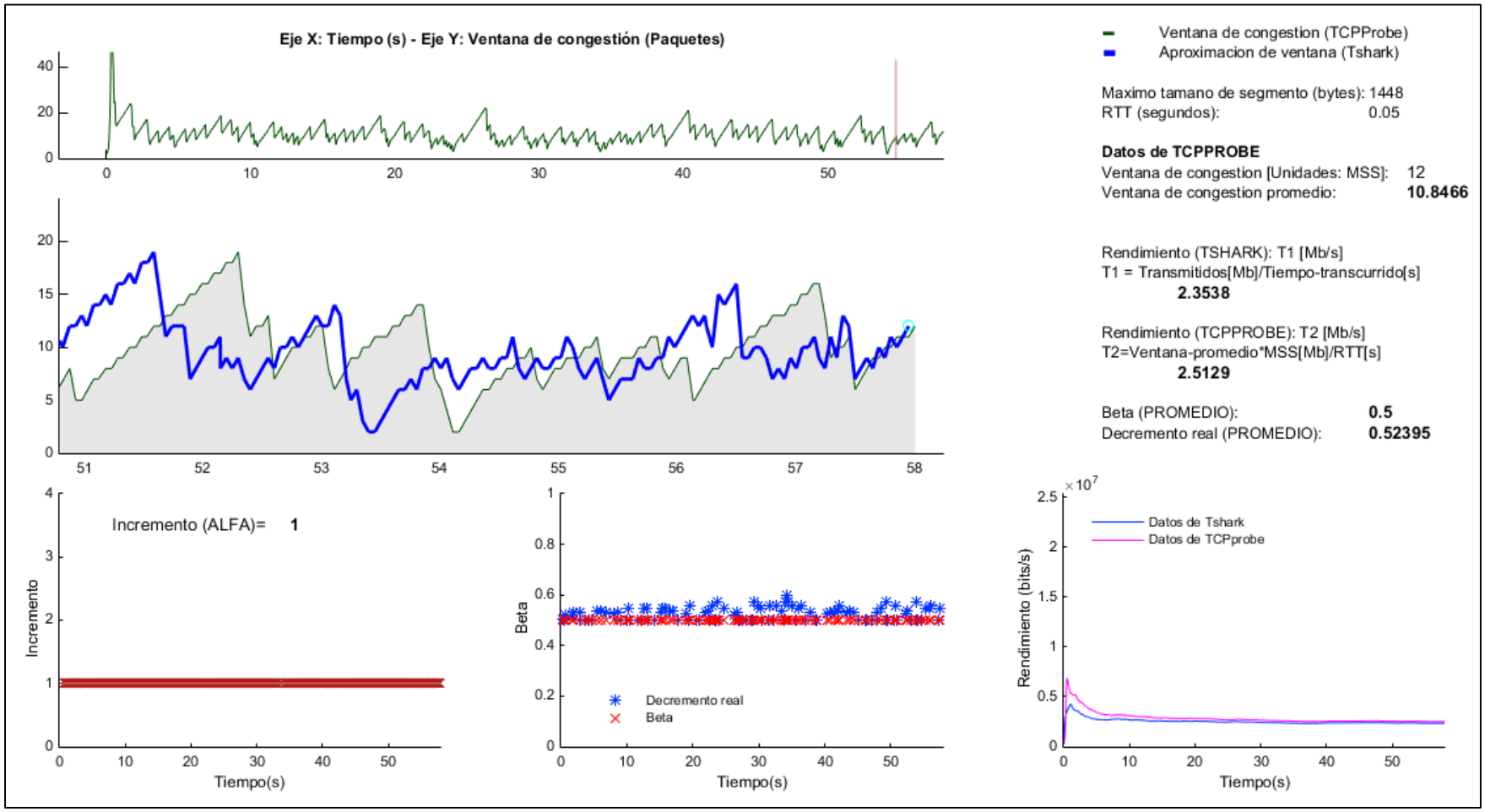


Figura K1. Transmisión empleando TCP RENO con RTT=50ms y Proporción de pérdidas = 1% (Pérdidas aleatorias)

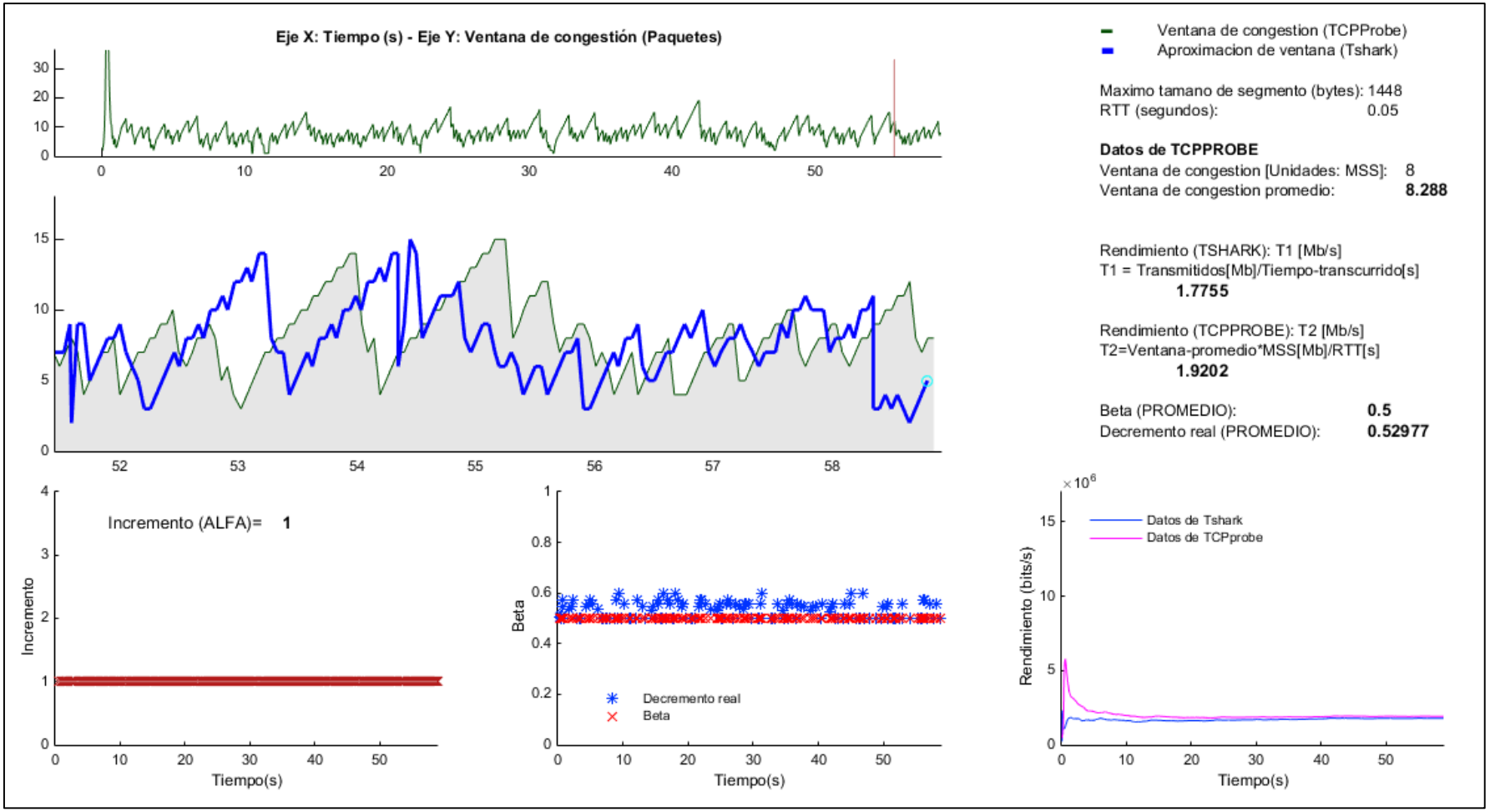


Figura K2. Transmisión empleando TCP RENO con RTT=50ms y Proporción de pérdidas = 2% (Pérdidas aleatorias)

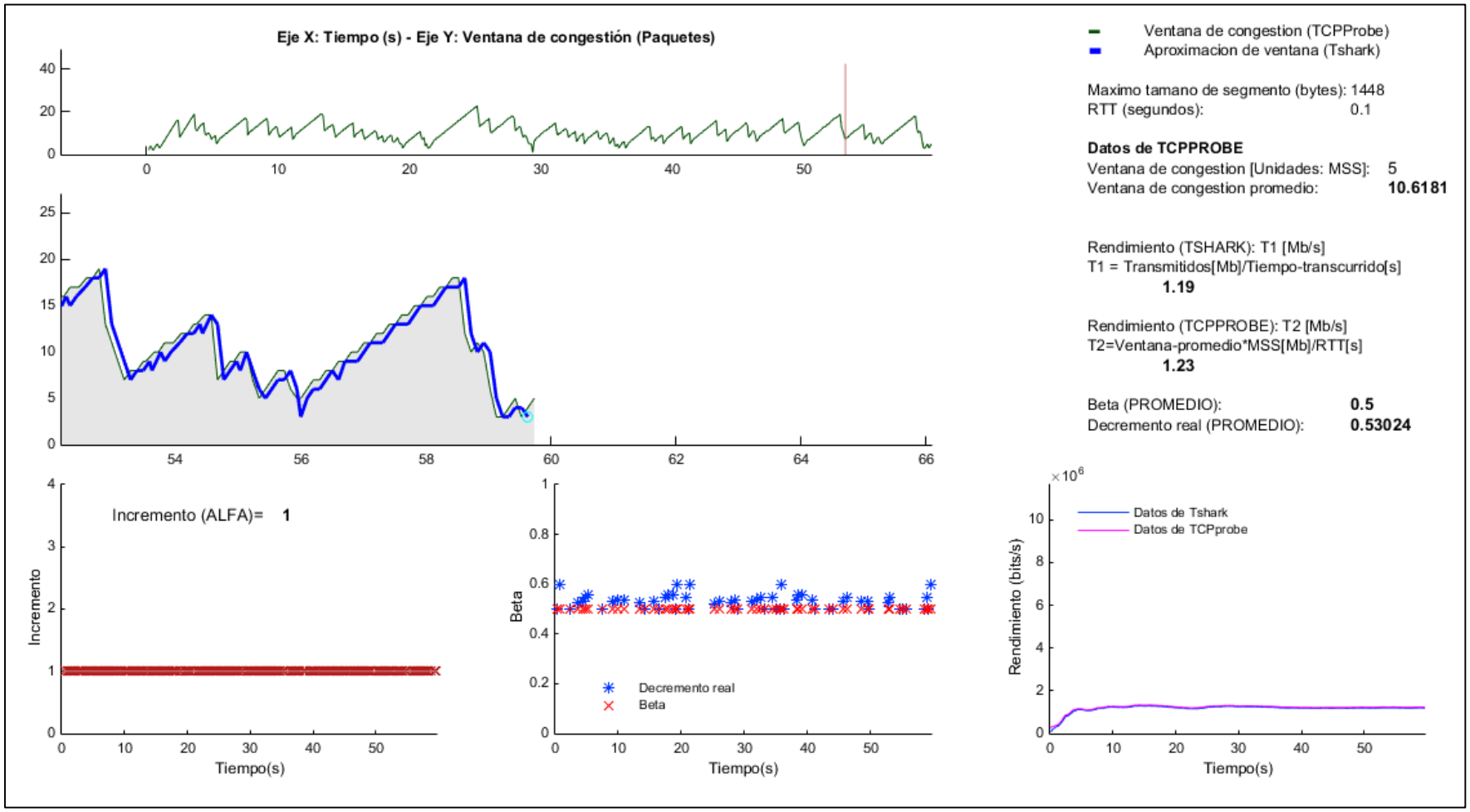


Figura K3. Transmisión empleando TCP RENO con RTT=100ms y Proporción de pérdidas = 1% (Pérdidas aleatorias)

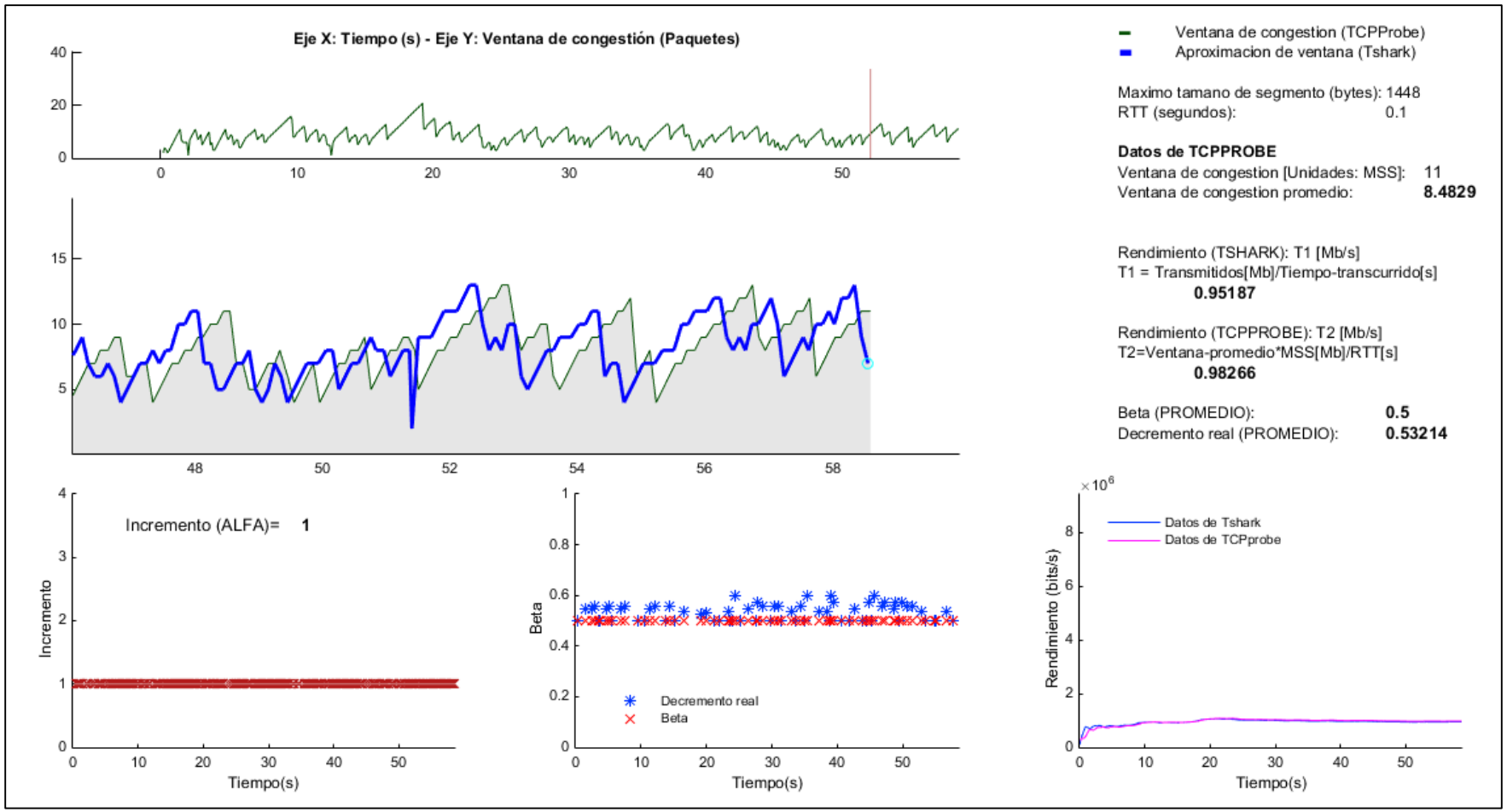
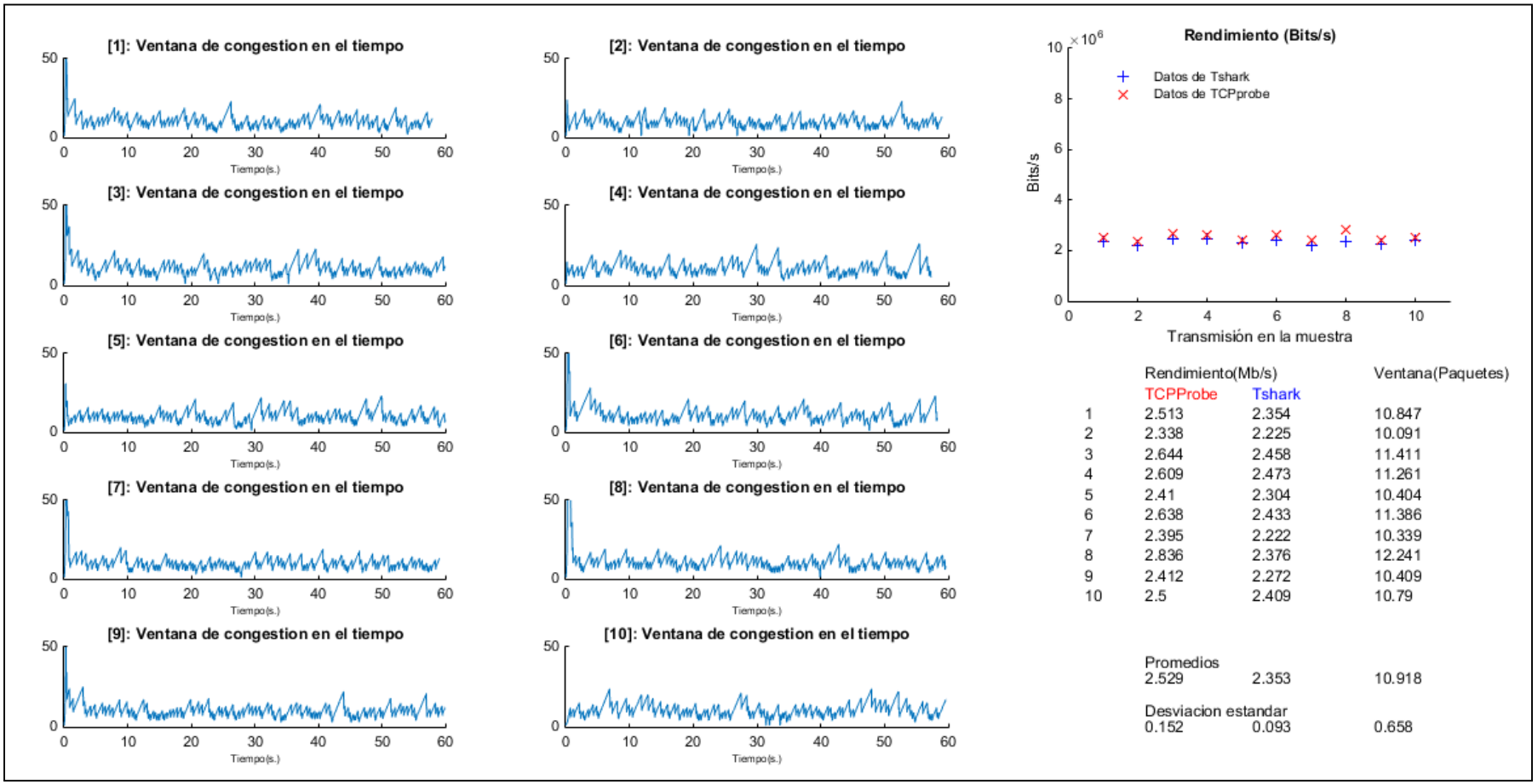


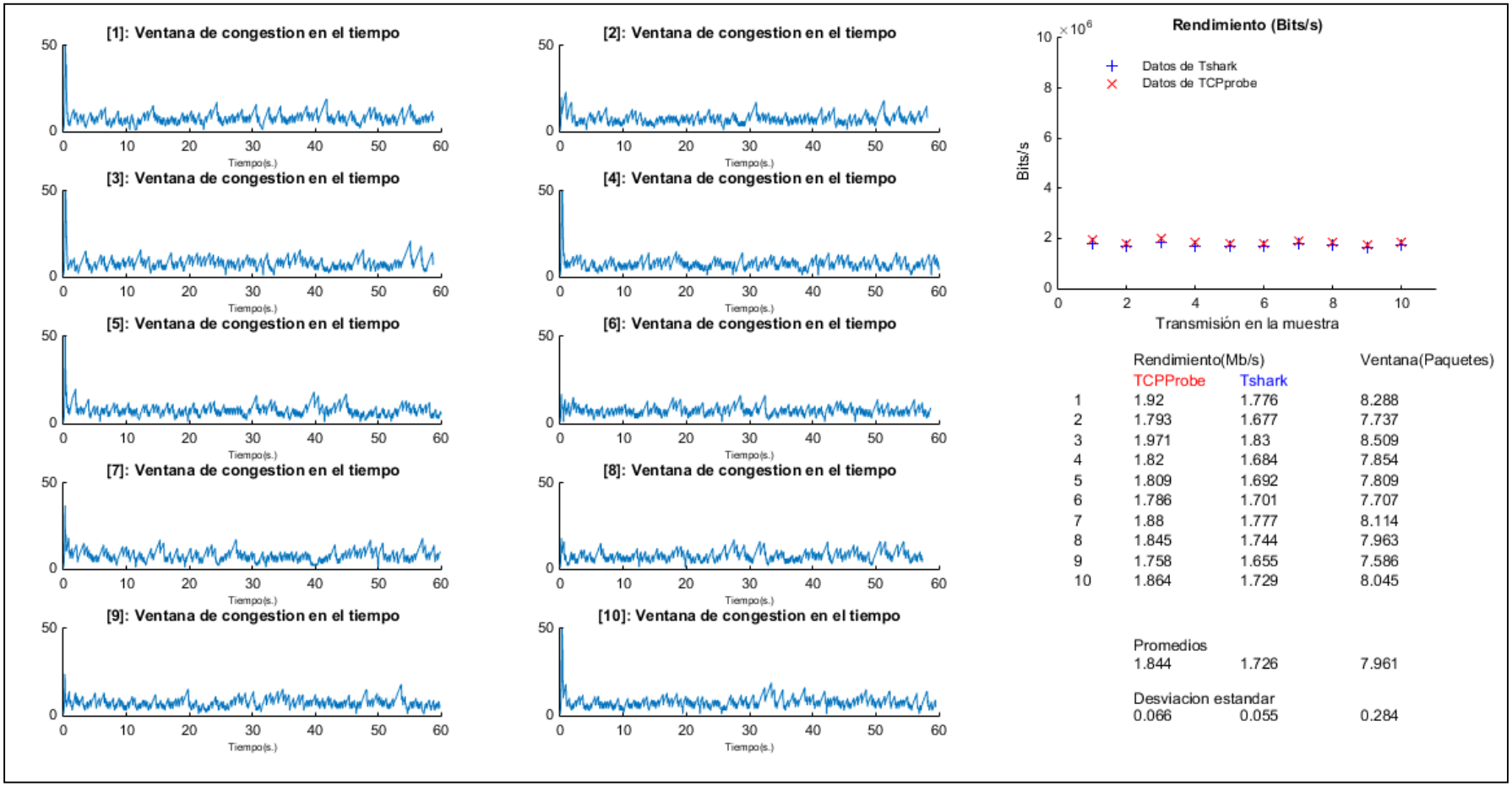
Figura K4. Transmisión empleando TCP RENO con RTT=100ms y Proporción de pérdidas = 2% (Pérdidas aleatorias)

2. MUESTRAS DE 10 TRANSMISIONES EMPLEANDO LOS MISMOS PARÁMETROS



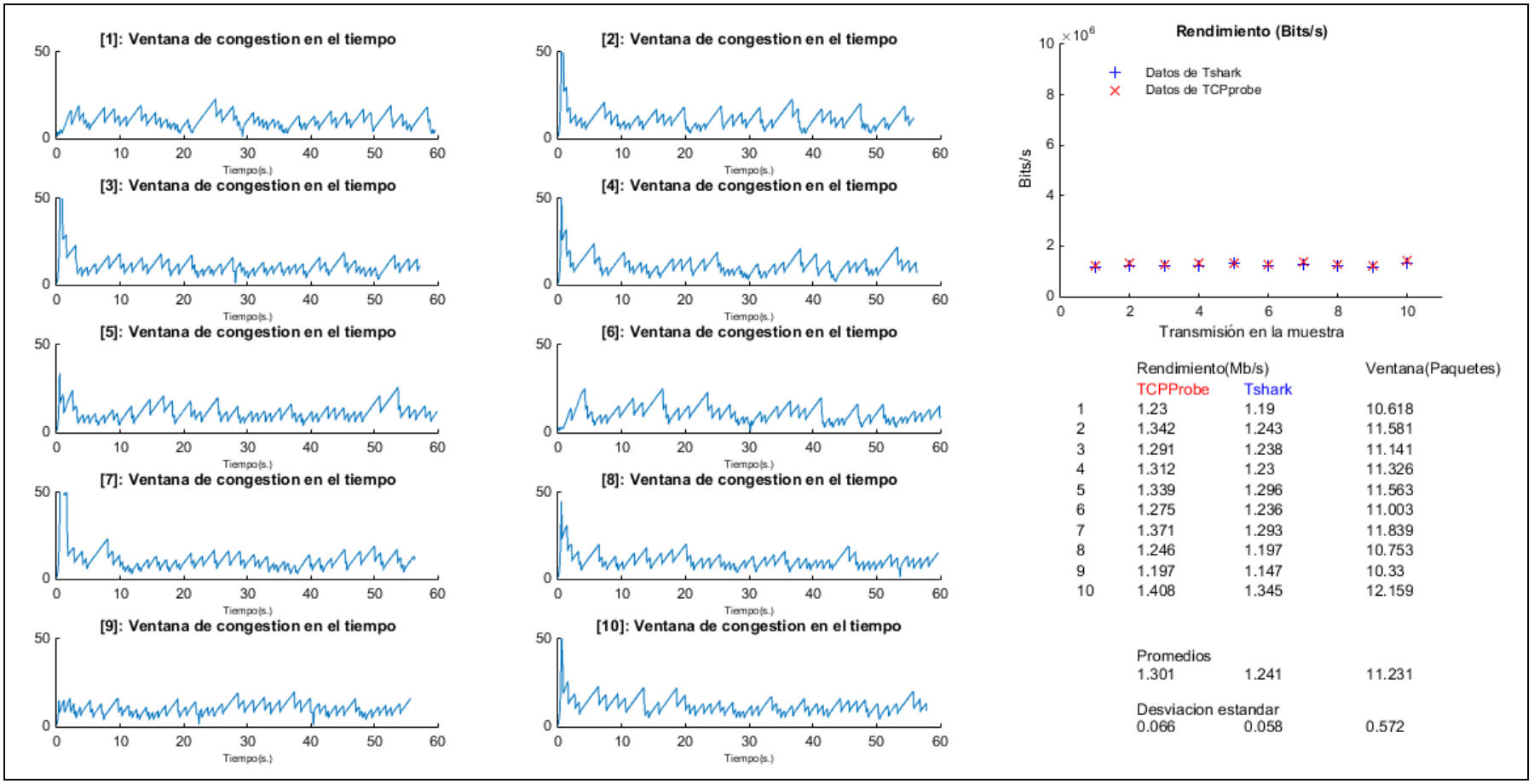
	Rendimiento(Mb/s)		Ventana(Paquetes)
	TCPProbe	Tshark	
1	2.513	2.354	10.847
2	2.338	2.225	10.091
3	2.644	2.458	11.411
4	2.609	2.473	11.261
5	2.41	2.304	10.404
6	2.638	2.433	11.386
7	2.395	2.222	10.339
8	2.836	2.376	12.241
9	2.412	2.272	10.409
10	2.5	2.409	10.79
Promedios			
	2.529	2.353	10.918
Desviacion estandar			
	0.152	0.093	0.658

Figura K5. Ventana de congestión y rendimiento empleando TCP RENO con RTT=50ms y Proporción de pérdidas =1%



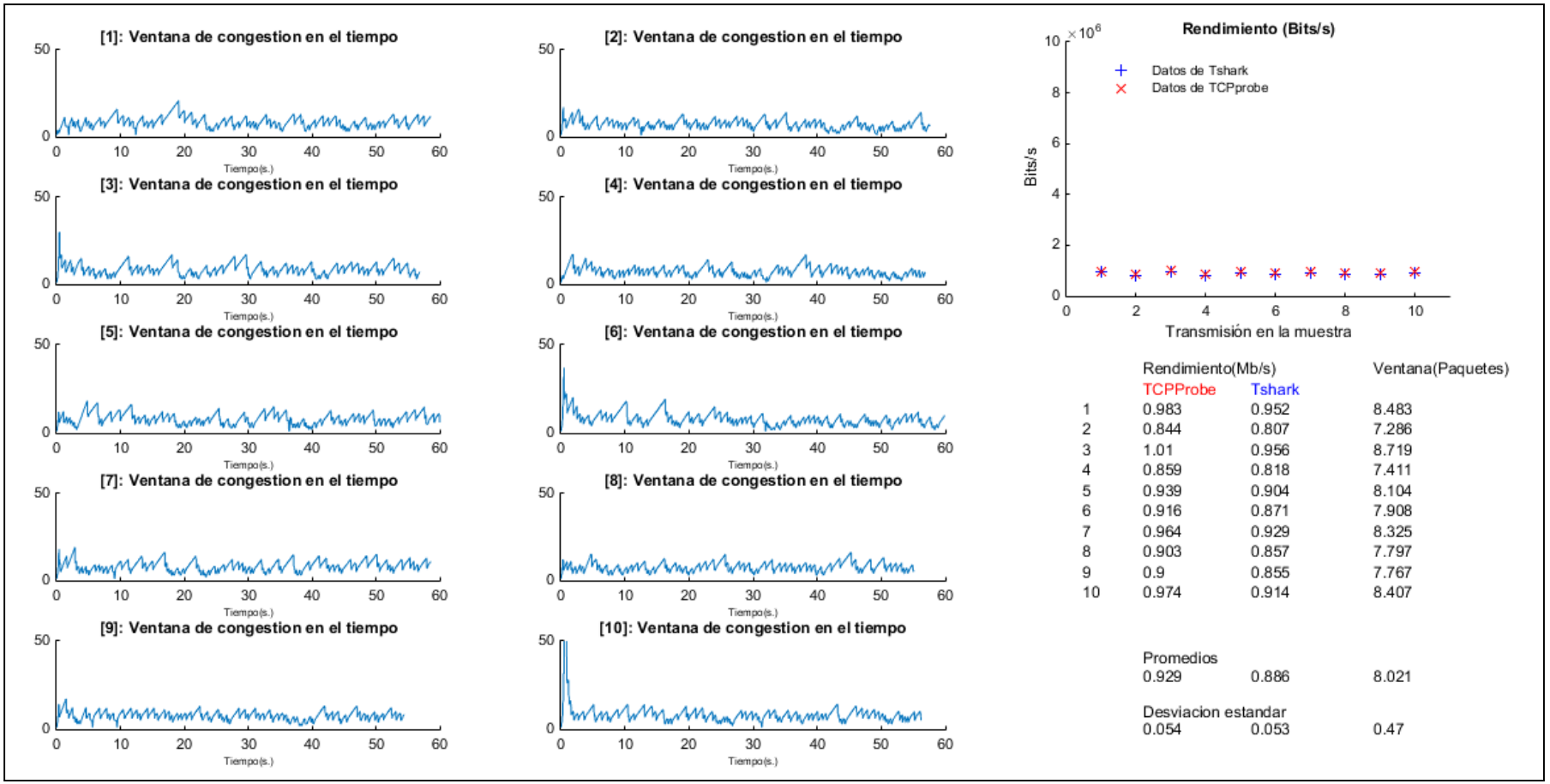
	Rendimiento(Mb/s)		Ventana(Paquetes)
	TCPProbe	Tshark	
1	1.92	1.776	8.288
2	1.793	1.677	7.737
3	1.971	1.83	8.509
4	1.82	1.684	7.854
5	1.809	1.692	7.809
6	1.786	1.701	7.707
7	1.88	1.777	8.114
8	1.845	1.744	7.963
9	1.758	1.655	7.586
10	1.864	1.729	8.045
Promedios			
	1.844	1.726	7.961
Desviacion estandar			
	0.066	0.055	0.284

Figura K6. Ventana de congestión y rendimiento empleando TCP RENO con RTT=50ms y Proporción de pérdidas =2%



	Rendimiento(Mb/s)		Ventana(Paquetes)
	TCPProbe	Tshark	
1	1.23	1.19	10.618
2	1.342	1.243	11.581
3	1.291	1.238	11.141
4	1.312	1.23	11.326
5	1.339	1.296	11.563
6	1.275	1.236	11.003
7	1.371	1.293	11.839
8	1.246	1.197	10.753
9	1.197	1.147	10.33
10	1.408	1.345	12.159
Promedios			
	1.301	1.241	11.231
Desviacion estandar			
	0.066	0.058	0.572

Figura K7. Ventana de congestión y rendimiento empleando TCP RENO con RTT=100ms y Proporción de pérdidas =1%



	Rendimiento(Mb/s)		Ventana(Paquetes)
	TCPProbe	Tshark	
1	0.983	0.952	8.483
2	0.844	0.807	7.286
3	1.01	0.956	8.719
4	0.859	0.818	7.411
5	0.939	0.904	8.104
6	0.916	0.871	7.908
7	0.964	0.929	8.325
8	0.903	0.857	7.797
9	0.9	0.855	7.767
10	0.974	0.914	8.407
Promedios			
	0.929	0.886	8.021
Desviacion estandar			
	0.054	0.053	0.47

Figura K8. Ventana de congestión y rendimiento empleando TCP RENO con RTT=100ms y Proporción de pérdidas =2%

ANEXO “L”
ANÁLISIS DE TCP GAIMD (α = 0.3125, β=0.875) CON MATLAB

1. TRANSMISIÓN PUNTO A PUNTO

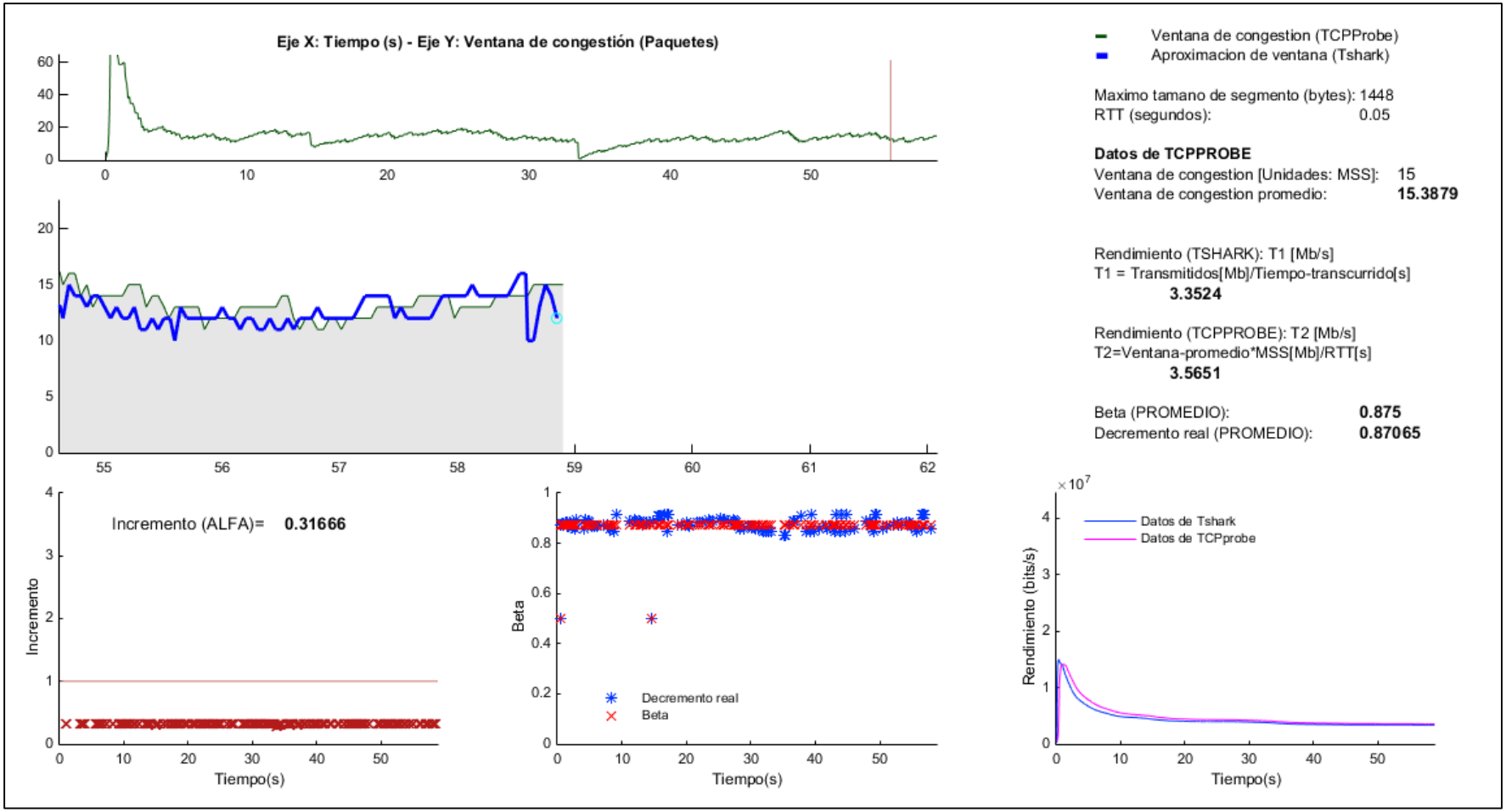


Figura L1. Transmisión empleando GAIMD con RTT=50ms y Proporción de pérdidas = 1% (Pérdidas aleatorias)

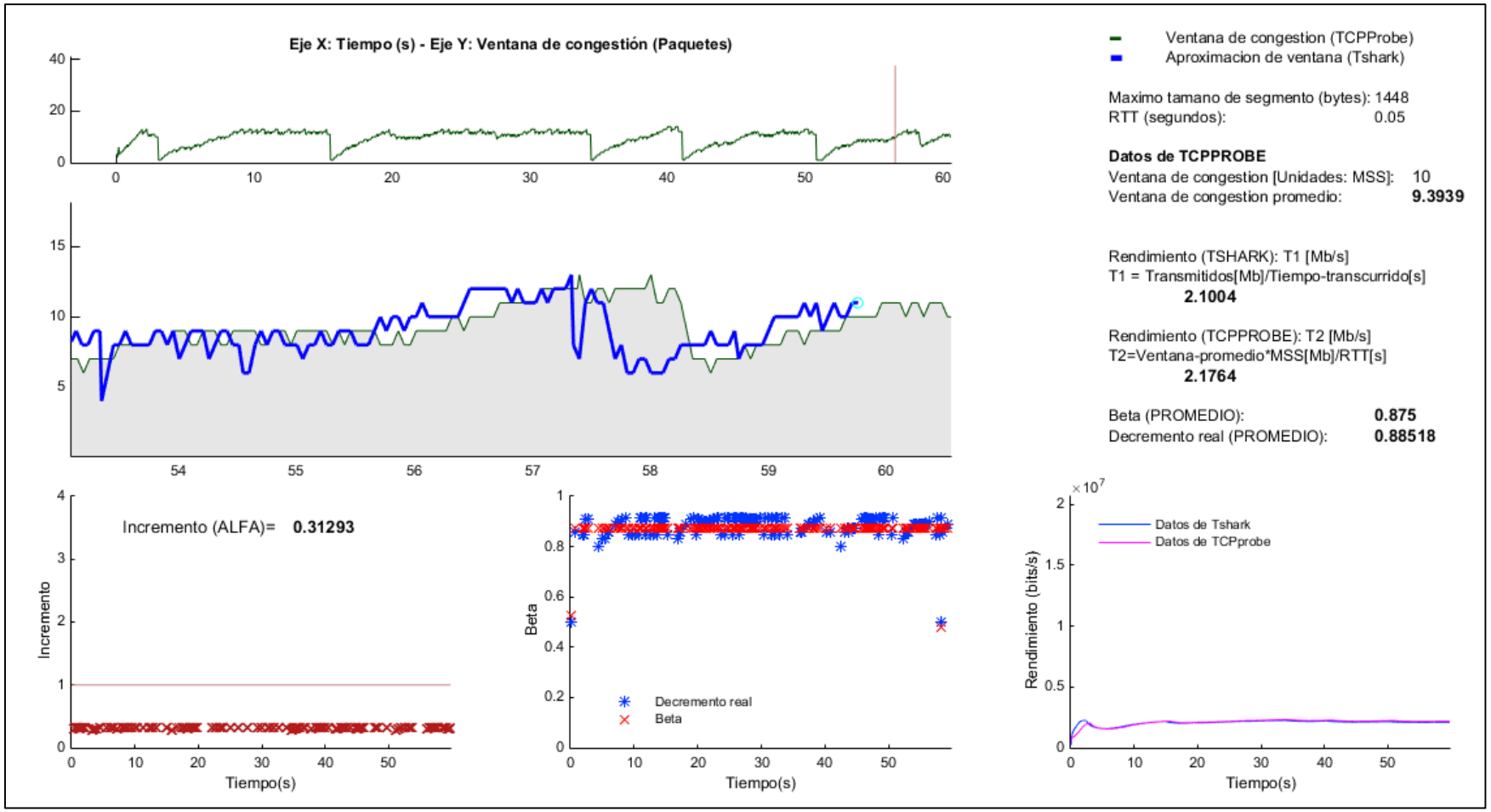


Figura L2. Transmisión empleando GAIMD con RTT=50ms y Proporción de pérdidas = 2% (Pérdidas aleatorias)

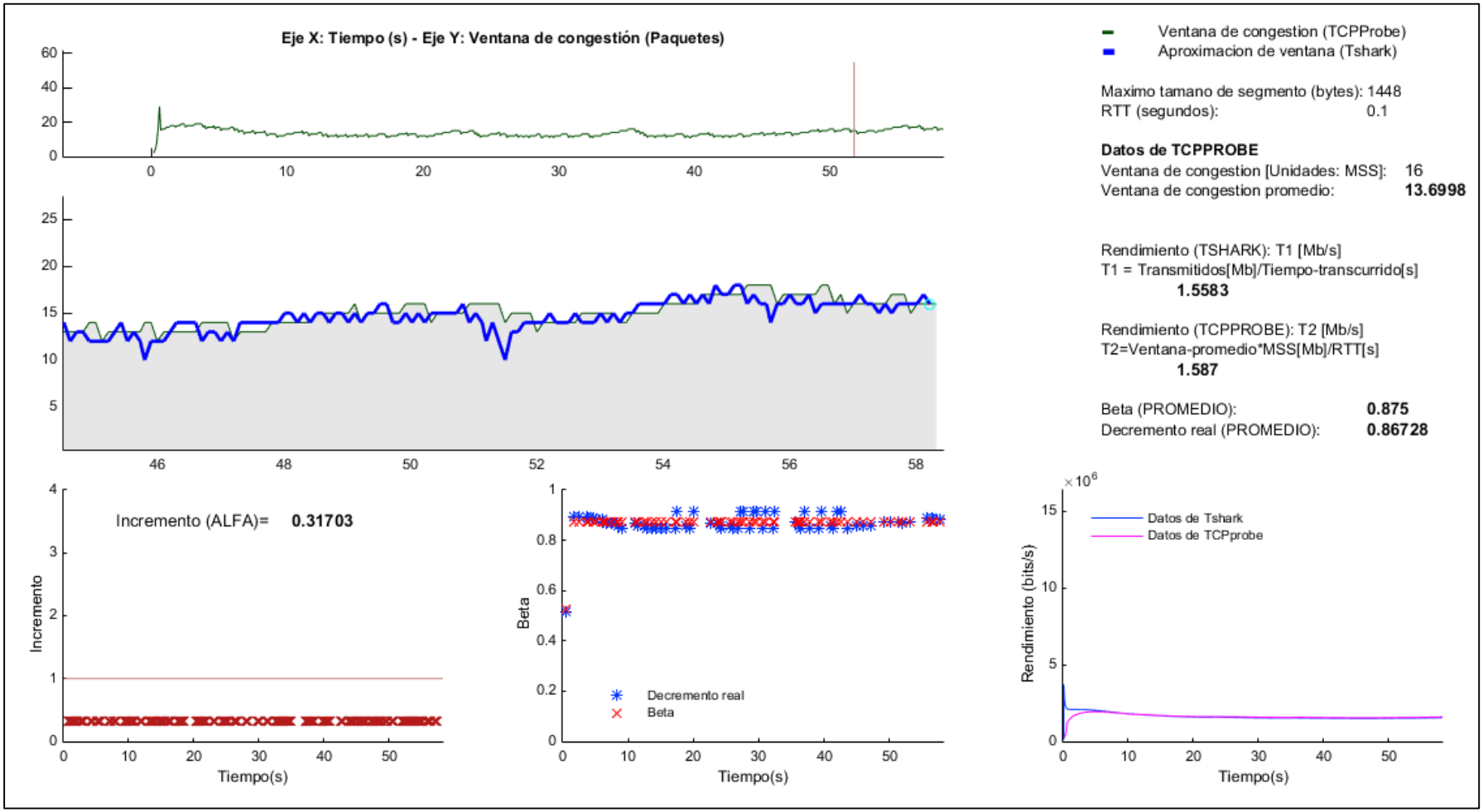


Figura L3. Transmisión empleando GAIMD con RTT=100ms y Proporción de pérdidas = 1% (Pérdidas aleatorias)

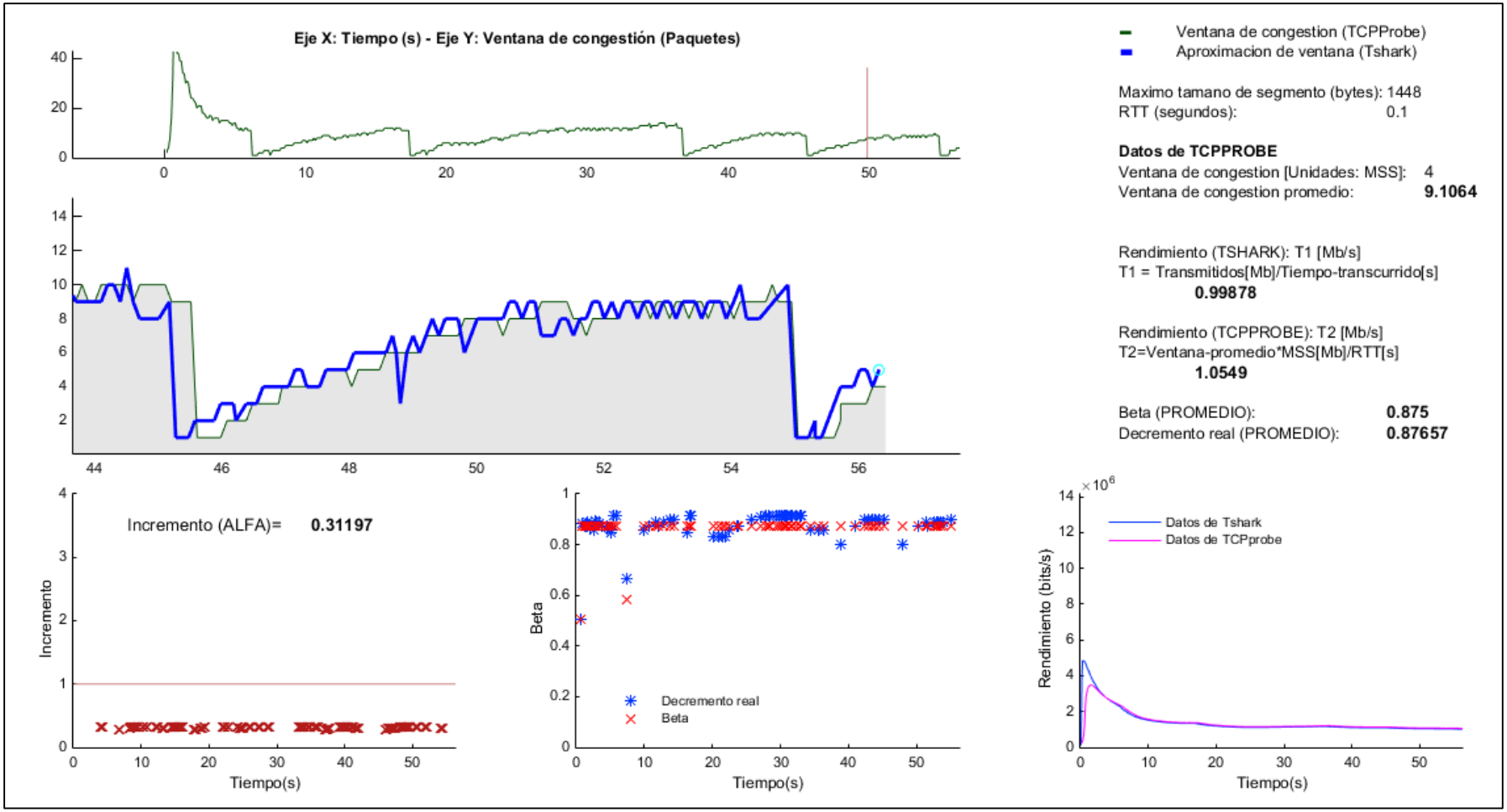
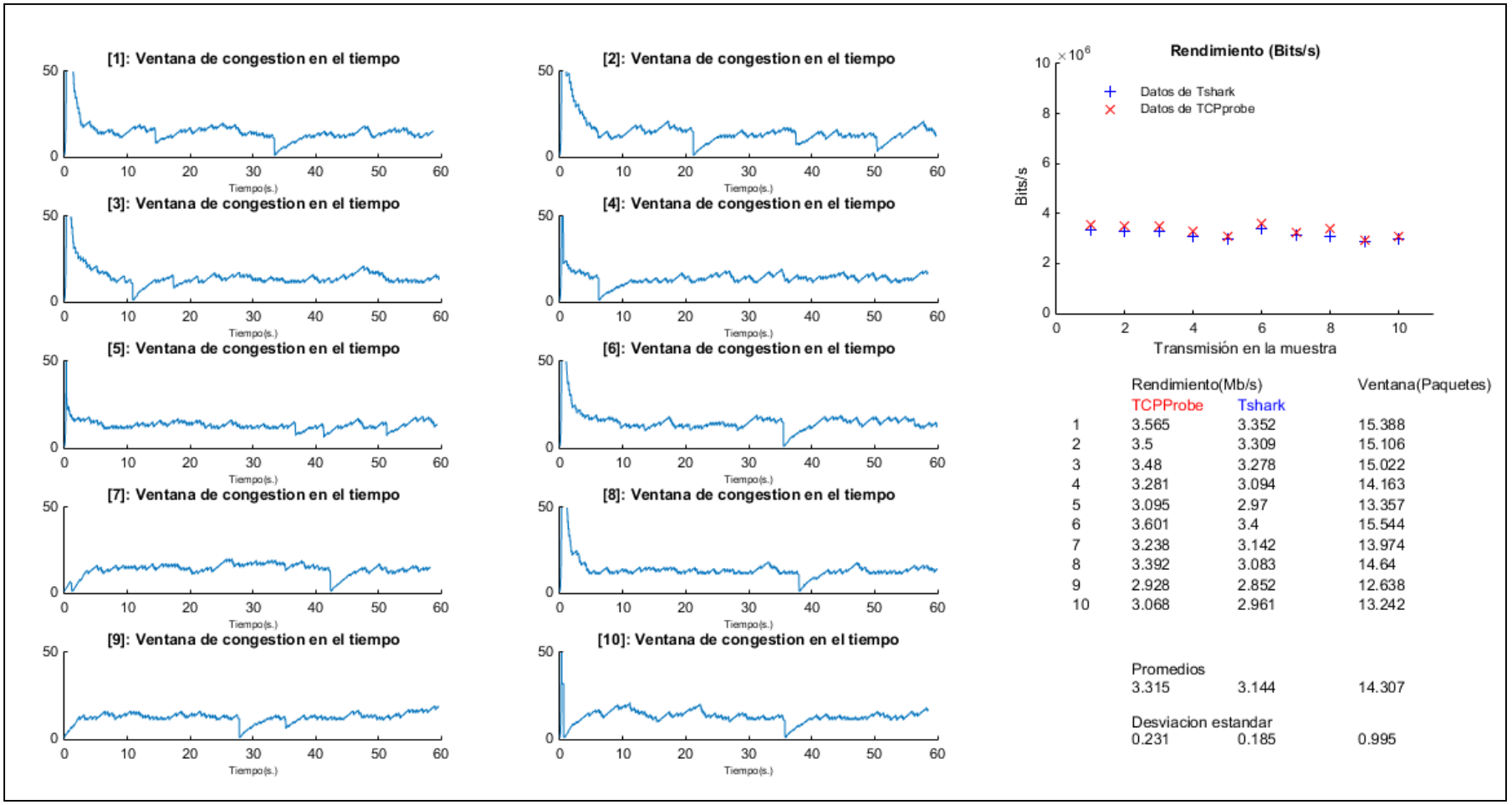


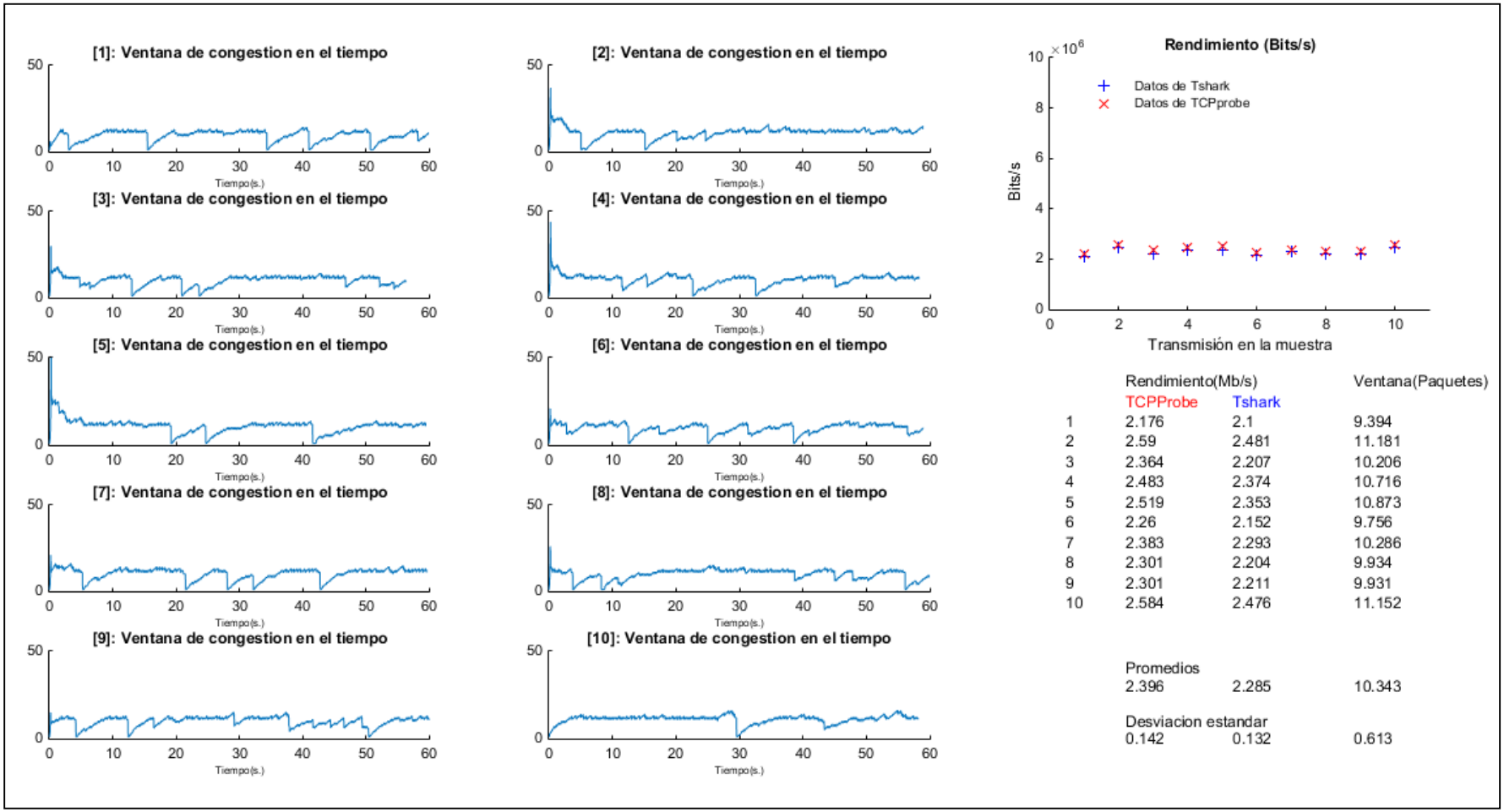
Figura L4. Transmisión empleando GAIMD con RTT=100ms y Proporción de pérdidas = 2% (Pérdidas aleatorias)

2. MUESTRAS DE 10 TRANSMISIONES EMPLEANDO LOS MISMOS PARÁMETROS



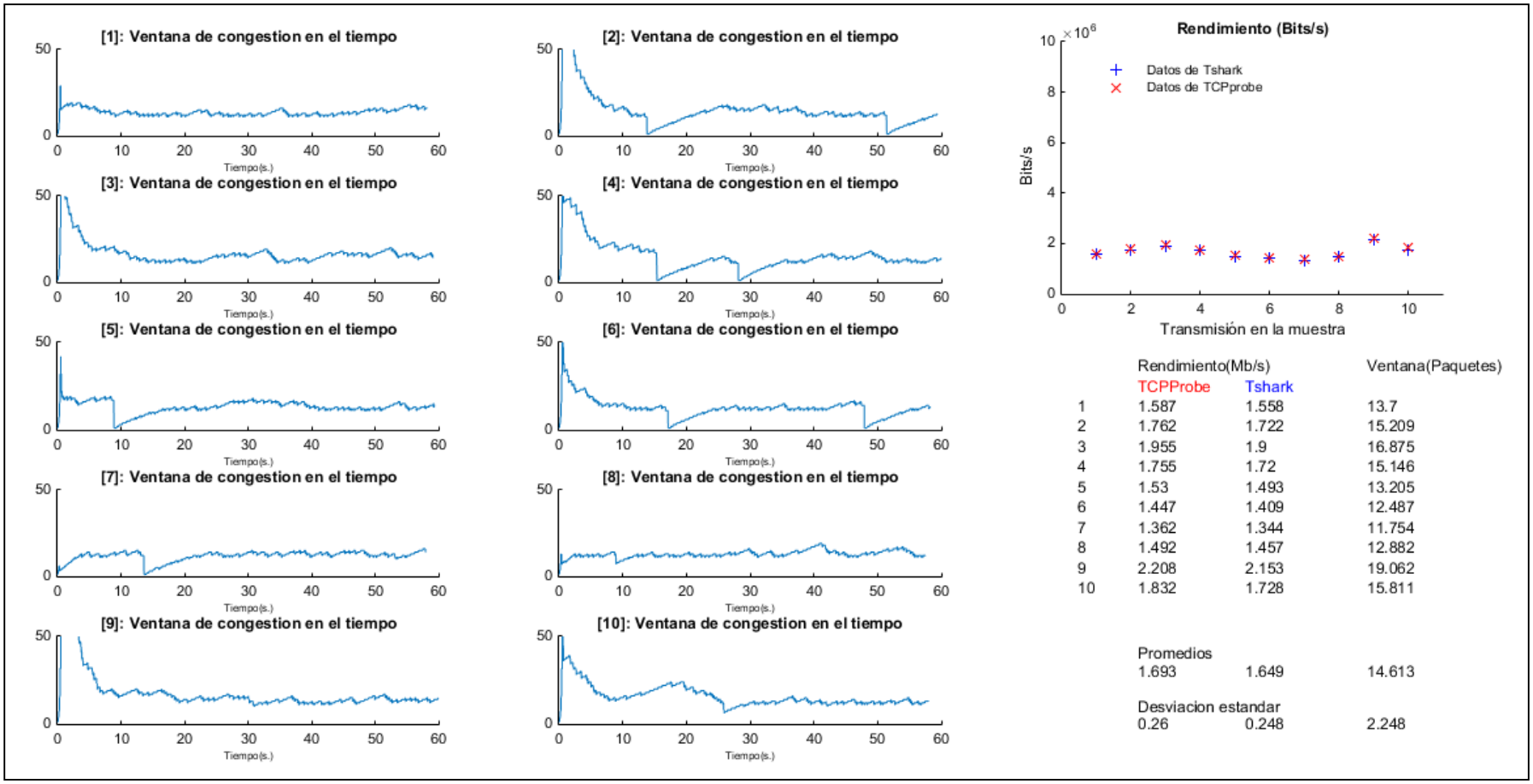
	Rendimiento(Mb/s) TCPProbe	Tshark	Ventana(Paquetes)
1	3.565	3.352	15.388
2	3.5	3.309	15.106
3	3.48	3.278	15.022
4	3.281	3.094	14.163
5	3.095	2.97	13.357
6	3.601	3.4	15.544
7	3.238	3.142	13.974
8	3.392	3.083	14.64
9	2.928	2.852	12.638
10	3.068	2.961	13.242
Promedios	3.315	3.144	14.307
Desviacion estandar	0.231	0.185	0.995

Figura L5. Ventana de congestión y rendimiento empleando GAIMD con RTT=50ms y Proporción de pérdidas =1%



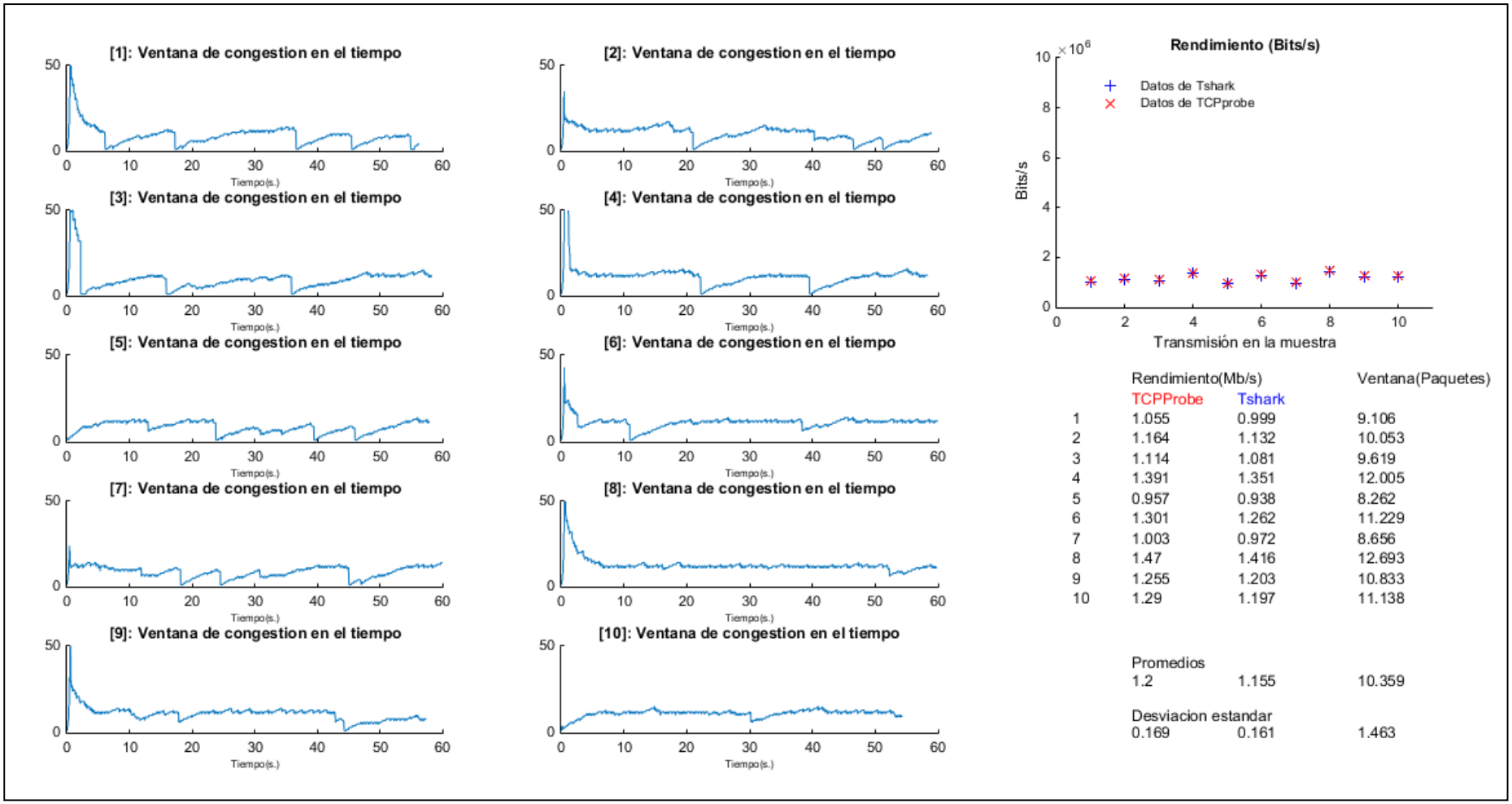
	Rendimiento(Mb/s) TCPProbe	Tshark	Ventana(Paquetes)
1	2.176	2.1	9.394
2	2.59	2.481	11.181
3	2.364	2.207	10.206
4	2.483	2.374	10.716
5	2.519	2.353	10.873
6	2.26	2.152	9.756
7	2.383	2.293	10.286
8	2.301	2.204	9.934
9	2.301	2.211	9.931
10	2.584	2.476	11.152
Promedios	2.396	2.285	10.343
Desviacion estandar	0.142	0.132	0.613

Figura L6. Ventana de congestión y rendimiento empleando GAIMD con **RTT=50ms y Proporción de pérdidas =2%**



	Rendimiento(Mb/s)		Ventana(Paquetes)
	TCPProbe	Tshark	
1	1.587	1.558	13.7
2	1.762	1.722	15.209
3	1.955	1.9	16.875
4	1.755	1.72	15.146
5	1.53	1.493	13.205
6	1.447	1.409	12.487
7	1.362	1.344	11.754
8	1.492	1.457	12.882
9	2.208	2.153	19.062
10	1.832	1.728	15.811
Promedios			
	1.693	1.649	14.613
Desviacion estandar			
	0.26	0.248	2.248

Figura L7. Ventana de congestión y rendimiento empleando GAIMD con RTT=100ms y Proporción de pérdidas =1%



	Rendimiento(Mb/s)		Ventana(Paquetes)
	TCPProbe	Tshark	
1	1.055	0.999	9.106
2	1.164	1.132	10.053
3	1.114	1.081	9.619
4	1.391	1.351	12.005
5	0.957	0.938	8.262
6	1.301	1.262	11.229
7	1.003	0.972	8.656
8	1.47	1.416	12.693
9	1.255	1.203	10.833
10	1.29	1.197	11.138
Promedios			
	1.2	1.155	10.359
Desviacion estandar			
	0.169	0.161	1.463

Figura L8. Ventana de congestión y rendimiento empleando GAIMD con RTT=100ms y Proporción de pérdidas =2%

ANEXO “M”
ANÁLISIS DE ESTP CON MATLAB

1. TRANSMISIÓN PUNTO A PUNTO EMPLEANDO ESTP CON “τ” FIJO

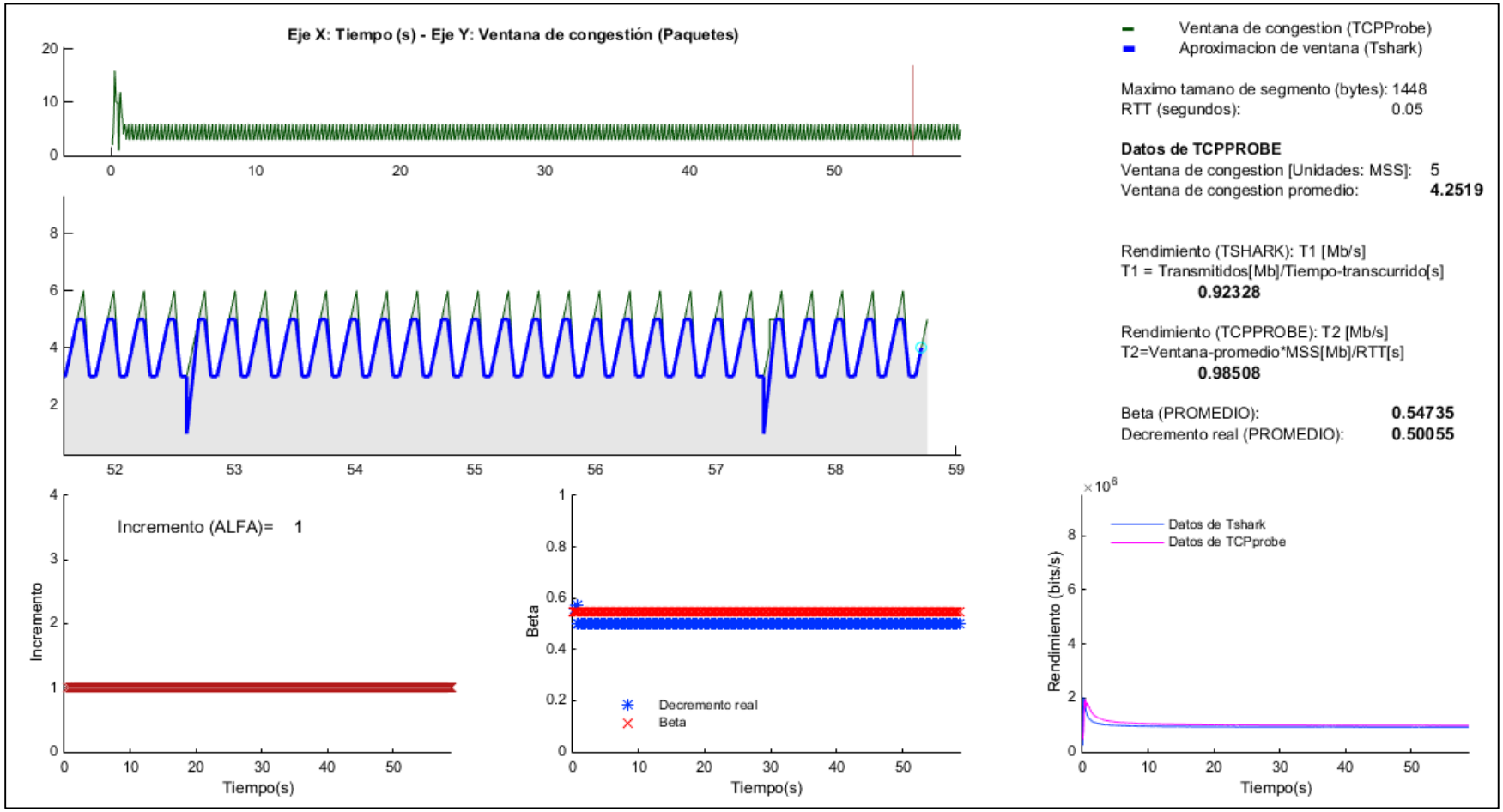
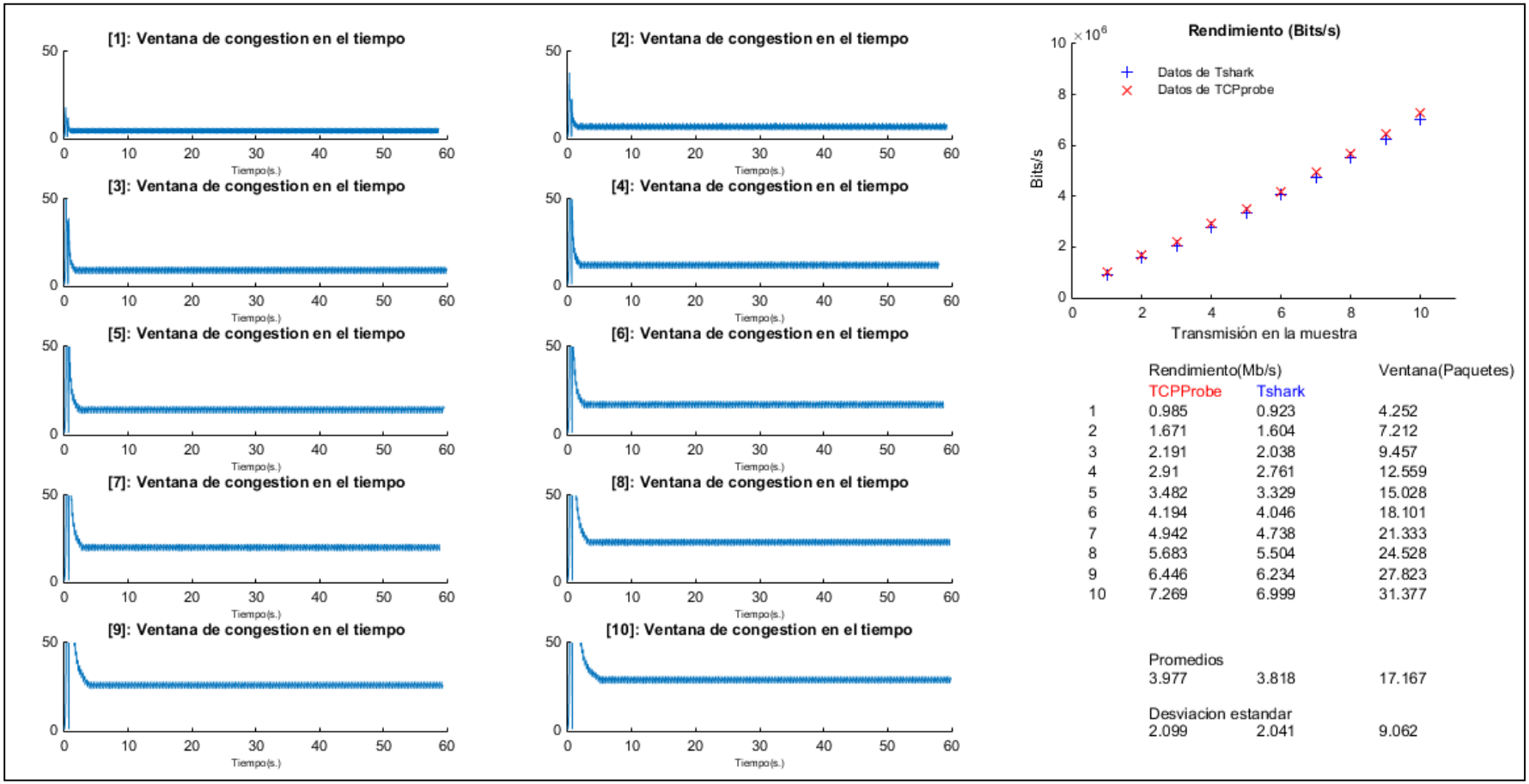


Figura M1. Transmisión empleando ESTP con RTT=50ms, τ=100, δ = 20 y Proporción de pérdidas = 5% (Pérdidas constantes)

2. MUESTRA DE 10 TRANSMISIONES CON UN VALOR DE "τ" = 100 Y DIFERENTES VALORES DE "δ"



	Rendimiento(Mb/s) TCPProbe	Tshark	Ventana(Paquetes)
1	0.985	0.923	4.252
2	1.671	1.604	7.212
3	2.191	2.038	9.457
4	2.91	2.761	12.559
5	3.482	3.329	15.028
6	4.194	4.046	18.101
7	4.942	4.738	21.333
8	5.683	5.504	24.528
9	6.446	6.234	27.823
10	7.269	6.999	31.377
Promedios	3.977	3.818	17.167
Desviacion estandar	2.099	2.041	9.062

Figura M2. Transmisiones con ESTP con RTT=50ms, τ=100, δ=20, δ=40, δ=60, δ=80, δ=100, δ=120, δ=140, δ=160, δ=180 y δ=200 (Ventana de congestión y rendimiento)

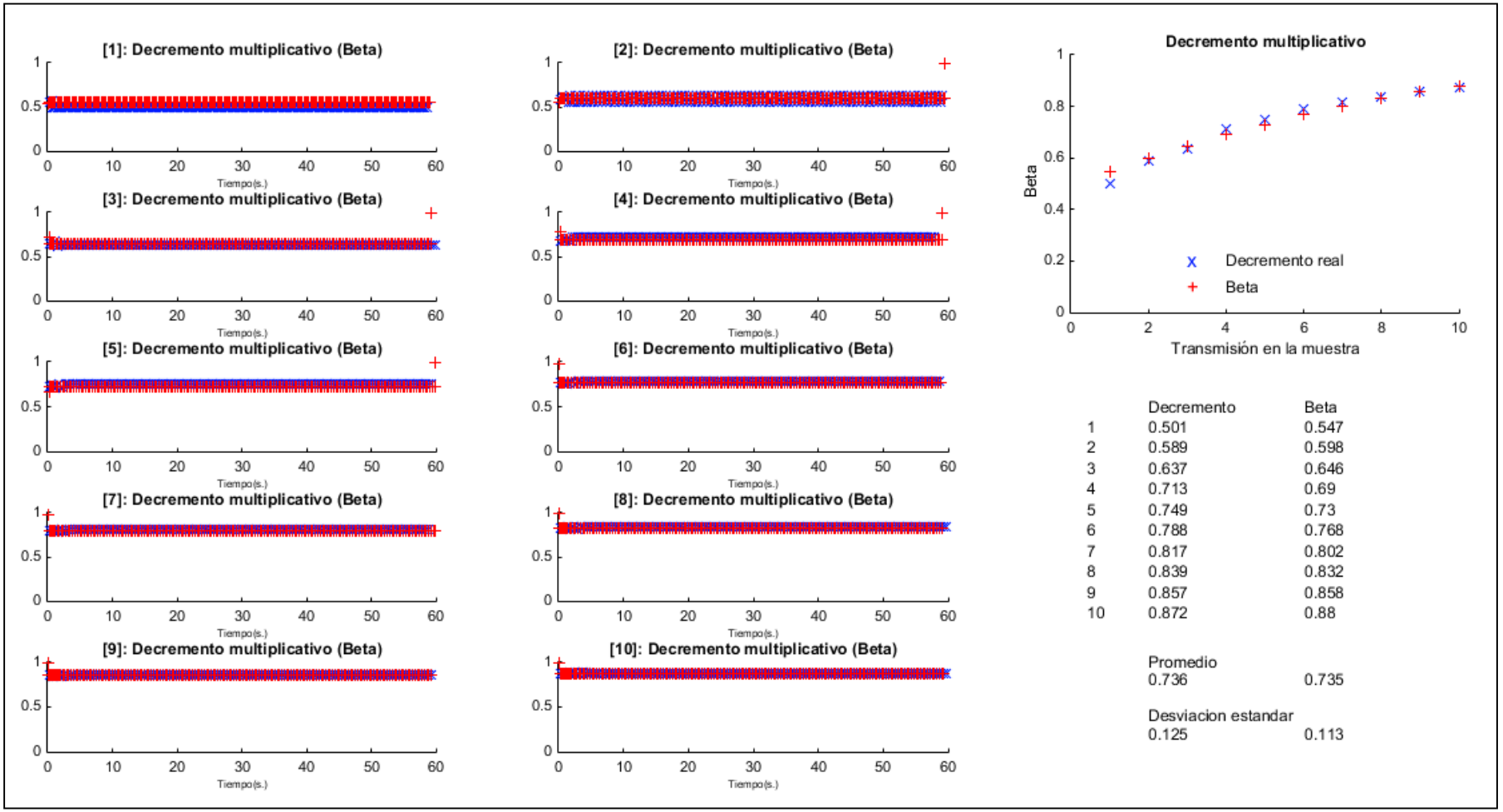
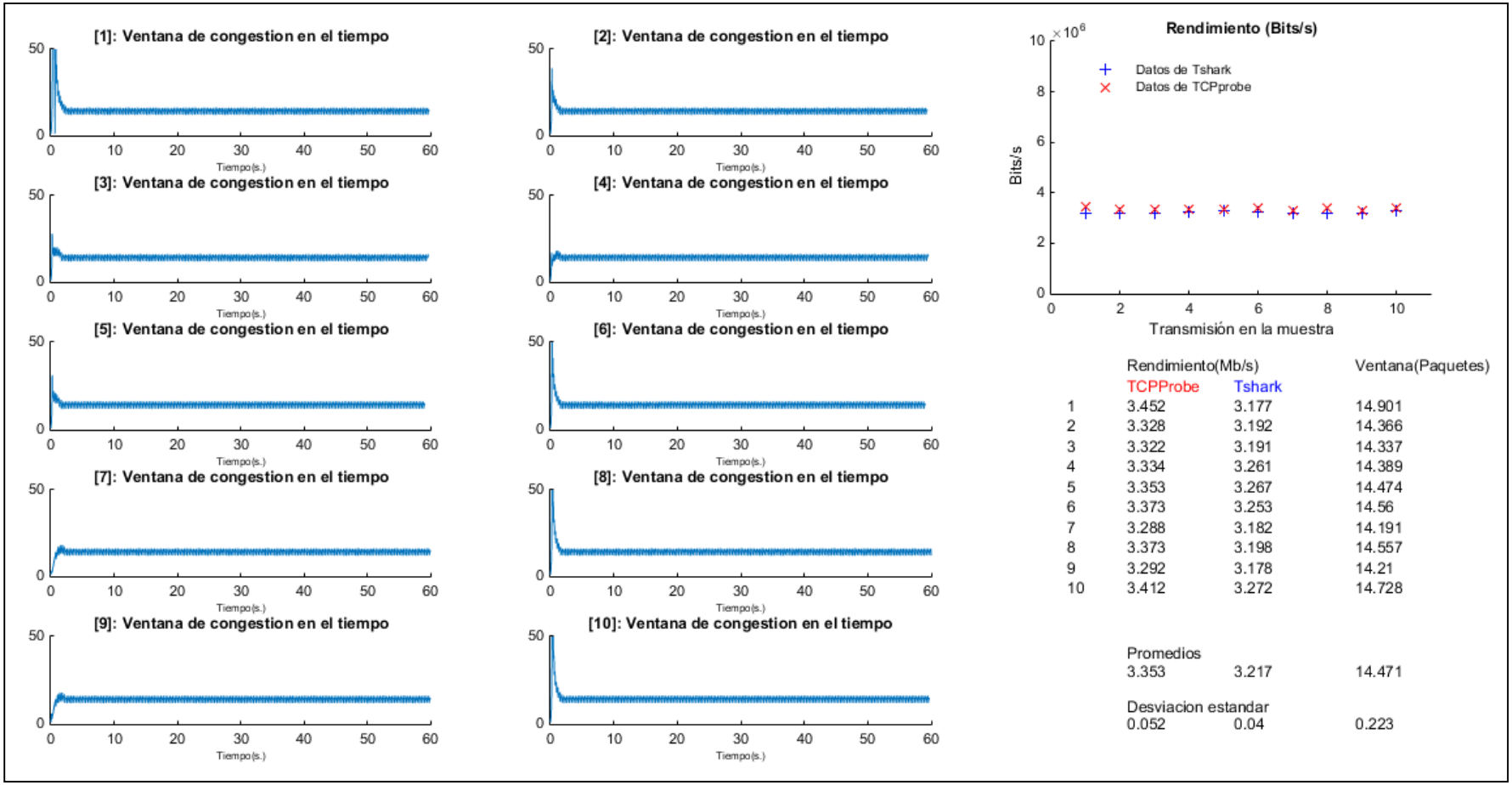


Figura M3. Transmisiones con ESTP con RTT=50ms, δ=20, δ=40, δ=60, δ=80, δ=100, δ=120, δ=140, δ=160, δ=180 y δ=200 (β de ejecución y decremento real)

3. MUESTRAS DE 10 TRANSMISIONES CON PERDIDAS CONSTANTES (SE EMPLEA τ FIJO).



	Rendimiento(Mb/s)		Ventana(Paquetes)
	TCPProbe	Tshark	
1	3.452	3.177	14.901
2	3.328	3.192	14.366
3	3.322	3.191	14.337
4	3.334	3.261	14.389
5	3.353	3.267	14.474
6	3.373	3.253	14.56
7	3.288	3.182	14.191
8	3.373	3.198	14.557
9	3.292	3.178	14.21
10	3.412	3.272	14.728
Promedios			
	3.353	3.217	14.471
Desviacion estandar			
	0.052	0.04	0.223

Figura M4. Transmisiones con ESTP con RTT=50ms y pérdidas constantes del 1% (δ = 100, τ = 100)

(Ventana de congestión y rendimiento)

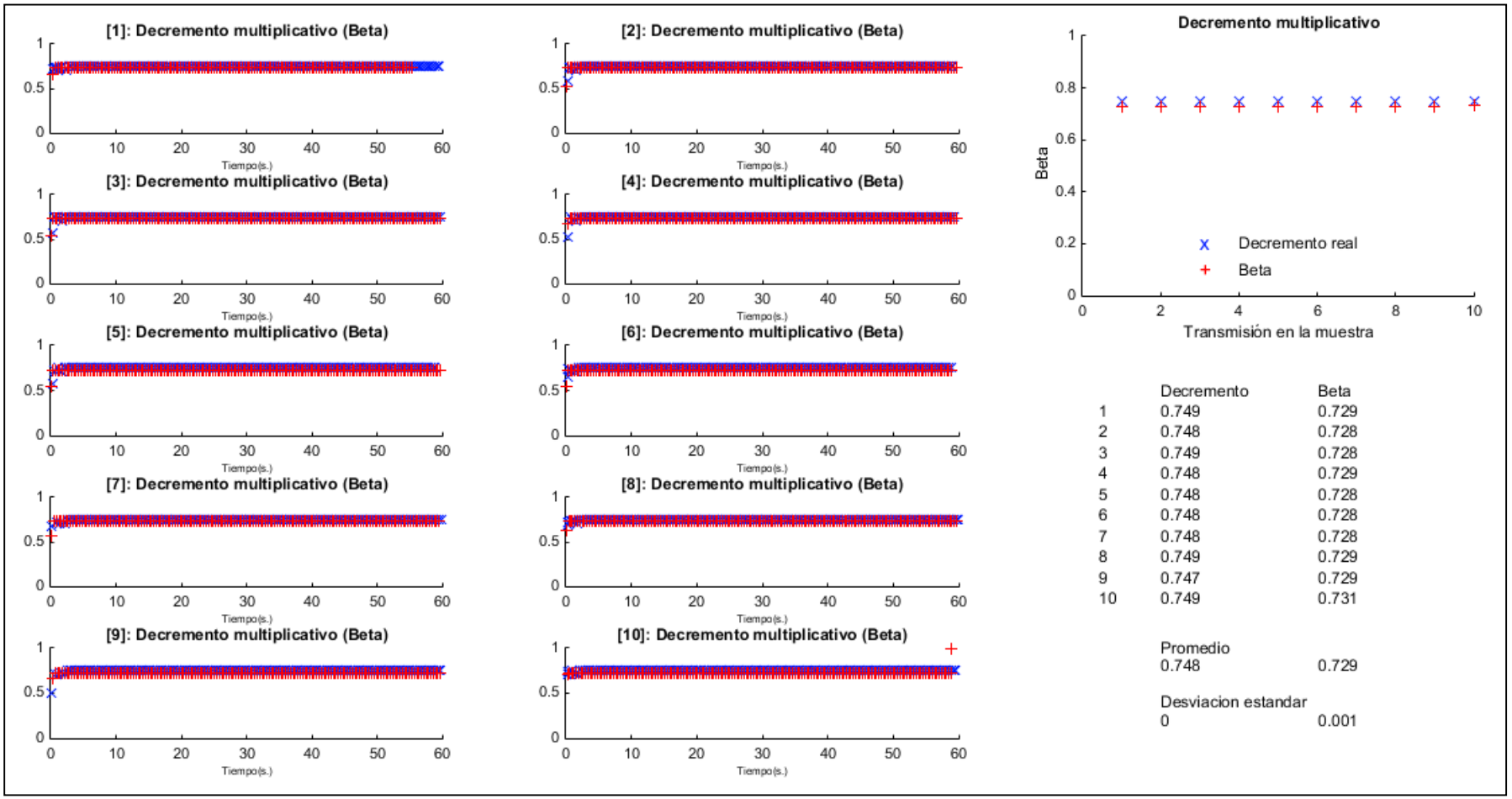
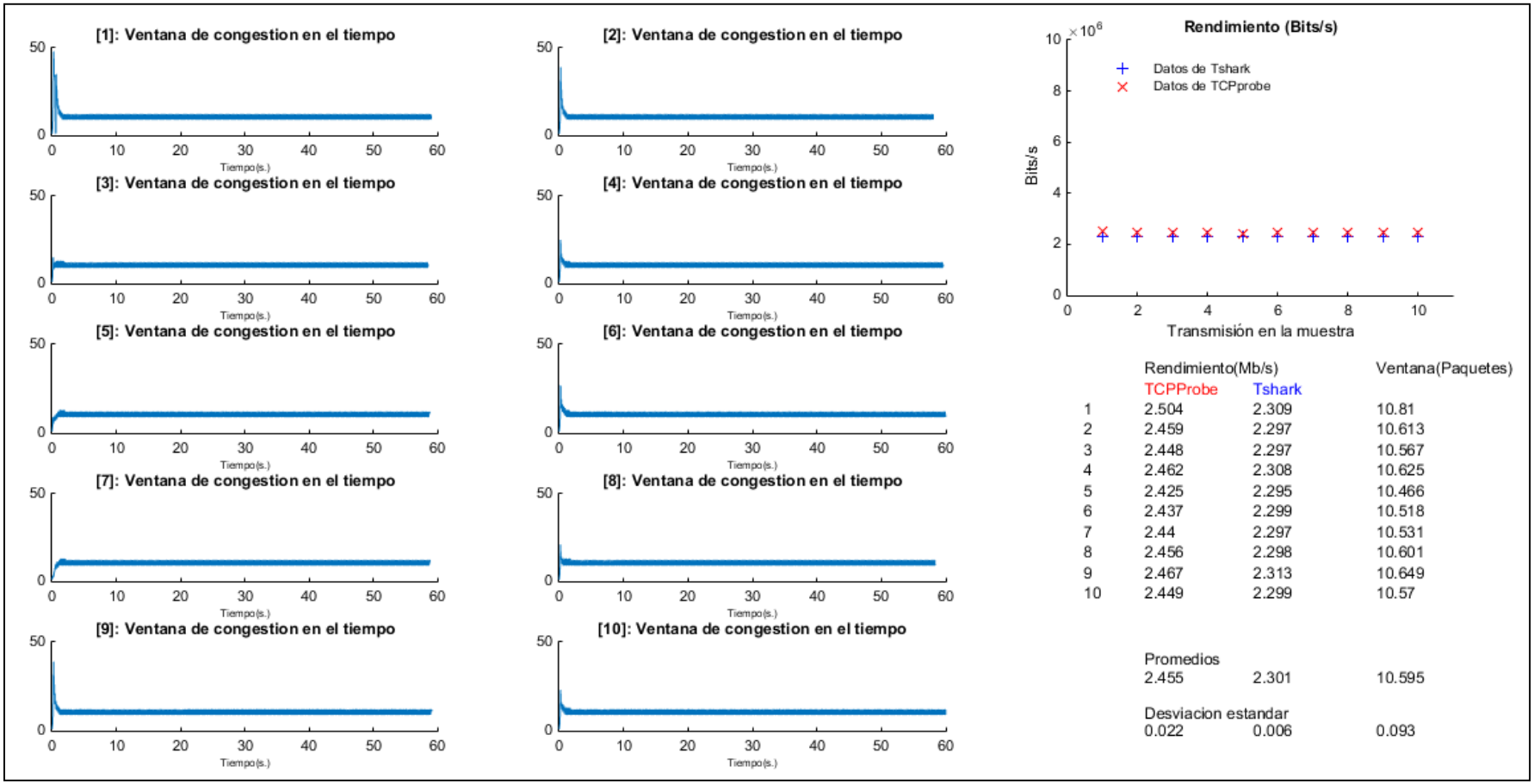


Figura M5. Transmisiones con ESTP con RTT=50ms y pérdidas constantes del 1% (δ = 100, τ = 100)

(β de ejecución y decremento real)



	Rendimiento(Mb/s)		Ventana(Paquetes)
	TCPProbe	Tshark	
1	2.504	2.309	10.81
2	2.459	2.297	10.613
3	2.448	2.297	10.567
4	2.462	2.308	10.625
5	2.425	2.295	10.466
6	2.437	2.299	10.518
7	2.44	2.297	10.531
8	2.456	2.298	10.601
9	2.467	2.313	10.649
10	2.449	2.299	10.57
Promedios			
	2.455	2.301	10.595
Desviacion estandar			
	0.022	0.006	0.093

Figura M6. Transmisiones con ESTP con RTT=50ms y pérdidas constantes del 2% (δ = 50, τ = 50)

(Ventana de congestión y rendimiento)

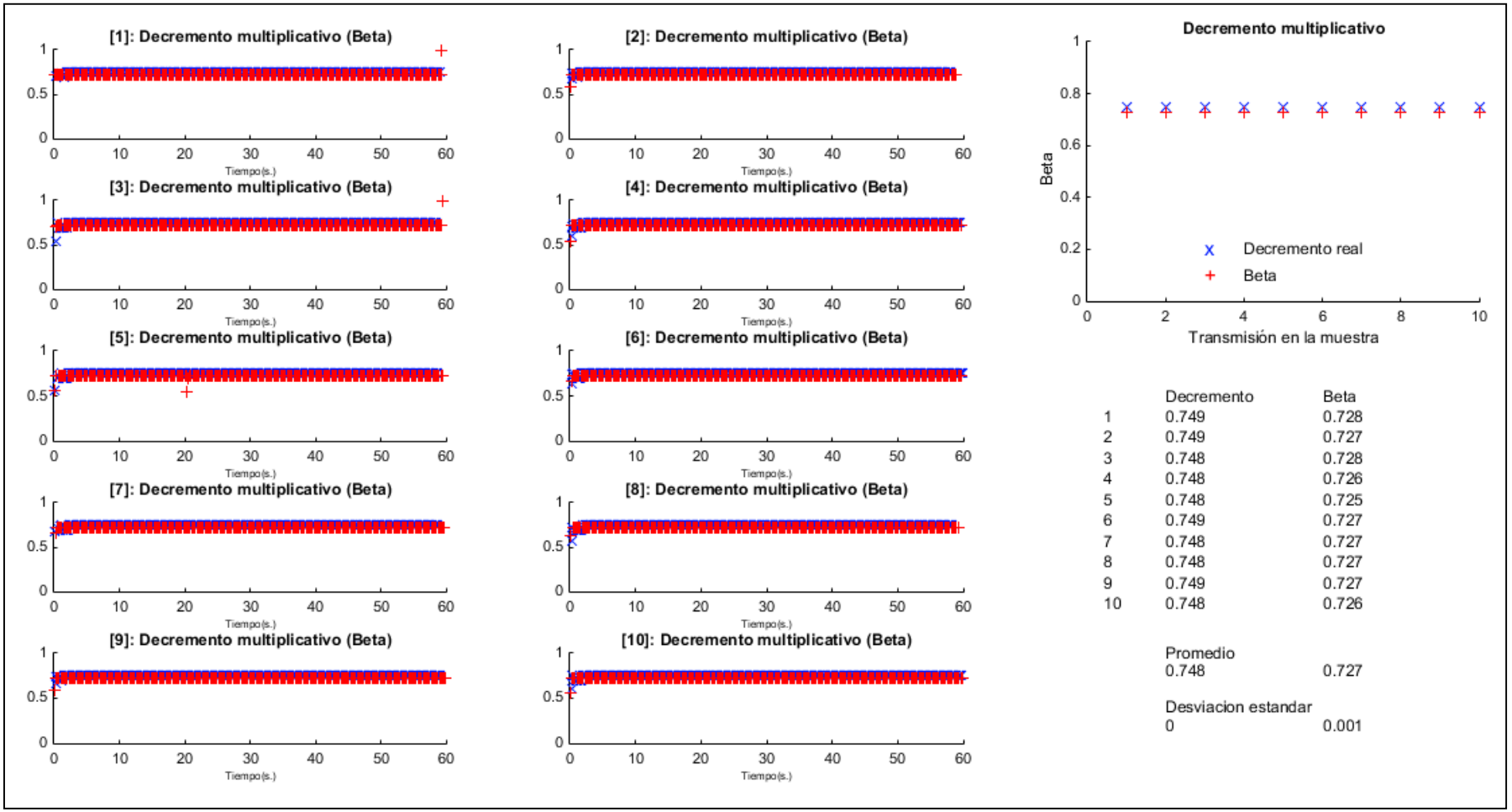
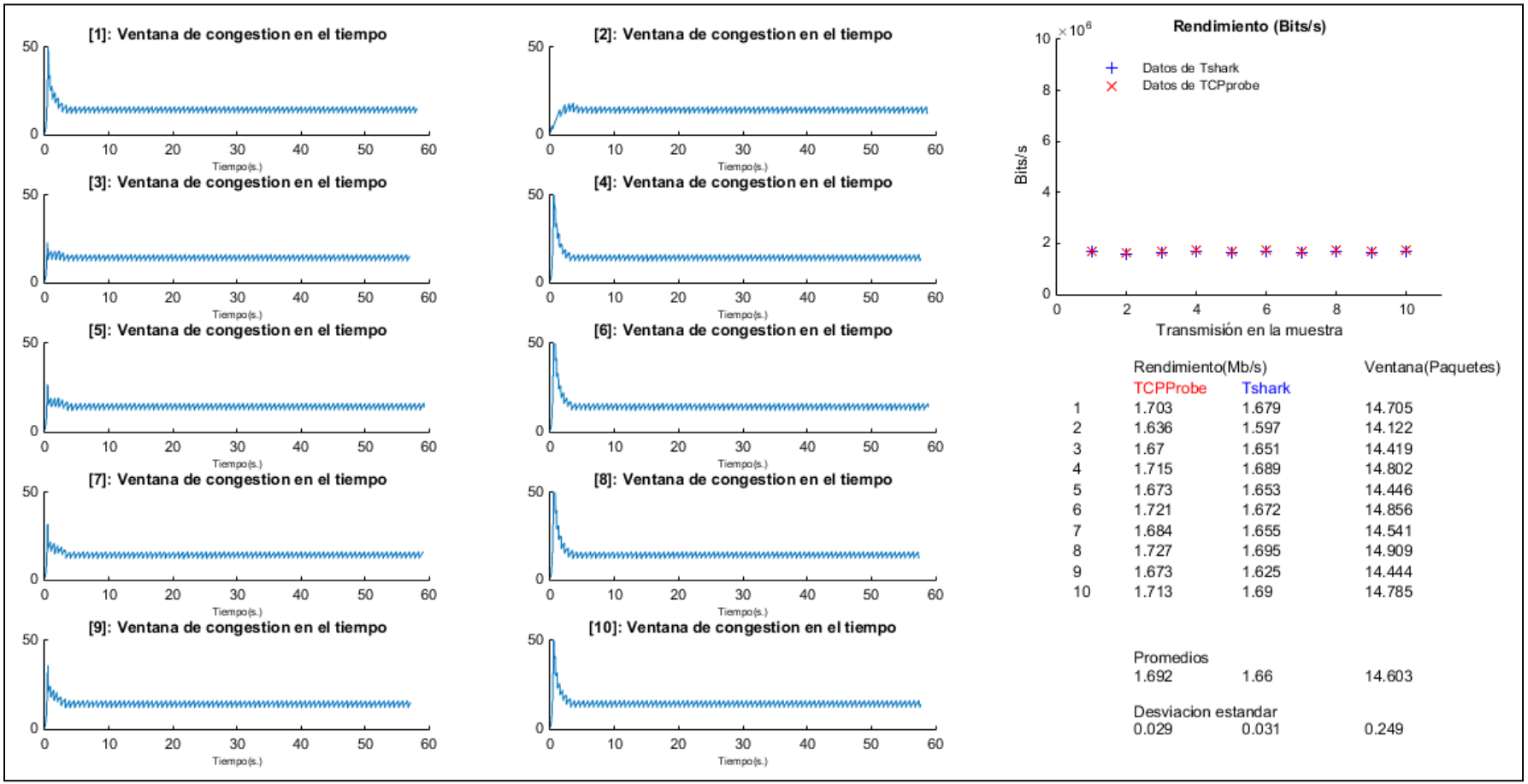


Figura M7. Transmisiones con ESTP con RTT=50ms y pérdidas constantes del 2% (δ = 50, τ = 50)

(β de ejecución y decremento real)



	Rendimiento(Mb/s)		Ventana(Paquetes)
	TCPProbe	Tshark	
1	1.703	1.679	14.705
2	1.636	1.597	14.122
3	1.67	1.651	14.419
4	1.715	1.689	14.802
5	1.673	1.653	14.446
6	1.721	1.672	14.856
7	1.684	1.655	14.541
8	1.727	1.695	14.909
9	1.673	1.625	14.444
10	1.713	1.69	14.785
Promedios			
	1.692	1.66	14.603
Desviacion estandar			
	0.029	0.031	0.249

Figura M8. Transmisiones con ESTP con RTT=100ms y pérdidas constantes del 1% (δ = 100, τ = 100)

(Ventana de congestión y rendimiento)

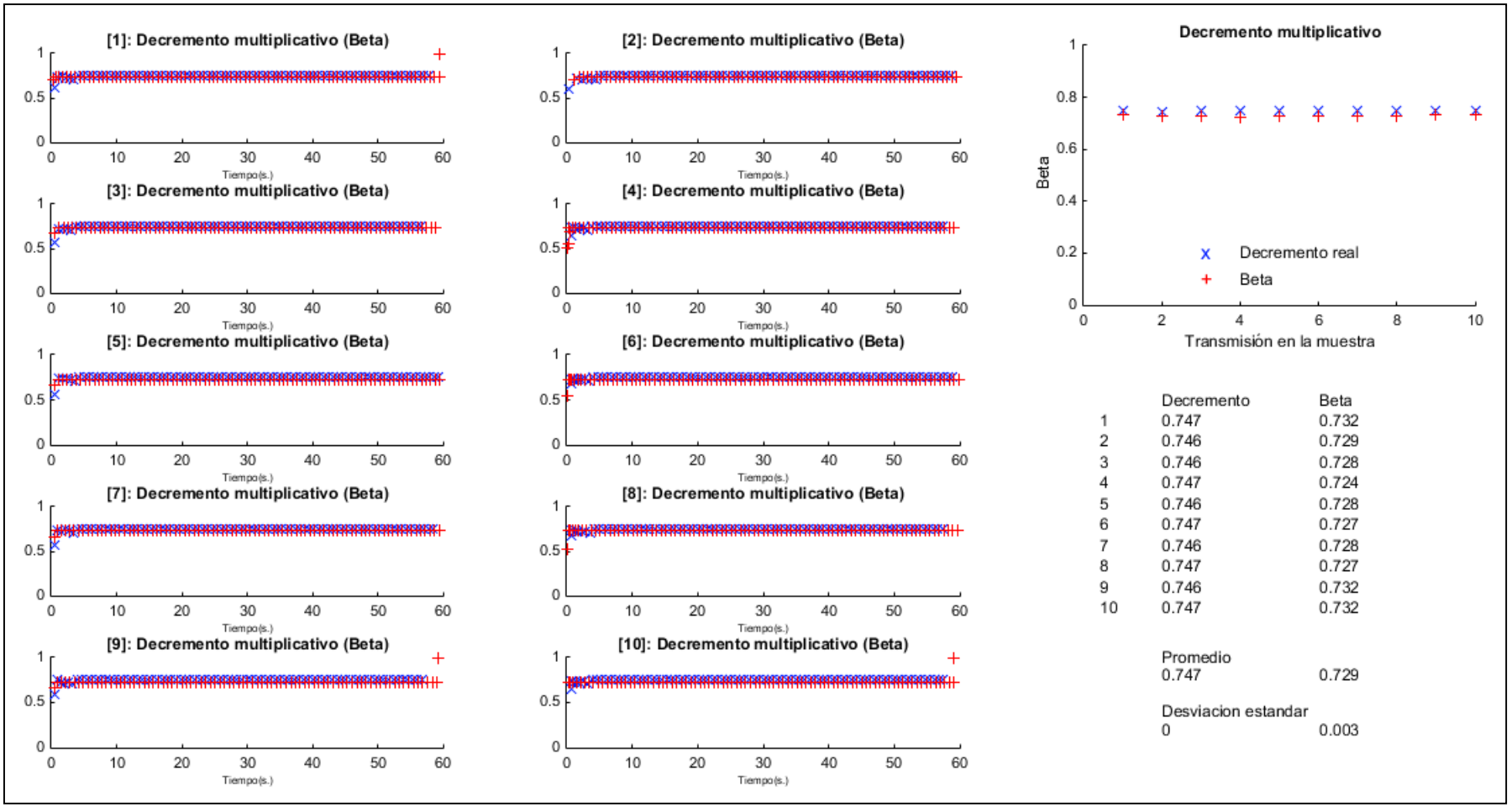
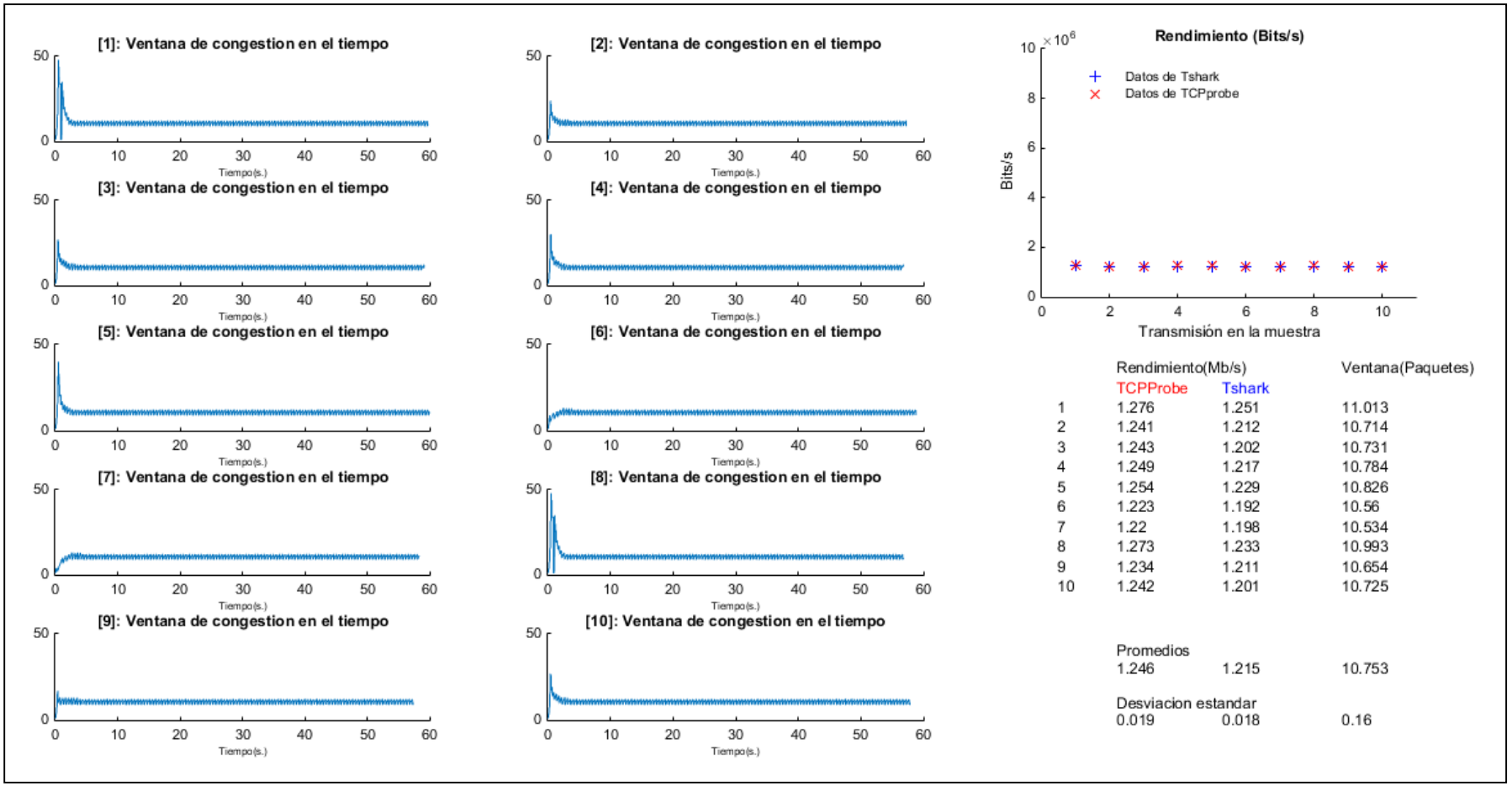


Figura M9. Transmisiones con ESTP con RTT=100ms y pérdidas constantes del 1% (δ = 100, τ = 100)

(β de ejecución y decremento real)



	Rendimiento(Mb/s)		Ventana(Paquetes)
	TCPProbe	Tshark	
1	1.276	1.251	11.013
2	1.241	1.212	10.714
3	1.243	1.202	10.731
4	1.249	1.217	10.784
5	1.254	1.229	10.826
6	1.223	1.192	10.56
7	1.22	1.198	10.534
8	1.273	1.233	10.993
9	1.234	1.211	10.654
10	1.242	1.201	10.725
Promedios			
	1.246	1.215	10.753
Desviacion estandar			
	0.019	0.018	0.16

Figura M10. Transmisiones con ESTP con RTT=100ms y pérdidas constantes del 2% (δ = 50, τ = 50)

(Ventana de congestión y rendimiento)

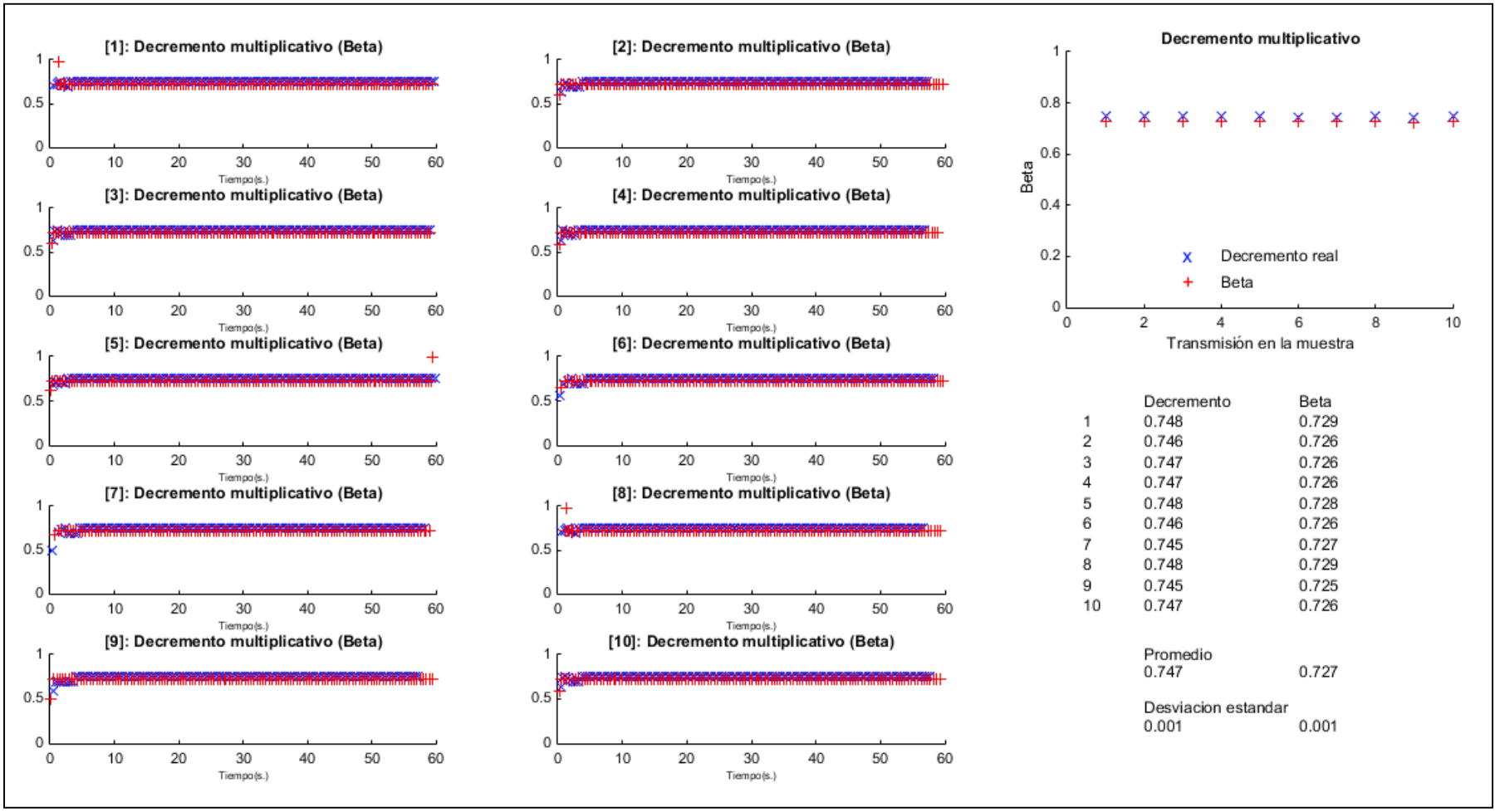


Figura M11. Transmisiones con ESTP con RTT=100ms y pérdidas constantes del 2% (δ = 50, τ = 50)

(β de ejecución y decremento real)

4. TRANSMISIONES CON PERDIDAS CONSTANTES (SE EMPLEA τ DINAMICO EN EL PROGRAMA DE CÓDIGO FUENTE).

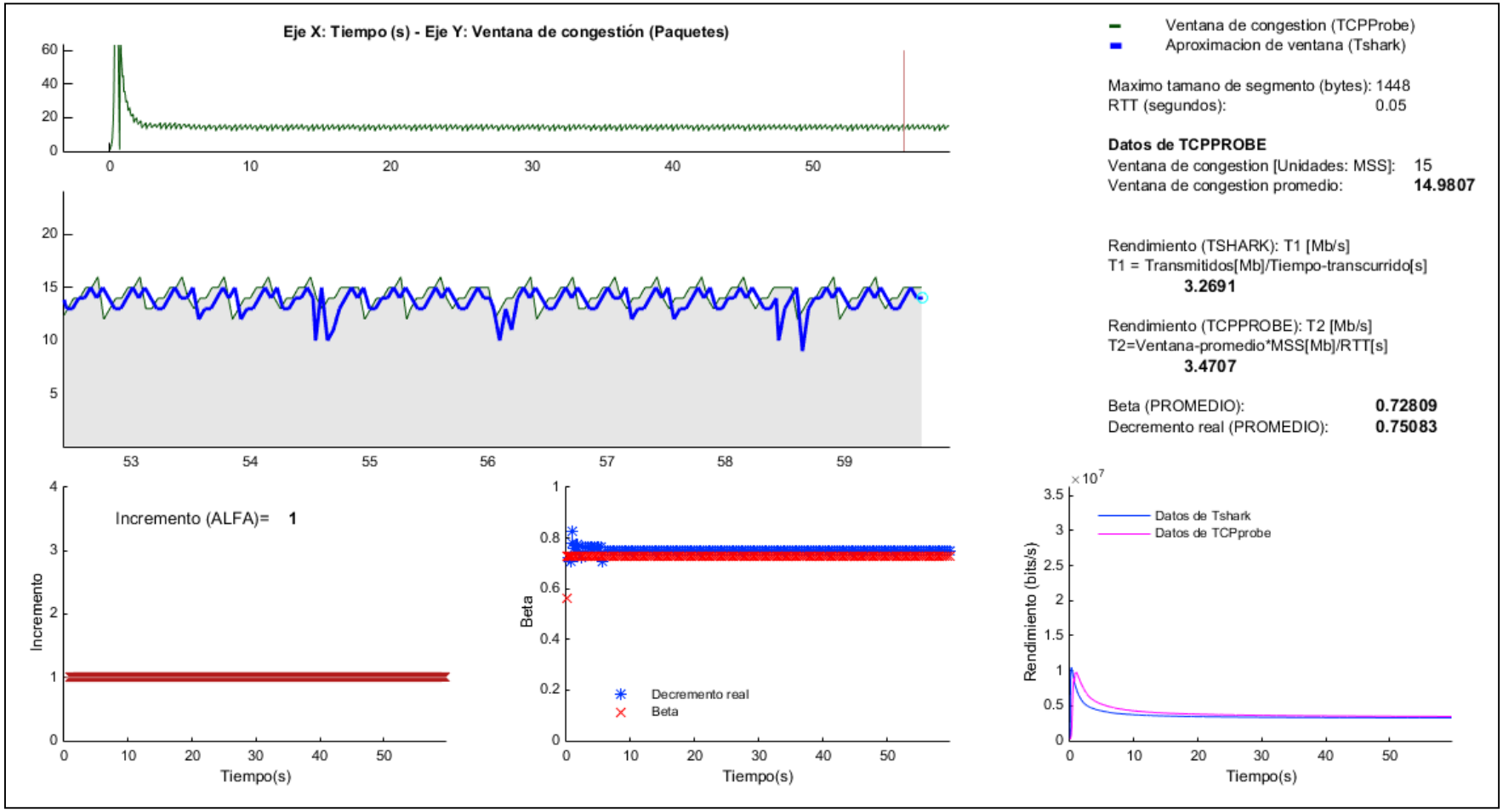


Figura M12. Transmisión que emplea ESTP con RTT=50ms y pérdidas constantes del 1%.

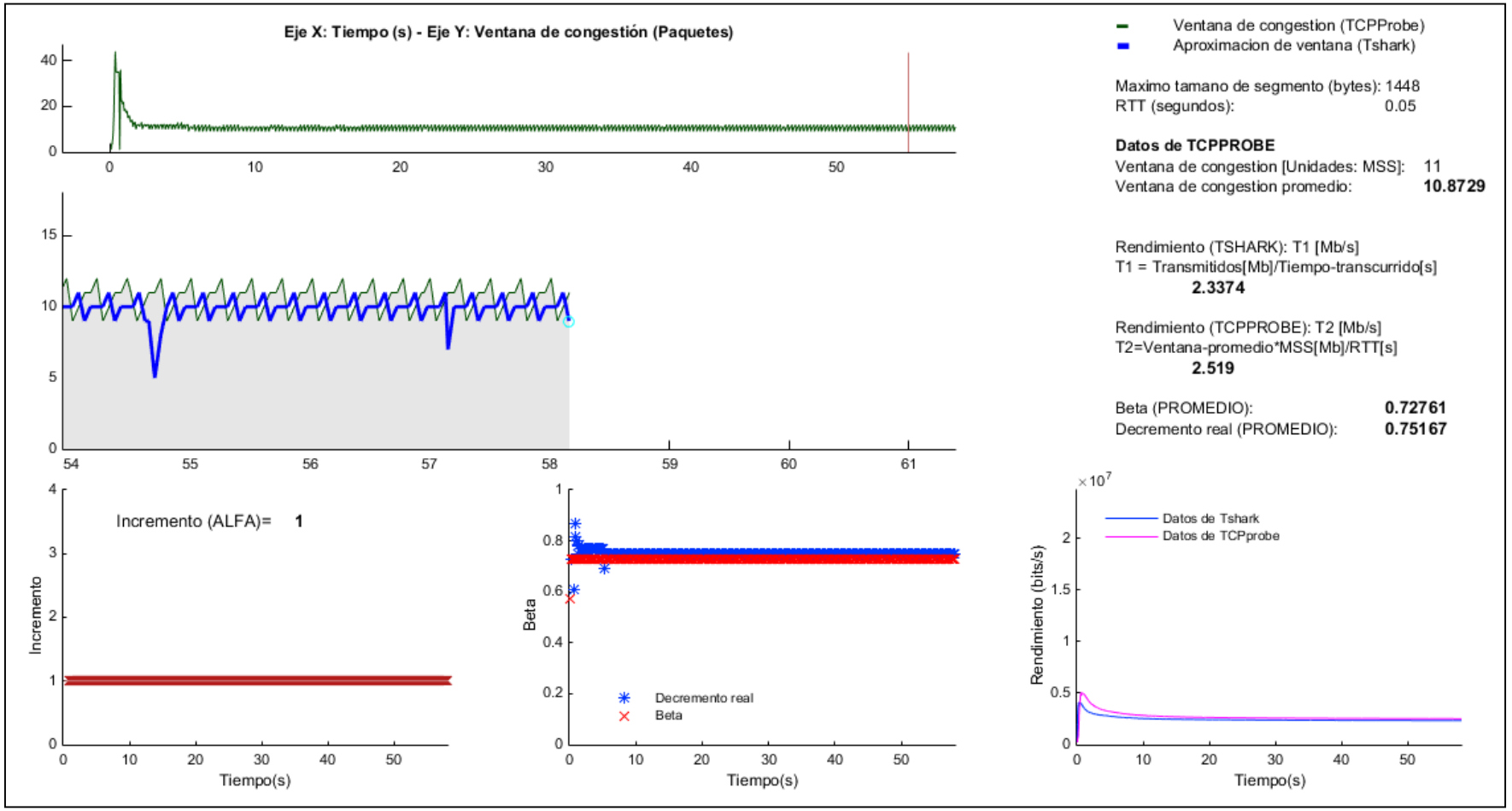


Figura M13. Transmisión que emplea ESTP con RTT=50ms y pérdidas constantes del 2%.

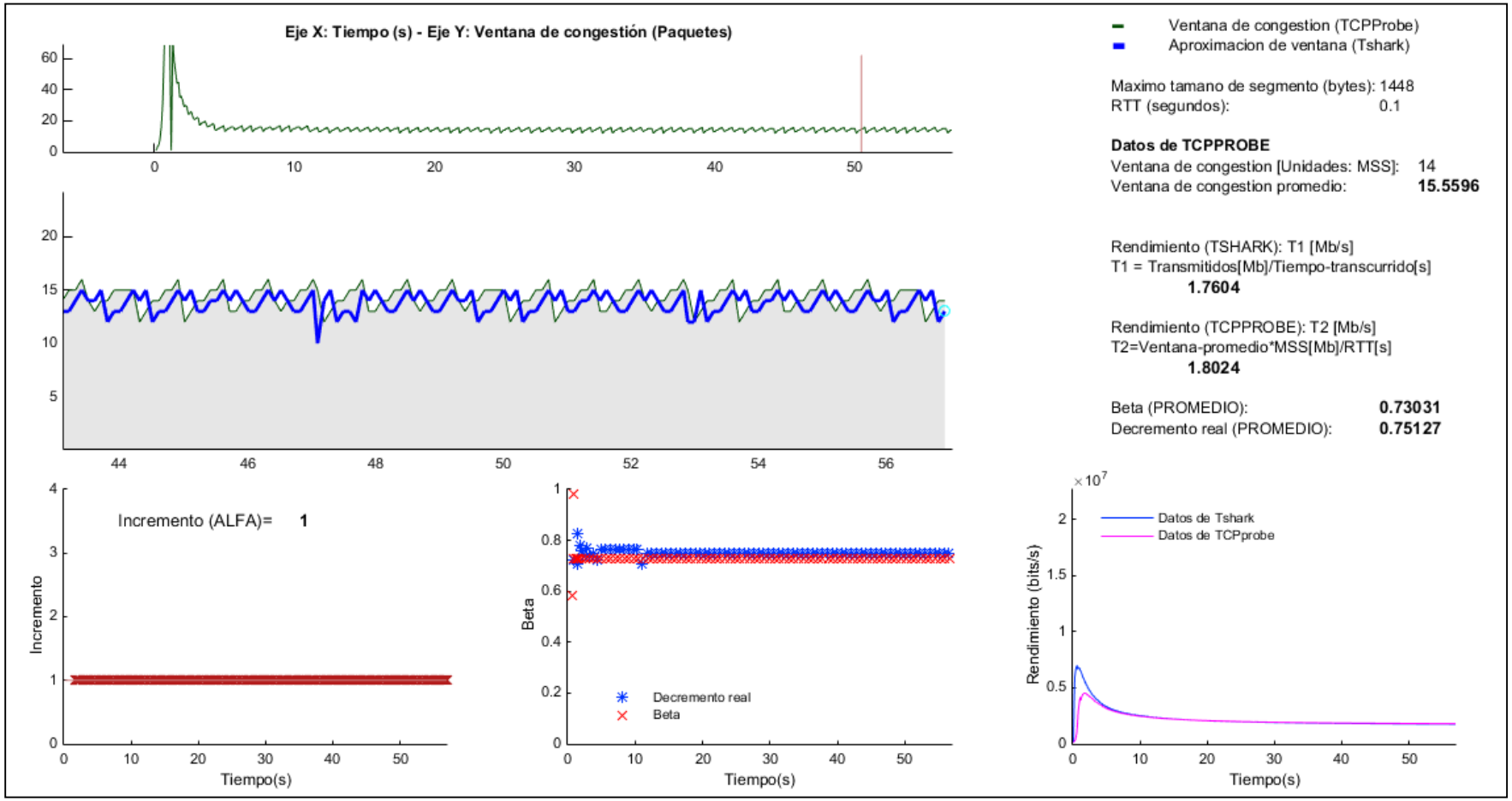


Figura M14. Transmisión que emplea ESTP con RTT=100ms y pérdidas constantes del 1%.

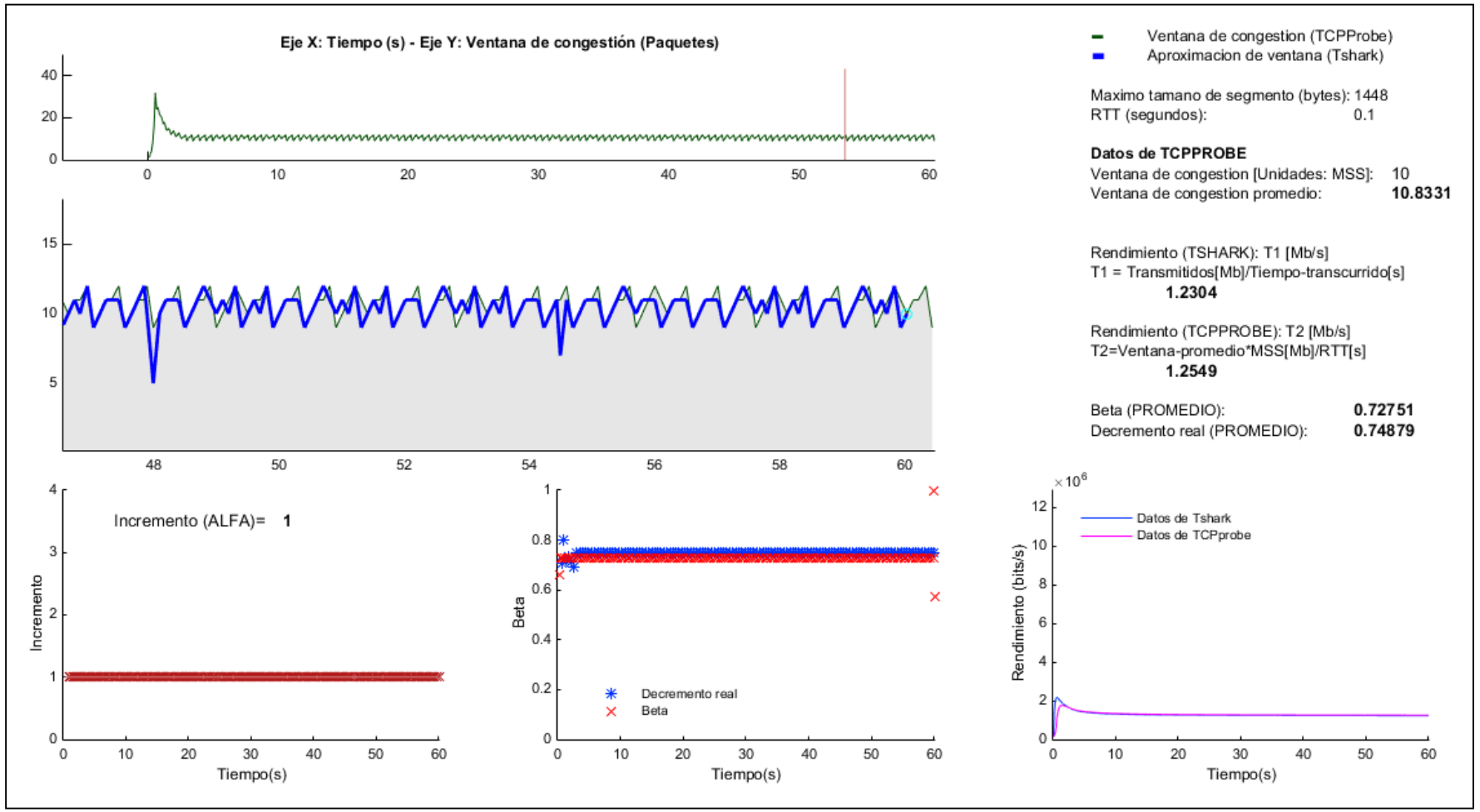
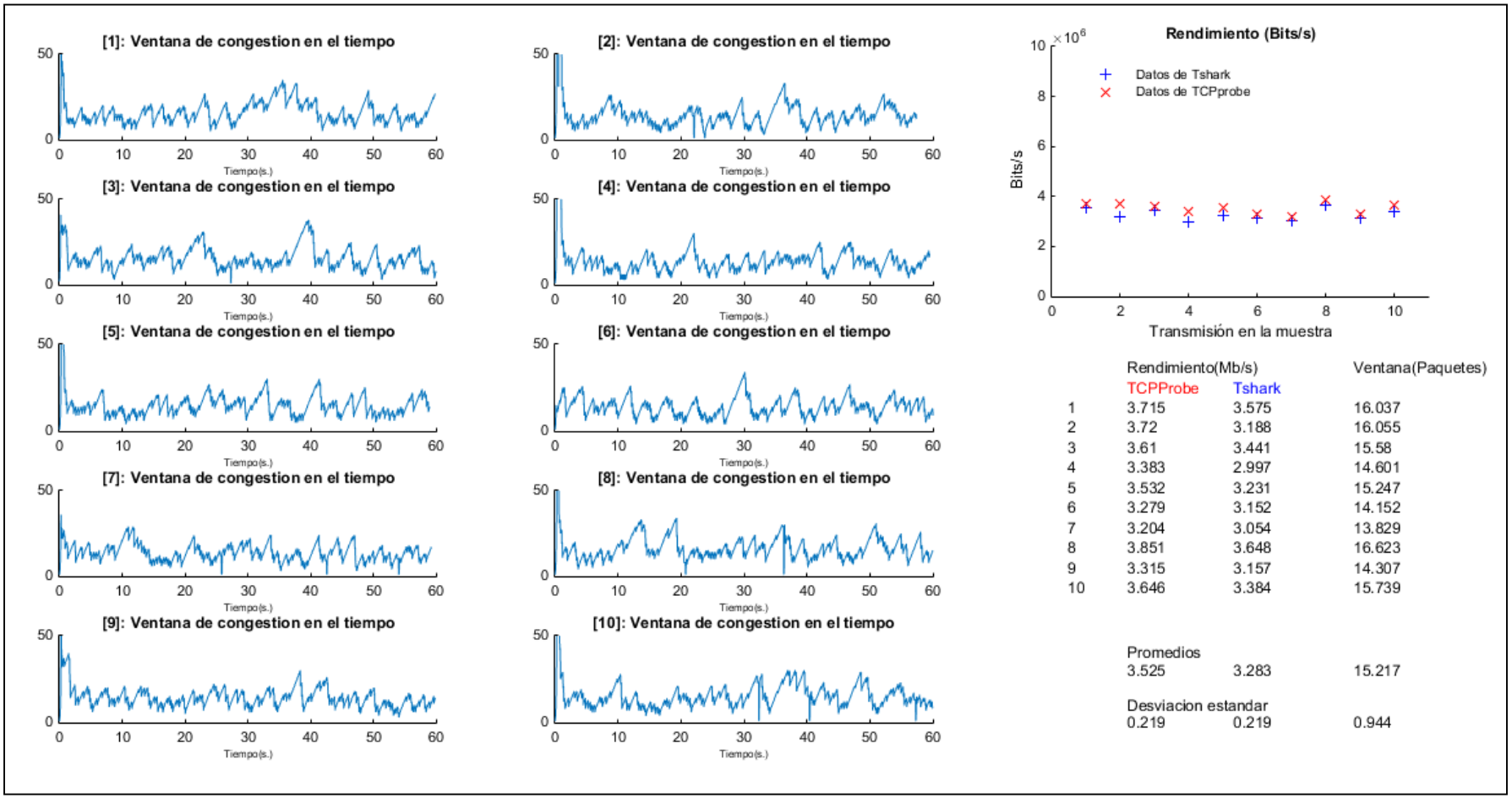


Figura M15. Transmisión que emplea ESTP con RTT=100ms y pérdidas constantes del 2%.

5. MUESTRAS DE 10 TRANSMISIONES CON PERDIDAS ALEATORIAS (SE EMPLEA τ DINAMICO EN EL PROGRAMA DE CÓDIGO FUENTE).



	Rendimiento(Mb/s) TCPProbe	Tshark	Ventana(Paquetes)
1	3.715	3.575	16.037
2	3.72	3.188	16.055
3	3.61	3.441	15.58
4	3.383	2.997	14.601
5	3.532	3.231	15.247
6	3.279	3.152	14.152
7	3.204	3.054	13.829
8	3.851	3.648	16.623
9	3.315	3.157	14.307
10	3.646	3.384	15.739
Promedios	3.525	3.283	15.217
Desviacion estandar	0.219	0.219	0.944

Figura M16. Transmisiones con ESTP con RTT=50ms y pérdidas aleatorias del 1%

(Ventana de congestión y rendimiento)

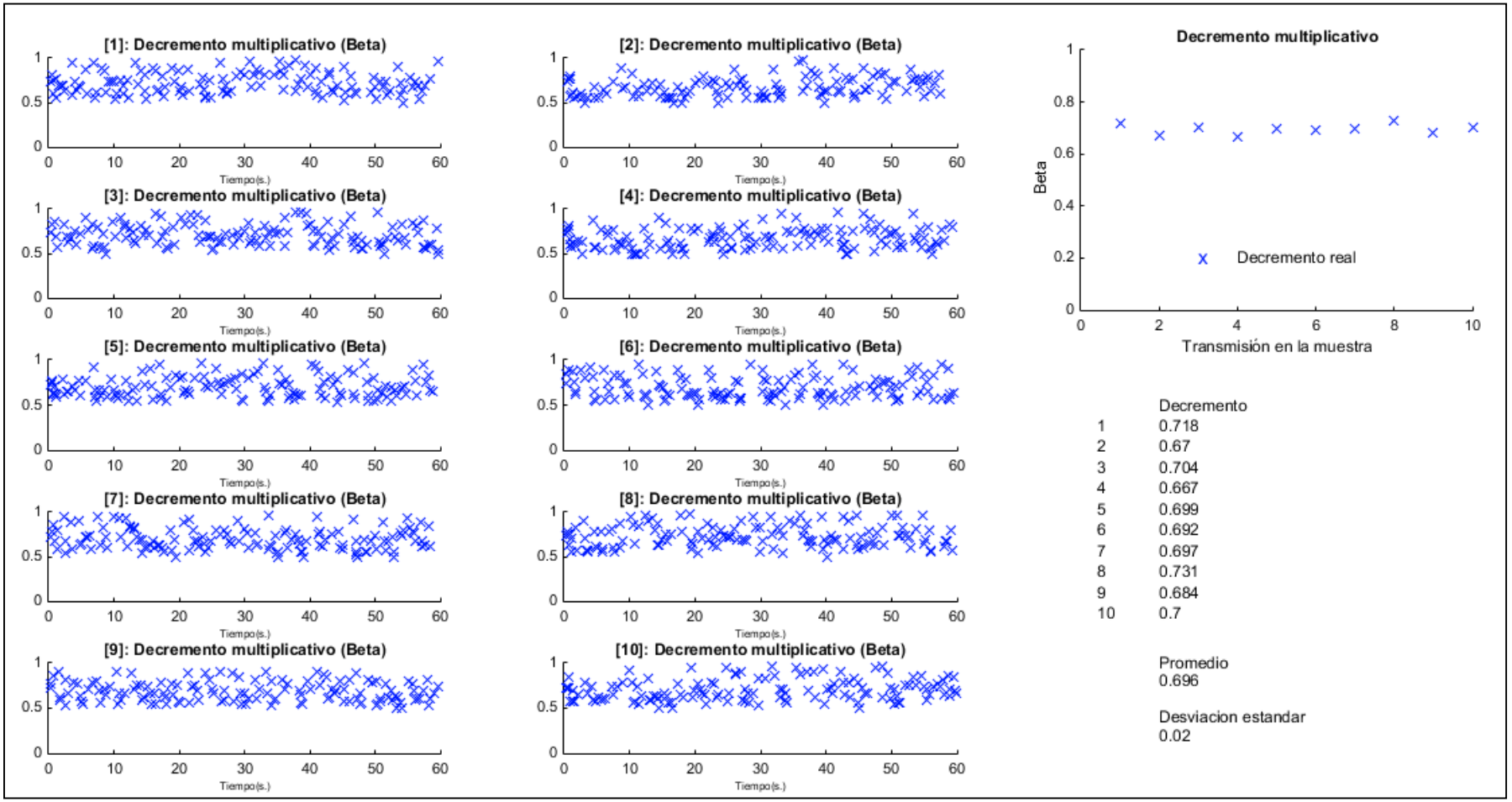
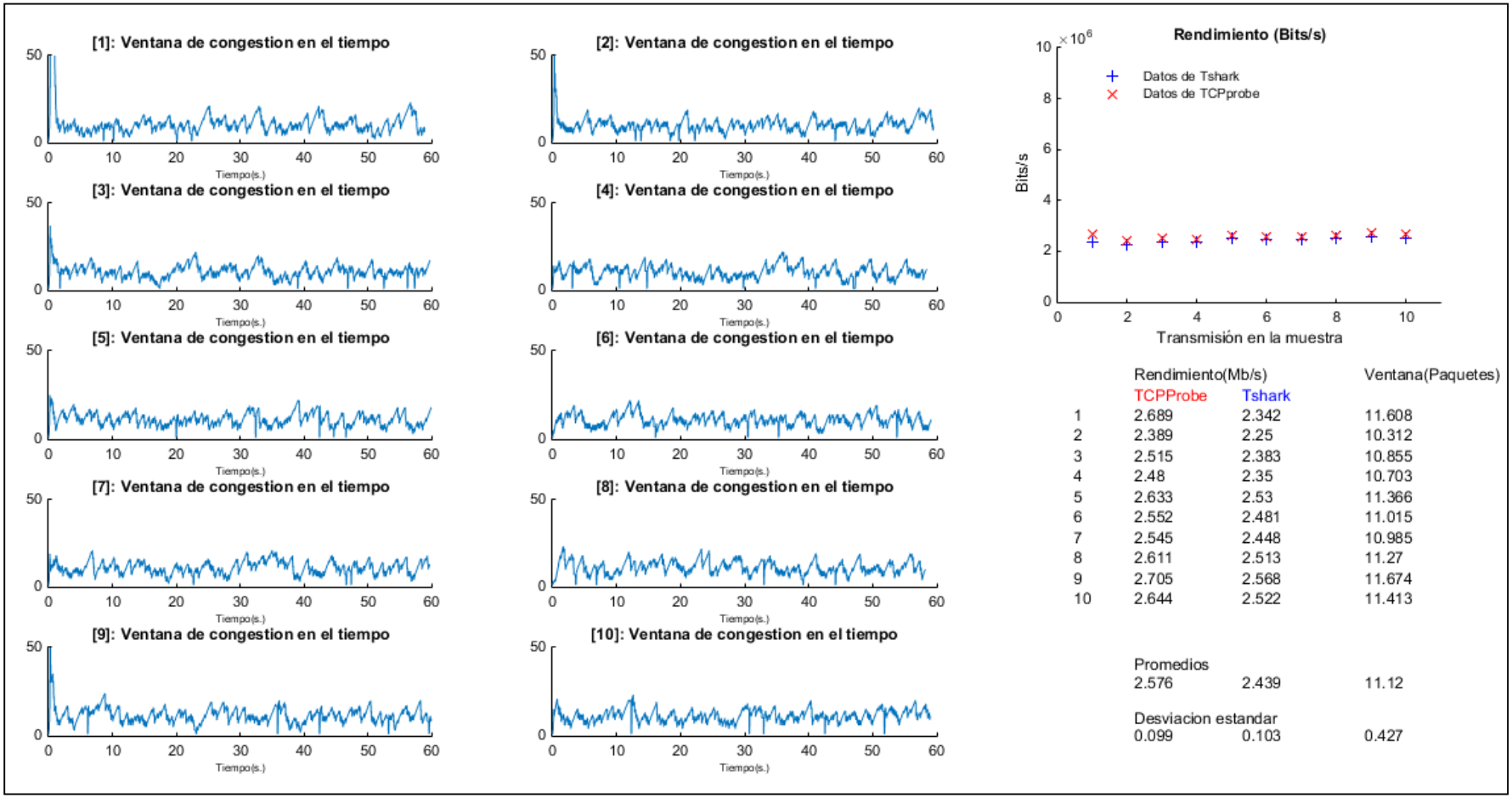


Figura M17. Transmisiones con ESTP con RTT=50ms y pérdidas aleatorias del 1% (decremento real)



	Rendimiento(Mb/s)		Ventana(Paquetes)
	TCPProbe	Tshark	
1	2.689	2.342	11.608
2	2.389	2.25	10.312
3	2.515	2.383	10.855
4	2.48	2.35	10.703
5	2.633	2.53	11.366
6	2.552	2.481	11.015
7	2.545	2.448	10.985
8	2.611	2.513	11.27
9	2.705	2.568	11.674
10	2.644	2.522	11.413
Promedios			
	2.576	2.439	11.12
Desviacion estandar			
	0.099	0.103	0.427

Figura M18. Transmisiones con ESTP con RTT=50ms y pérdidas aleatorias del 2%

(Ventana de congestión y rendimiento)

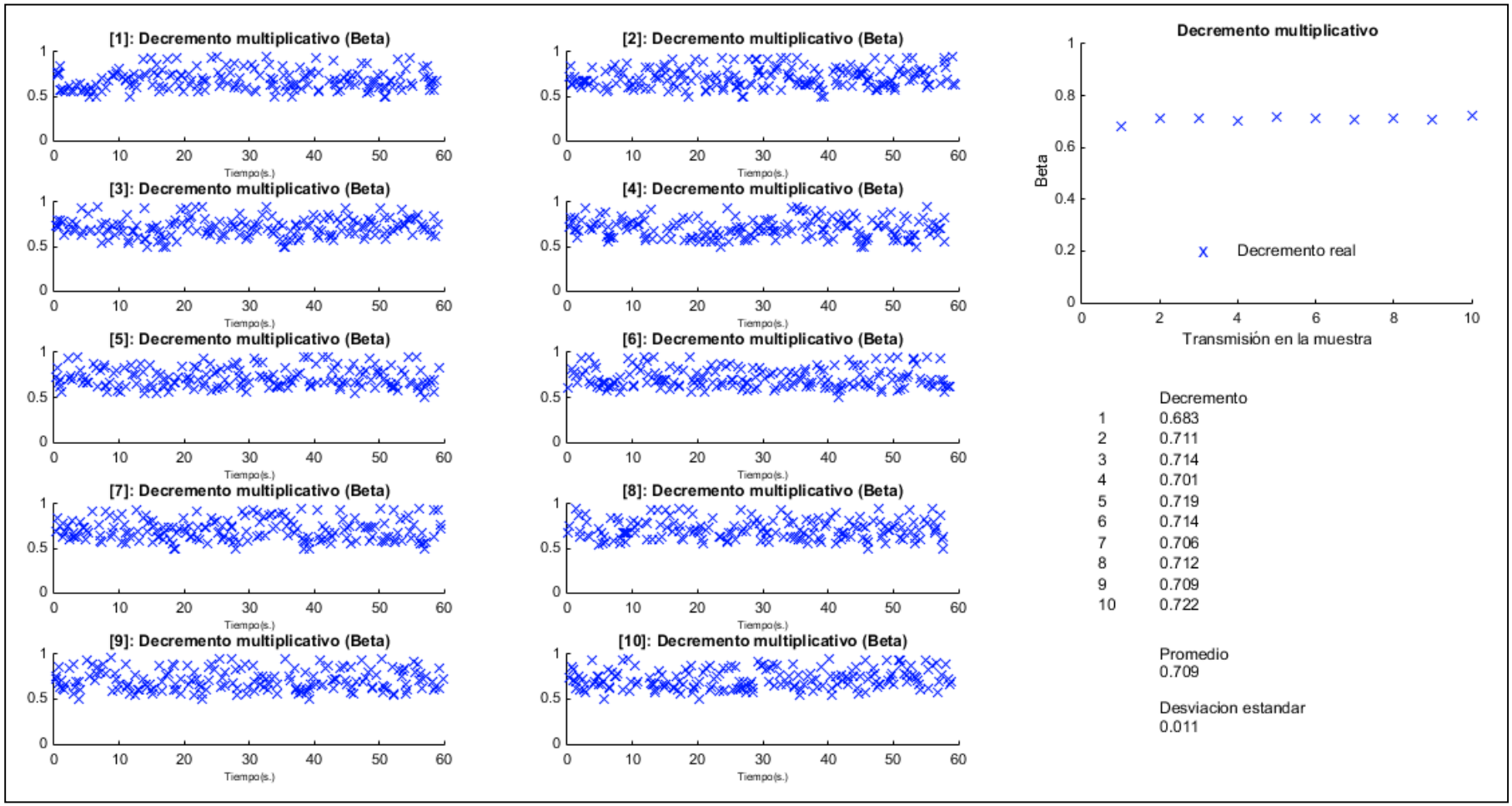
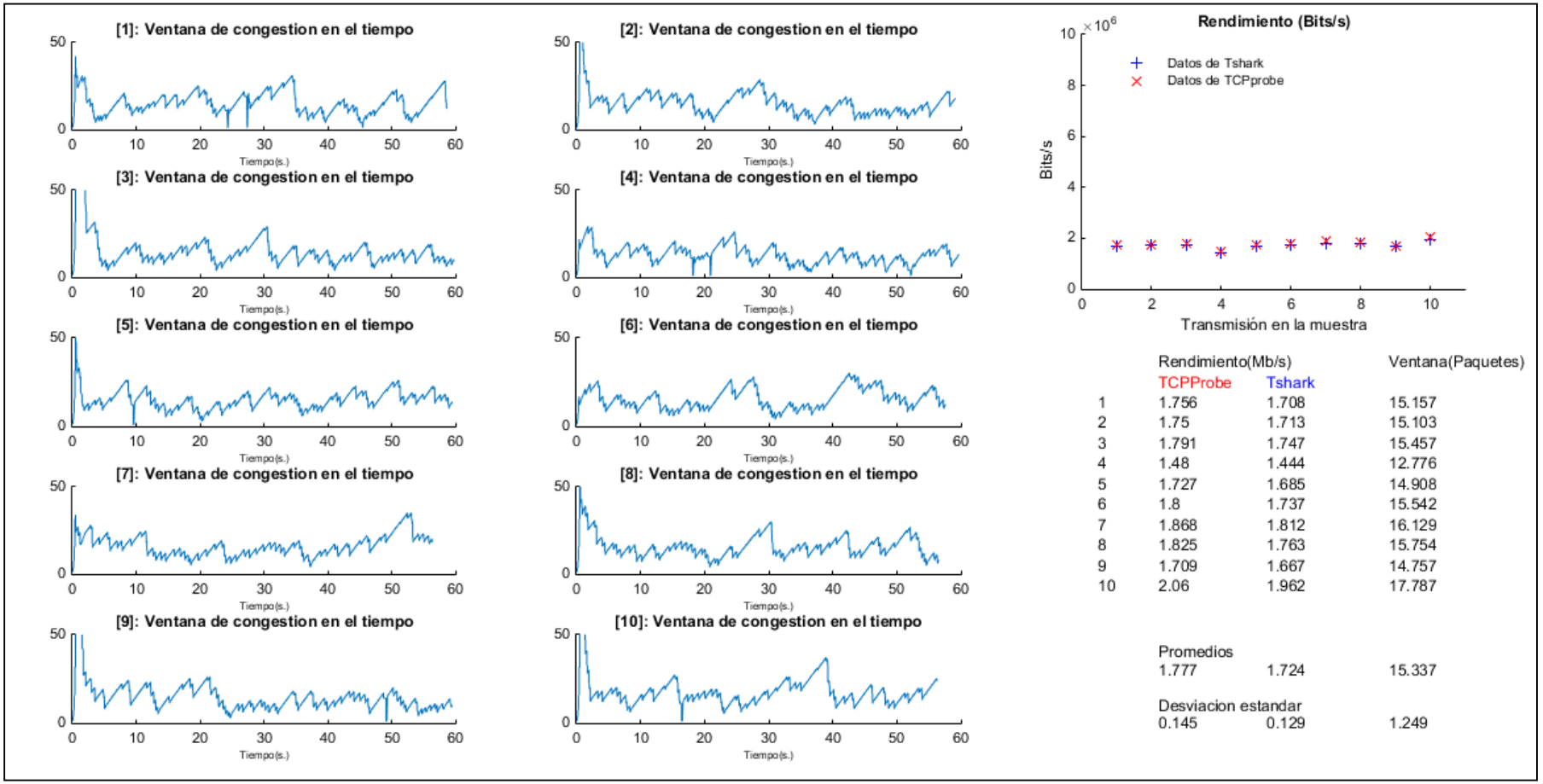


Figura M19. Transmisiones con ESTP con RTT=50ms y pérdidas aleatorias del 2% (decremento real)



	Rendimiento(Mb/s)		Ventana(Paquetes)
	TCPProbe	Tshark	
1	1.756	1.708	15.157
2	1.75	1.713	15.103
3	1.791	1.747	15.457
4	1.48	1.444	12.776
5	1.727	1.685	14.908
6	1.8	1.737	15.542
7	1.868	1.812	16.129
8	1.825	1.763	15.754
9	1.709	1.667	14.757
10	2.06	1.962	17.787
Promedios	1.777	1.724	15.337
Desviacion estandar	0.145	0.129	1.249

Figura M20. Transmisiones con ESTP con RTT=100ms y pérdidas aleatorias del 1%

(Ventana de congestión y rendimiento)

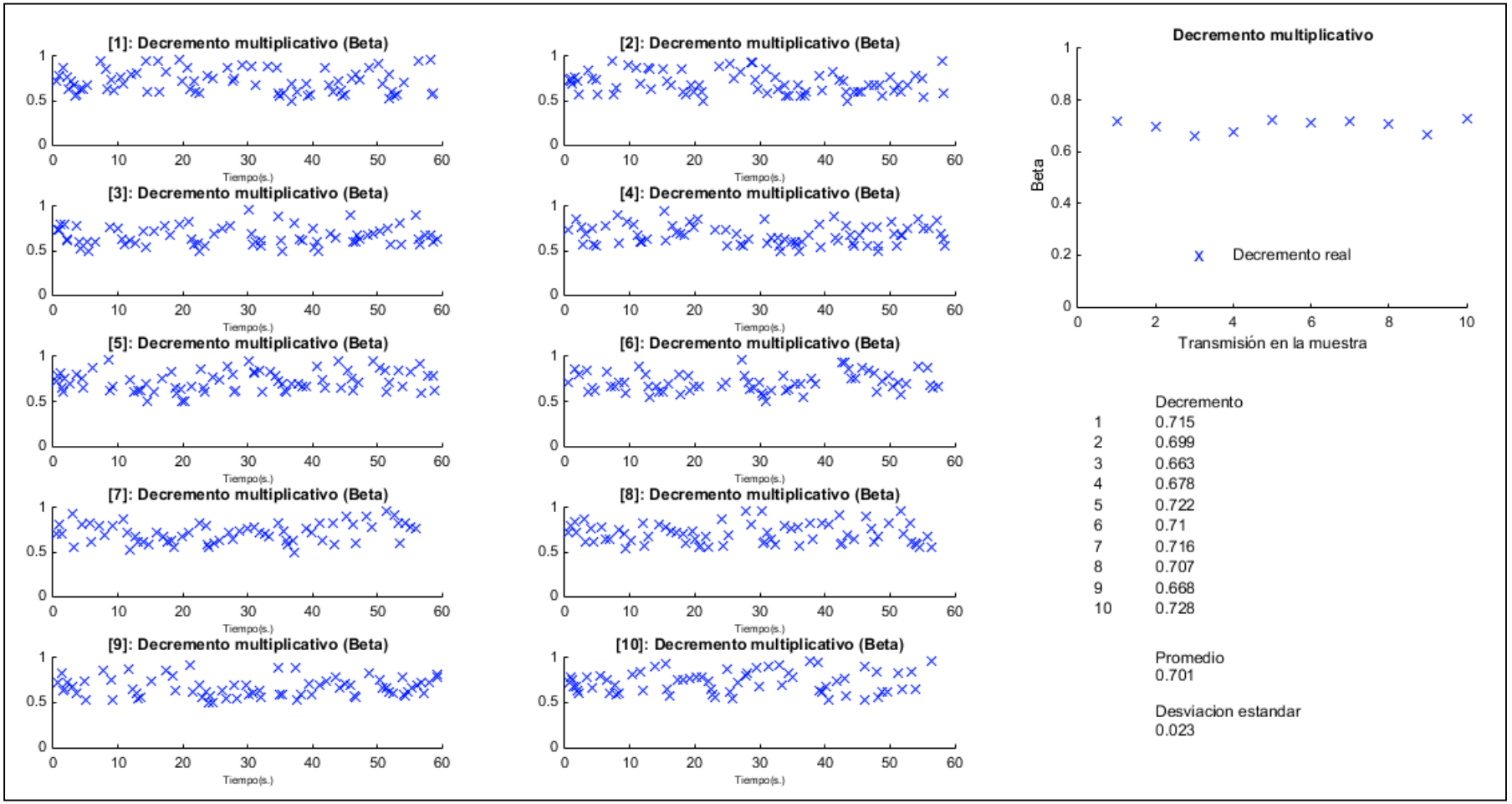
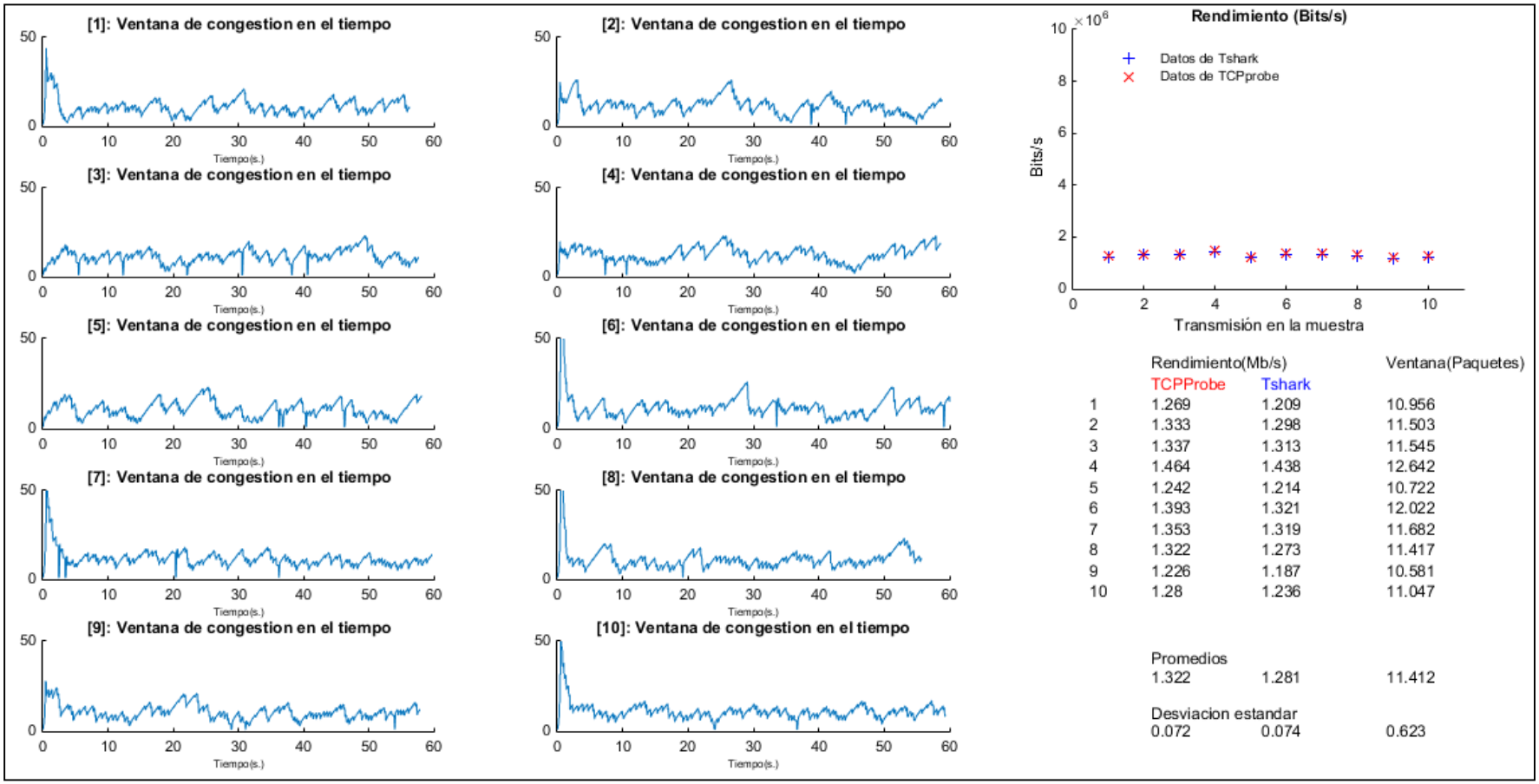


Figura M21. Transmisiones con ESTP con RTT=100ms y pérdidas aleatorias del 1% (decremento real)



	Rendimiento(Mb/s)		Ventana(Paquetes)
	TCPProbe	Tshark	
1	1.269	1.209	10.956
2	1.333	1.298	11.503
3	1.337	1.313	11.545
4	1.464	1.438	12.642
5	1.242	1.214	10.722
6	1.393	1.321	12.022
7	1.353	1.319	11.682
8	1.322	1.273	11.417
9	1.226	1.187	10.581
10	1.28	1.236	11.047
Promedios			
	1.322	1.281	11.412
Desviacion estandar			
	0.072	0.074	0.623

Figura M22. Transmisiones con ESTP con RTT=100ms y pérdidas aleatorias del 2%

(Ventana de congestión y rendimiento)

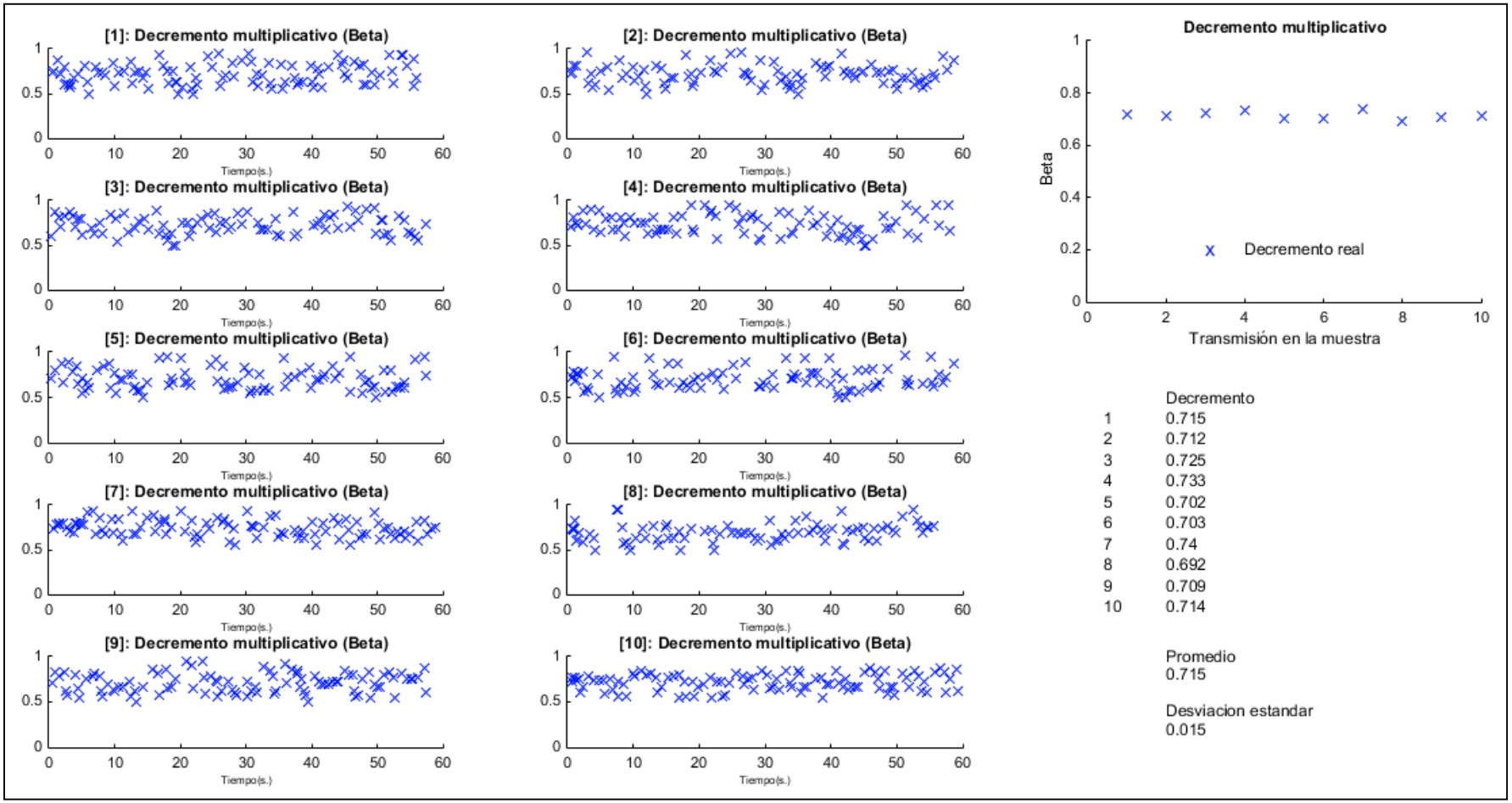


Figura M23. Transmisiones con ESTP con RTT=100ms y pérdidas aleatorias del 2% (decremento real)

ANEXO "N"

USO DE R PARA EVALUACIÓN DE TEST DE HIPÓTESIS

1. TEST DE HIPÓTESIS QUE EVALUA EL COMPORTAMIENTO DEL RENDIMIENTO EN TCP RENO.

Las suposiciones a contrastar son:

Suposición A: Si aumentan las pérdidas para un mismo RTT, el rendimiento disminuye.

Suposición B: Para dos conexiones con el mismo porcentaje de pérdidas, el rendimiento disminuye cuando aumenta el RTT.

1.1. Datos a evaluar

Los datos de los promedios de rendimiento obtenidos para cada tasa de pérdidas de paquetes y para cada valor de RTT son hallados en los archivos: ~/ EVALUACION_R/DatosRENO_A.csv y ~/ EVALUACION_R/DatosRENO_B.csv.

	A	B	C	D
1	RTT	Herramienta	PERDIDA_0.01	PERDIDA_0.02
2	0.05	iPerf	2.47	1.77
3	0.05	Tcpprobe	2.53	1.844
4	0.05	Tshark	2.353	1.726
5	0.1	iPerf	1.31	0.926
6	0.1	Tcpprobe	1.301	0.929
7	0.1	Tshark	1.241	0.886

Figura N1. Archivo ~/ EVALUACION_R/ DatosRENO_A.csv.

	A	B	C	D
1	PERDIDAS	Herramienta	RTT_0.05 s	RTT_0.10 s
2	0.01	iPerf	2.47	1.31
3	0.01	Tcpprobe	2.53	1.301
4	0.01	Tshark	2.353	1.241
5	0.02	iPerf	1.77	0.926
6	0.02	Tcpprobe	1.844	0.929
7	0.02	Tshark	1.726	0.886

Figura N2. Archivo ~/ EVALUACION_R/ DatosRENO_B.csv.

En la figura N1:

Columna A : Valor de RTT.

Columna B : Herramienta de emulación y/o captura empleada.

Columna C : Rendimiento con TCP RENO cuando ocurre una tasa de pérdidas de 1%.

Columna D : Rendimiento con TCP RENO cuando ocurre una tasa de pérdidas de 2%.

En la figura N2:

Columna A : Valor de tasa de pérdida de paquetes.

Columna B : Herramienta de emulación y/o captura empleada.

Columna C : Rendimiento con TCP RENO cuando RTT=0.05s.

Columna D : Rendimiento con TCP RENO cuando RTT=0.10s.

1.2. Script en R para realizar test de hipótesis

El script en R que evalúa los datos presentados es:

~/ EVALUACION_R/RENO.R.

```
HIPOTESIS_A = read.table("home/owl-wkstn-tcp/EVALUACION_R/DatosRENO_A.csv", sep=",", header = TRUE)
HIPOTESIS_B = read.table("home/owl-wkstn-tcp/EVALUACION_R/DatosRENO_B.csv", sep=",", header = TRUE)
#
#
# Test 1
#
# 1) Test1
prop.test (HIPOTESIS_A[1: 3,]$PERDIDA_0.02, HIPOTESIS_A[1: 3,]$PERDIDA_0.01, conf.level = 0.95, correct = FALSE)
#
# 2) Test2
prop.test (HIPOTESIS_A[4: 6,]$PERDIDA_0.02, HIPOTESIS_A[4: 6,]$PERDIDA_0.01, conf.level = 0.95, correct = FALSE)
#
# 3) Test3
prop.test (HIPOTESIS_B[1: 3,]$RTT_0.10.s, HIPOTESIS_B[1: 3,]$RTT_0.05.s, conf.level = 0.95, correct = FALSE)
#
# 4) Test4
prop.test (HIPOTESIS_B[4: 6,]$RTT_0.10.s, HIPOTESIS_B[4: 6,]$RTT_0.05.s, conf.level = 0.95, correct = FALSE)
#
# 5) Test 5: (Rendimiento cuando tasa de pérdidas = 2%) / (Rendimiento cuando tasa de pérdidas =1%) = sqrt(0.01)/sqr(0.02)
cuando RTT=0.05s.
prop.test (c(HIPOTESIS_A[1,]$PERDIDA_0.02,sqrt(0.01)), c(HIPOTESIS_A[1,]$PERDIDA_0.01,sqrt(0.02)), conf.level =
0.95, correct = FALSE)
#
# 6) Test 6: (Rendimiento cuando tasa de pérdidas = 2%) / (Rendimiento cuando tasa de pérdidas =1%) = sqrt(0.01)/sqr(0.02)
cuando RTT=0.10s.
prop.test (c(HIPOTESIS_A[4,]$PERDIDA_0.02,sqrt(0.01)), c(HIPOTESIS_A[4,]$PERDIDA_0.01,sqrt(0.02)), conf.level =
0.95, correct = FALSE)
#
# 7) Test 7: (Rendimiento cuando RTT = 0.10s) / (Rendimiento cuando RTT =0.05s) = 0.05s./0.10s. cuando tasa de pérdidas=1%.
prop.test (c(HIPOTESIS_B[1,]$RTT_0.10.s,0.05), c(HIPOTESIS_B[1,]$RTT_0.05.s,0.10), conf.level = 0.95, correct = FALSE)
#
# 8) Test 8: (Rendimiento cuando RTT = 0.10s) / (Rendimiento cuando RTT =0.05s) = 0.05s./0.10s. cuando tasa de pérdidas=2%.
prop.test (c(HIPOTESIS_B[4,]$RTT_0.10.s,0.05), c(HIPOTESIS_B[4,]$RTT_0.05.s,0.10), conf.level = 0.95, correct = FALSE)
#
# Correlación entre valores de rendimiento
cor(HIPOTESIS_A[1: 3,]$PERDIDA_0.01, HIPOTESIS_A[1: 3,]$PERDIDA_0.02)
cor(HIPOTESIS_A[4: 6,]$PERDIDA_0.01, HIPOTESIS_A[4: 6,]$PERDIDA_0.02)
cor(HIPOTESIS_B[1: 3,]$RTT_0.05.s, HIPOTESIS_B[1: 3,]$RTT_0.10.s)
cor(HIPOTESIS_B[1: 3,]$RTT_0.05.s, HIPOTESIS_B[1: 3,]$RTT_0.10.s)
```

Figura N3. Archivo script: ~/ EVALUACION_R/RENO.R

1.3. Resultados en R

Los resultados observados en la consola de R se muestran en las siguientes imágenes:

```
> HIPOTESIS_A = read.table("D:/TESIS/EVALUACION_R/DatosRENO_A.csv", sep=",", header = TRUE)
> HIPOTESIS_B = read.table("D:/TESIS/EVALUACION_R/DatosRENO_B.csv", sep=",", header = TRUE)
> #
> # Test 1
> #
> # 1) Test1
> prop.test (HIPOTESIS_A[1: 3,]$PERDIDA_0.02, HIPOTESIS_A[1: 3,]$PERDIDA_0.01, conf.level = 0.95, correct = FALSE)

        3-sample test for equality of proportions without continuity
        correction

data:  HIPOTESIS_A[1:3, ]$PERDIDA_0.02 out of HIPOTESIS_A[1:3, ]$PERDIDA_0.01
X-squared = 0.0018709, df = 2, p-value = 0.9991
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3
0.7165992 0.7288538 0.7335317

Warning message:
In prop.test(HIPOTESIS_A[1:3, ]$PERDIDA_0.02, HIPOTESIS_A[1:3, ]$PERDIDA_0.01,  :
  Chi-squared approximation may be incorrect
> #
> # 2) Test2
> prop.test (HIPOTESIS_A[4: 6,]$PERDIDA_0.02, HIPOTESIS_A[4: 6,]$PERDIDA_0.01, conf.level = 0.95, correct = FALSE)

        3-sample test for equality of proportions without continuity
        correction

data:  HIPOTESIS_A[4:6, ]$PERDIDA_0.02 out of HIPOTESIS_A[4:6, ]$PERDIDA_0.01
X-squared = 0.00021445, df = 2, p-value = 0.9999
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3
0.7068702 0.7140661 0.7139404

Warning message:
In prop.test(HIPOTESIS_A[4:6, ]$PERDIDA_0.02, HIPOTESIS_A[4:6, ]$PERDIDA_0.01,  :
  Chi-squared approximation may be incorrect
> #
> # 3) Test3
> prop.test (HIPOTESIS_B[1: 3,]$RTT_0.10.s, HIPOTESIS_B[1: 3,]$RTT_0.05.s, conf.level = 0.95, correct = FALSE)

        3-sample test for equality of proportions without continuity
        correction

data:  HIPOTESIS_B[1:3, ]$RTT_0.10.s out of HIPOTESIS_B[1:3, ]$RTT_0.05.s
X-squared = 0.0014787, df = 2, p-value = 0.9993
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3
0.5303644 0.5142292 0.5274118

Warning message:
In prop.test(HIPOTESIS_B[1:3, ]$RTT_0.10.s, HIPOTESIS_B[1:3, ]$RTT_0.05.s,  :
  Chi-squared approximation may be incorrect
> #
> # 4) Test4
> prop.test (HIPOTESIS_B[4: 6,]$RTT_0.10.s, HIPOTESIS_B[4: 6,]$RTT_0.05.s, conf.level = 0.95, correct = FALSE)

        3-sample test for equality of proportions without continuity
        correction

data:  HIPOTESIS_B[4:6, ]$RTT_0.10.s out of HIPOTESIS_B[4:6, ]$RTT_0.05.s
X-squared = 0.001356, df = 2, p-value = 0.9993
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3
0.5231638 0.5037961 0.5133256

Warning message:
In prop.test(HIPOTESIS_B[4:6, ]$RTT_0.10.s, HIPOTESIS_B[4:6, ]$RTT_0.05.s,  :
  Chi-squared approximation may be incorrect
> #
```

Figura N4. Resultados de test de proporciones (Parte 1).

```
> # 5) Test 5: (Rendimiento cuando tasa de pérdidas = 2%) / (Rendimiento cuando tasa de pérdidas =1%) = sqrt(0.01)/sqr(0.02) cuando RTT=0.05s.
> prop.test (c(HIPOTESIS_A[1,]$PERDIDA_0.02,sqrt(0.01)), c(HIPOTESIS_A[1,]$PERDIDA_0.01,sqrt(0.02)), conf.level = 0.95, correct = FALSE)

        2-sample test for equality of proportions without continuity correction

data:  c(HIPOTESIS_A[1, ]$PERDIDA_0.02, sqrt(0.01)) out of c(HIPOTESIS_A[1, ]$PERDIDA_0.01, sqrt(0.02))
X-squared = 5.9284e-05, df = 1, p-value = 0.9939
alternative hypothesis: two.sided
95 percent confidence interval:
 -1  1
sample estimates:
   prop 1    prop 2
0.7165992 0.7071068

Warning message:
In prop.test(c(HIPOTESIS_A[1, ]$PERDIDA_0.02, sqrt(0.01)), c(HIPOTESIS_A[1,  :
  Chi-squared approximation may be incorrect
> #
> # 6) Test 6: (Rendimiento cuando tasa de pérdidas = 2%) / (Rendimiento cuando tasa de pérdidas =1%) = sqrt(0.01)/sqr(0.02) cuando RTT=0.10s.
> prop.test (c(HIPOTESIS_A[4,]$PERDIDA_0.02,sqrt(0.01)), c(HIPOTESIS_A[4,]$PERDIDA_0.01,sqrt(0.02)), conf.level = 0.95, correct = FALSE)

        2-sample test for equality of proportions without continuity correction

data:  c(HIPOTESIS_A[4, ]$PERDIDA_0.02, sqrt(0.01)) out of c(HIPOTESIS_A[4, ]$PERDIDA_0.01, sqrt(0.02))
X-squared = 3.4472e-08, df = 1, p-value = 0.9999
alternative hypothesis: two.sided
95 percent confidence interval:
 -1  1
sample estimates:
   prop 1    prop 2
0.7068702 0.7071068

Warning message:
In prop.test(c(HIPOTESIS_A[4, ]$PERDIDA_0.02, sqrt(0.01)), c(HIPOTESIS_A[4,  :
  Chi-squared approximation may be incorrect
> #
> # 7) Test 7: (Rendimiento cuando RTT = 0.10s) / (Rendimiento cuando RTT =0.05s) = 0.05s./0.10s. cuando tasa de pérdidas=1%.
> prop.test (c(HIPOTESIS_B[1,]$RTT_0.10.s,0.05), c(HIPOTESIS_B[1,]$RTT_0.05.s,0.10), conf.level = 0.95, correct = FALSE)

        2-sample test for equality of proportions without continuity correction

data:  c(HIPOTESIS_B[1, ]$RTT_0.10.s, 0.05) out of c(HIPOTESIS_B[1, ]$RTT_0.05.s, 0.1)
X-squared = 0.00035566, df = 1, p-value = 0.985
alternative hypothesis: two.sided
95 percent confidence interval:
 -1  1
sample estimates:
   prop 1    prop 2
0.5303644 0.5000000

Warning message:
In prop.test(c(HIPOTESIS_B[1, ]$RTT_0.10.s, 0.05), c(HIPOTESIS_B[1,  :
  Chi-squared approximation may be incorrect
> #
> # 8) Test 8: (Rendimiento cuando RTT = 0.10s) / (Rendimiento cuando RTT =0.05s) = 0.05s./0.10s. cuando tasa de pérdidas=2%.
> prop.test (c(HIPOTESIS_B[4,]$RTT_0.10.s,0.05), c(HIPOTESIS_B[4,]$RTT_0.05.s,0.10), conf.level = 0.95, correct = FALSE)

        2-sample test for equality of proportions without continuity correction

data:  c(HIPOTESIS_B[4, ]$RTT_0.10.s, 0.05) out of c(HIPOTESIS_B[4, ]$RTT_0.05.s, 0.1)
X-squared = 0.00020354, df = 1, p-value = 0.9886
alternative hypothesis: two.sided
95 percent confidence interval:
 -1  1
sample estimates:
   prop 1    prop 2
0.5231638 0.5000000

Warning message:
In prop.test(c(HIPOTESIS_B[4, ]$RTT_0.10.s, 0.05), c(HIPOTESIS_B[4,  :
  Chi-squared approximation may be incorrect
> #
```

Figura N5. Resultados de test de proporciones (Parte 2).

```
> # Correlación entre valores de rendimiento
> cor(HIPOTESIS_A[1: 3,]$PERDIDA_0.01, HIPOTESIS_A[1: 3,]$PERDIDA_0.02)
[1] 0.9461814
> cor(HIPOTESIS_A[4: 6,]$PERDIDA_0.01, HIPOTESIS_A[4: 6,]$PERDIDA_0.02)
[1] 0.9833421
> cor(HIPOTESIS_B[1: 3,]$RTT_0.05.s, HIPOTESIS_B[1: 3,]$RTT_0.10.s)
[1] 0.8960396
> cor(HIPOTESIS_B[1: 3,]$RTT_0.05.s, HIPOTESIS_B[1: 3,]$RTT_0.10.s)
[1] 0.8960396
```

Figura N6. Resultados de correlación.

2. TEST DE HIPÓTESIS QUE COMPARA GAIMD CON TCP RENO.

Las suposiciones a contrastar son:

Suposición A: El rendimiento promedio es mayor cuando se emplea GAIMD en comparación a TCP RENO.

Suposición B: La ventana de congestión promedio es mayor cuando se emplea GAIMD en comparación a TCP RENO.

2.1. Datos a evaluar

Los datos de los promedios de rendimiento y ventana de congestión de cada muestra son colocados en el archivo ~/ EVALUACION_R/GAIMD_vs_RENO.csv.

	A	B	C	D	E	F	G	H	I
1	Muestra	RENO	GAIMD	RENO	GAIMD	RENO	GAIMD	RENO	GAIMD
2	1	2.47	3.254	2.53	3.315	2.353	3.144	10.918	14.307
3	2	1.77	2.353	1.844	2.396	1.726	2.285	7.961	10.343
4	3	1.31	1.663	1.301	1.693	1.301	1.649	11.231	14.613
5	4	0.926	1.21	0.929	1.2	0.886	1.155	8.021	10.36

Figura N7. Archivo ~/ EVALUACION_R/ GAIMD_vs_RENO.csv.

En la figura N7:

Columna A : Número de muestra

Columna B : Rendimiento con TCP RENO obtenido desde datos de captura de iPerf.

Columna C : Rendimiento con GAIMD obtenido desde datos de captura de iPerf.

Columna D : Rendimiento con TCP RENO calculado con datos de captura de TCP Probe

Columna E : Rendimiento con GAIMD calculado con datos de captura de TCP Probe

Columna F : Rendimiento con TCP RENO calculado desde datos de captura de tshark.

Columna G : Rendimiento con GAIMD calculado desde datos de captura de tshark.

Columna H : Ventana de congestión promedio con TCP RENO

Columna I : Ventana de congestión promedio con GAIMD

2.2. Script en R para realizar test de hipótesis

El script en R que evalúa los datos del archivo .csv anterior es:

~/ EVALUACION_R/ComparacionGAIMD_RENO.R.

```
RENO_GAIMD = read.table("home/owl-wkstn-tcp/EVALUACION_R/GAIMD_vs_RENO.csv", sep=",", header = TRUE)
#
Muestra = c(1,4)
R_iperf_reno = RENO_GAIMD$RENO
```

```
R_iperf_gaimd = RENO_GAIMD$GAIMD
R_probe_reno = RENO_GAIMD$RENO.1
R_probe_gaimd = RENO_GAIMD$GAIMD.1
R_tshark_reno = RENO_GAIMD$RENO.2
R_tshark_gaimd = RENO_GAIMD$GAIMD.2
cwnd_reno = RENO_GAIMD$RENO.3
cwnd_gaimd = RENO_GAIMD$GAIMD.3
R_iperf_reno #Vector de rendimiento promedio obtenido por iPerf empleando TCP RENO
R_iperf_gaimd #Vector de rendimiento promedio obtenido por iPerf empleando GAIMD
R_probe_reno #Vector de rendimiento promedio obtenido de los datos capturados por TCP Probe empleando TCP RENO
R_probe_gaimd #Vector de rendimiento promedio obtenido de los datos capturados por TCP Probe empleando GAIMD
R_tshark_reno #Vector de rendimiento promedio obtenido de los datos capturados por TSHARK empleando TCP RENO
R_tshark_gaimd #Vector de rendimiento promedio obtenido de los datos capturados por TSHARK empleando GAIMD
cwnd_reno #Vector de ventana de congestion promedio con TCP RENO
cwnd_gaimd #Vector de ventana de congestion promedio con GAIMD
#
# Test de hipotesis para comprobar si rendimiento de GAIMD es mejor que rendimiento de TCP RENO
#
# 1) Empleando datos de iperf
t.test (R_iperf_gaimd, R_iperf_reno, conf.level = 0.95, paired = TRUE)
# 2) Empleando datos de tcpprobe
t.test (R_probe_gaimd, R_probe_reno, conf.level = 0.95, paired = TRUE)
# 3) Empleando datos de tshark
t.test (R_tshark_gaimd, R_tshark_reno, conf.level = 0.95, paired = TRUE)
#
# Test de hipotesis para comprobar que CWND es mayor cuando se emplea GAIMD
t.test (cwnd_gaimd, cwnd_reno, conf.level = 0.95, paired = TRUE)
#
# Correlación entre valores de rendimiento y CWND de GAIMD y TCP RENO.
cor(R_iperf_gaimd, R_iperf_reno)
cor(R_probe_gaimd, R_probe_reno)
cor(R_tshark_gaimd, R_tshark_reno)
cor(cwnd_reno,cwnd_gaimd)
```

Figura N8. Archivo script: ~/ EVALUACION_R/ComparacionGAIMD_RENO.R

2.3. Resultados en R

Los resultados observados en la consola de R se muestran en las siguientes imágenes:

```
> t.test (R_iperf_gaimd, R_iperf_reno, conf.level = 0.95, paired = TRUE)

        Paired t-test

data:  R_iperf_gaimd and R_iperf_reno
t = 4.3968, df = 3, p-value = 0.02181
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.138371 0.863629
sample estimates:
mean of the differences
                  0.501

> # 2) Empleando datos de tcpprobe
> t.test (R_probe_gaimd, R_probe_reno, conf.level = 0.95, paired = TRUE)

        Paired t-test

data:  R_probe_gaimd and R_probe_reno
t = 4.5017, df = 3, p-value = 0.02047
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.1465311 0.8534689
sample estimates:
mean of the differences
                    0.5

> # 3) Empleando datos de tshark
> t.test (R_tshark_gaimd, R_tshark_reno, conf.level = 0.95, paired = TRUE)

        Paired t-test

data:  R_tshark_gaimd and R_tshark_reno
t = 4.2019, df = 3, p-value = 0.0246
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.1193055 0.8641945
sample estimates:
mean of the differences
                0.49175
```

Figura N9. Resultados de test que contrasta la hipótesis A.

Según la Figura N9: El rendimiento empleando GAIMD es mayor que si se emplea TCP RENO.

```
> t.test (cwnd_gaimd, cwnd_reno, conf.level = 0.95, paired = TRUE)

        Paired t-test

data:  cwnd_gaimd and cwnd_reno
t = 9.7052, df = 3, p-value = 0.002323
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 1.930914 3.815086
sample estimates:
mean of the differences
                  2.873
```

Figura N10. Resultados de test que contrasta la hipótesis B.

Según la Figura N10: La ventana de congestión promedio empleando GAIMD es mayor que si se emplea TCP RENO.

```
> cor(R_iperf_gaimd, R_iperf_reno)
[1] 0.9993329
> cor(R_probe_gaimd, R_probe_reno)
[1] 0.9999747
> cor(R_tshark_gaimd, R_tshark_reno)
[1] 0.9993107
> cor(cwnd_reno,cwnd_gaimd)
[1] 0.9997605
```

Figura N11. Correlación entre valores de rendimiento y ventana de congestión promedio con GAIMD y TCP RENO.

De los valores que se observan en la figura N11:

- Existe una relación lineal entre rendimientos de TCP RENO y GAIMD.
- Existe una relación lineal entre ventanas de congestión promedio de TCP RENO y GAIMD.

3. TEST DE PROPORCIONES PARA CONSTRUIR Y VALIDAR UNA RELACIÓN ENTRE TCP RENO Y GAIMD.

3.1. Datos a evaluar.

Son dados en ~/ EVALUACION_R/GAIMD_vs_RENO.csv (Ver figura M7).

3.2. Script para realizar los tests de proporciones

El script en R es: ~/ EVALUACION_R/relacion_renogaimd.R

```
RENO_GAIMD = read.table("home/owl-wkstn-tcp/GAIMD_vs_RENO.csv", sep=",", header = TRUE)
#
R_iperf_reno = RENO_GAIMD$RENO #rendimiento promedio obtenido por iPerf empleando TCP RENO
R_iperf_gaimd = RENO_GAIMD$GAIMD #rendimiento promedio obtenido por iPerf empleando GAIMD
R_probe_reno = RENO_GAIMD$RENO.1 #rendimiento promedio obtenido de los datos capturados por TCP Probe empleando TCP RENO
R_probe_gaimd = RENO_GAIMD$GAIMD.1 #rendimiento promedio obtenido de los datos capturados por TCP Probe empleando TCP RENO
R_tshark_reno = RENO_GAIMD$RENO.2 #rendimiento promedio obtenido de los datos capturados por TSHARK empleando TCP RENO
R_tshark_gaimd = RENO_GAIMD$GAIMD.2 #rendimiento promedio obtenido de los datos capturados por TSHARK empleando GAIMD
cwnd_reno = RENO_GAIMD$RENO.3 # ventana de congestion promedio con TCP RENO
cwnd_gaimd = RENO_GAIMD$GAIMD.3 # ventana de congestion promedio con GAIMD
#
# Test de proporciones para rendimiento
#
# Empleando datos de iperf (k1)
prop.test (R_iperf_reno, R_iperf_gaimd, conf.level = 0.95, correct = FALSE)
# Empleando datos de tcpprobe (k2)
prop.test (R_probe_reno, R_probe_gaimd, conf.level = 0.95, correct = FALSE)
# Empleando datos de tshark (k3)
prop.test (R_tshark_reno, R_tshark_gaimd, conf.level = 0.95, correct = FALSE)
# Test de proporciones para CWND. (k4)
prop.test (cwnd_reno, cwnd_gaimd, conf.level = 0.95, correct = FALSE)
#
# Tests de proporciones de validación.
#
# Empleando datos de iperf (k*1)
prop.test (c(R_iperf_reno,1), c(R_iperf_gaimd,1.303), conf.level = 0.95, correct = FALSE)
# Empleando datos de tcpprobe (k*2)
prop.test (c(R_probe_reno,1), c(R_probe_gaimd,1.303), conf.level = 0.95, correct = FALSE)
# Empleando datos de tshark (k*3)
prop.test (c(R_tshark_reno,1), c(R_tshark_gaimd,1.303), conf.level = 0.95, correct = FALSE)
# Test de proporciones para CWND. (k*4)
prop.test (c(cwnd_reno,1), c(cwnd_gaimd,1.303), conf.level = 0.95, correct = FALSE)
```

Figura N12. Archivo script: ~/ EVALUACION_R/relacion_renogaimd.R

3.3. Resultados en R.

Los resultados observados en la consola de R se muestran en las siguientes imágenes:

```
> # Test de proporciones para rendimiento
> #
> # Empleando datos de iperf (k1)
> prop.test (R_iperf_reno, R_iperf_gaimd, conf.level = 0.95, correct = FALSE)

        4-sample test for equality of proportions without continuity correction

data:  R_iperf_reno out of R_iperf_gaimd
X-squared = 0.0074413, df = 3, p-value = 0.9998
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4
0.7590658 0.7522312 0.7877330 0.7652893

Warning message:
In prop.test(R_iperf_reno, R_iperf_gaimd, conf.level = 0.95, correct = FALSE) :
  Chi-squared approximation may be incorrect
> # Empleando datos de tcpprobe (k2)
> prop.test (R_probe_reno, R_probe_gaimd, conf.level = 0.95, correct = FALSE)

        4-sample test for equality of proportions without continuity correction

data:  R_probe_reno out of R_probe_gaimd
X-squared = 0.00071158, df = 3, p-value = 1
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4
0.7631976 0.7696160 0.7684584 0.7741667

Warning message:
In prop.test(R_probe_reno, R_probe_gaimd, conf.level = 0.95, correct = FALSE) :
  Chi-squared approximation may be incorrect
> # Empleando datos de tshark (k3)
> prop.test (R_tshark_reno, R_tshark_gaimd, conf.level = 0.95, correct = FALSE)

        4-sample test for equality of proportions without continuity correction

data:  R_tshark_reno out of R_tshark_gaimd
X-squared = 0.010467, df = 3, p-value = 0.9997
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4
0.7484097 0.7553611 0.7889630 0.7670996

Warning message:
In prop.test(R_tshark_reno, R_tshark_gaimd, conf.level = 0.95, correct = FALSE) :
  Chi-squared approximation may be incorrect
> # Test de proporciones para CWND. (k4)
> prop.test (cwnd_reno, cwnd_gaimd, conf.level = 0.95, correct = FALSE)

        4-sample test for equality of proportions without continuity correction

data:  cwnd_reno out of cwnd_gaimd
X-squared = 0.0043163, df = 3, p-value = 0.9999
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4
0.7631229 0.7696993 0.7685622 0.7742278

Warning message:
In prop.test(cwnd_reno, cwnd_gaimd, conf.level = 0.95, correct = FALSE) :
  Chi-squared approximation may be incorrect
```

Figura N13. Resultados de test de proporciones que relaciona TCP RENO con GAIMD

```
> # Tests de proporciones de validación.
> #
> # Empleando datos de iperf (k1*)
> prop.test (c(R_iperf_reno,1), c(R_iperf_gaimd,1.303), conf.level = 0.95, correct = FALSE)

        5-sample test for equality of proportions without continuity correction

data:  c(R_iperf_reno, 1) out of c(R_iperf_gaimd, 1.303)
X-squared = 0.0075419, df = 4, p-value = 1
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4    prop 5
0.7590658 0.7522312 0.7877330 0.7652893 0.7674597

Warning message:
In prop.test(c(R_iperf_reno, 1), c(R_iperf_gaimd, 1.303), conf.level = 0.95,  :
  Chi-squared approximation may be incorrect
> # Empleando datos de tcpprobe (k2*)
> prop.test (c(R_probe_reno,1), c(R_probe_gaimd,1.303), conf.level = 0.95, correct = FALSE)

        5-sample test for equality of proportions without continuity correction

data:  c(R_probe_reno, 1) out of c(R_probe_gaimd, 1.303)
X-squared = 0.00071161, df = 4, p-value = 1
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4    prop 5
0.7631976 0.7696160 0.7684584 0.7741667 0.7674597

Warning message:
In prop.test(c(R_probe_reno, 1), c(R_probe_gaimd, 1.303), conf.level = 0.95,  :
  Chi-squared approximation may be incorrect
> # Empleando datos de tshark (k3*)
> prop.test (c(R_tshark_reno,1), c(R_tshark_gaimd,1.303), conf.level = 0.95, correct = FALSE)

        5-sample test for equality of proportions without continuity correction

data:  c(R_tshark_reno, 1) out of c(R_tshark_gaimd, 1.303)
X-squared = 0.010746, df = 4, p-value = 1
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4    prop 5
0.7484097 0.7553611 0.7889630 0.7670996 0.7674597

Warning message:
In prop.test(c(R_tshark_reno, 1), c(R_tshark_gaimd, 1.303), conf.level = 0.95,  :
  Chi-squared approximation may be incorrect
> # Test de proporciones para CWND. (k4*)
> prop.test (c(cwnd_reno,1), c(cwnd_gaimd,1.303), conf.level = 0.95, correct = FALSE)

        5-sample test for equality of proportions without continuity correction

data:  c(cwnd_reno, 1) out of c(cwnd_gaimd, 1.303)
X-squared = 0.0043225, df = 4, p-value = 1
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4    prop 5
0.7631229 0.7696993 0.7685622 0.7742278 0.7674597

Warning message:
In prop.test(c(cwnd_reno, 1), c(cwnd_gaimd, 1.303), conf.level = 0.95,  :
  Chi-squared approximation may be incorrect
```

Figura N14. Resultados de test de proporciones que valida la relación entre TCP RENO y GAIMD

4. TEST DE HIPÓTESIS QUE EVALUA "β" EN ESTP.

Las suposiciones a contrastar son:

Suposición A: Existe una alta correlación positiva entre β de ejecución, β teórico y el decremento real.

Suposición B: β de ejecución ≈ β teórico.

4.1. Datos a evaluar

Los datos de los promedios de rendimiento y ventana de congestión de cada muestra son colocados en el archivo:

~/SIMULACION/datatcp_ESTP_taufijo_variosdelta.csv.

	A	B	C	D	E	F
1	1	0.9851	0.9233	4.2519	0.5473	0.5005
2	2	1.6709	1.6041	7.2121	0.598	0.5892
3	3	2.191	2.0375	9.4572	0.6458	0.6365
4	4	2.9096	2.7608	12.5585	0.6902	0.7128
5	5	3.4816	3.3292	15.0277	0.7303	0.7487
6	6	4.1937	4.0456	18.1013	0.7679	0.7879
7	7	4.9424	4.7381	21.3331	0.8017	0.8167
8	8	5.6827	5.5041	24.5284	0.8315	0.839
9	9	6.4461	6.2337	27.8232	0.8577	0.8568
10	10	7.2694	6.9991	31.377	0.8804	0.8717
11	0	3.9773	3.8175	17.167	0.7351	0.736
12	0	2.0994	2.0413	9.0618	0.113	0.1249

Figura N15. Archivo ~/SIMULACION /datatcp_ESTP_taufijo_variosdelta.csv.

En la figura N15:

Columna A : Número de orden de la transmisión en la muestra.

Columna B : Rendimiento con ESTP calculado con datos de captura de TCP Probe

Columna C : Rendimiento con ESTP calculado desde datos de captura de tshark.

Columna D : Ventana de congestión promedio con ESTP

Columna E : β de ejecución al emplear ESTP.

Columna F : Decremento real al emplear ESTP.

4.2. Script en R para realizar test de hipótesis

El script en R que evalúa los datos del archivo .csv anterior es:

~/ EVALUACION_R/test_beta.R

```
ESTP1 = read.table(“home/owl-wkstn-tcp/SIMULACION /datatcp_ESTP_taufijo_varios delta.csv", sep=",")
colnames(ESTP1)= c("N","thrput_tcpprobe","thrput_tshark","cwnd","beta_ejecucion","decremento_real")
#
ESTP1[ 1: 10,]
#
# Decremento real = (Ventana despues de la congestion)/(Ventana al momento de la congestion)
decremento_real = ESTP1[ 1: 10,]$decremento_real
#
# Valor de Beta de ejecucion para ESTP, obtenido de Tshark
beta_ejecucion = ESTP1[ 1: 10,]$beta_ejecucion
#
# Cálculo de valor de beta teorico
delta = c(20,40,60,80,100,120,140,160,180,200)
tau = 100
beta_teorico = c(1,10)
for (i in 1:10){
beta_teorico[i] = 1/(exp(-(delta[i]-1)/tau)+1)
}
#
beta = data.frame(cbind(beta_teorico,beta_ejecucion,decremento_real))
# Cálculo de coeficiente de correlacion entre los valores de beta teorico, beta de ejecución y decremento real.
cor(beta)
#
# Test de hipotesis de igualdad entre beta_teorico y beta_ejecucion
#
t.test(beta_teorico,beta_ejecucion, conf.level = 0.95, paired = TRUE)
```

Figura M16. Archivo ~/EVALUACION_R/ test_beta.R.

4.3. Resultados en R

Los resultados observados en la consola de R se muestran en las siguientes imágenes:

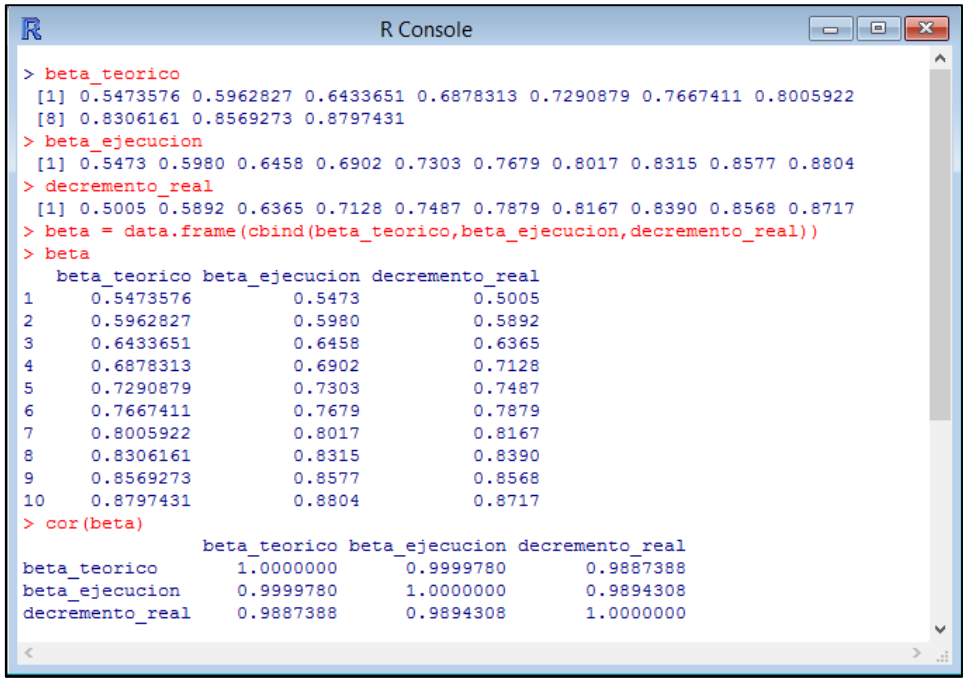


Figura N17. Correlación entre β de ejecución, β teórico y el decremento real

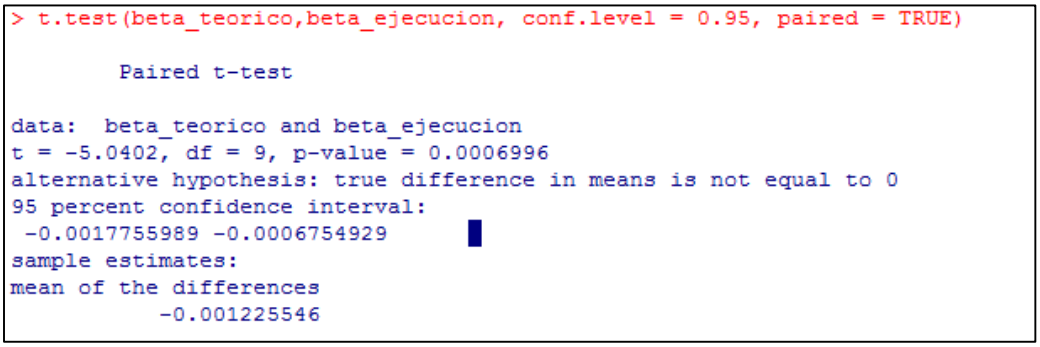


Figura N18. Test de hipótesis con R

Los resultados observados en las figuras M17 y M18, permiten aceptan las hipótesis A y B.

5. TEST DE HIPÓTESIS QUE EVALUA ESTP BAJO PERDIDAS CONSTANTES

5.1. Suposición a contrastar

Suposición A: La relación (rendimiento)/(ventana de congestión promedio) es constante.

5.2. Datos a evaluar en el test

Los datos a evaluar se muestran en la siguiente figura:

```
> # Rendimiento calculado con datos de TCP Probe
> ESTP1[ 1: 10,]$thrput_tcpprobe
 [1] 3.4524 3.3284 3.3215 3.3337 3.3533 3.3733 3.2877 3.3725 3.2922 3.4122
> ESTP2[ 1: 10,]$thrput_tcpprobe
 [1] 2.5044 2.4588 2.4481 2.4616 2.4249 2.4368 2.4398 2.4560 2.4673 2.4488
> ESTP3[ 1: 10,]$thrput_tcpprobe
 [1] 1.7034 1.6359 1.6702 1.7147 1.6734 1.7209 1.6844 1.7271 1.6732 1.7127
> ESTP4[ 1: 10,]$thrput_tcpprobe
 [1] 1.2758 1.2411 1.2431 1.2492 1.2541 1.2232 1.2203 1.2734 1.2341 1.2424
> # Rendimiento calculado con datos de TSHARK
> ESTP1[ 1: 10,]$thrput_tshark
 [1] 3.1773 3.1921 3.1914 3.2612 3.2667 3.2525 3.1825 3.1978 3.1776 3.2718
> ESTP2[ 1: 10,]$thrput_tshark
 [1] 2.3087 2.2974 2.2969 2.3079 2.2945 2.2985 2.2972 2.2983 2.3135 2.2995
> ESTP3[ 1: 10,]$thrput_tshark
 [1] 1.6789 1.5972 1.6508 1.6893 1.6528 1.6716 1.6547 1.6949 1.6250 1.6895
> ESTP4[ 1: 10,]$thrput_tshark
 [1] 1.2508 1.2118 1.2022 1.2170 1.2286 1.1922 1.1983 1.2328 1.2112 1.2008
> # Ventana de congestion promedio
> ESTP1[ 1: 10,]$cwnd
 [1] 14.9014 14.3665 14.3367 14.3893 14.4738 14.5601 14.1909 14.5568 14.2101 14.7283
> ESTP2[ 1: 10,]$cwnd
 [1] 10.8098 10.6127 10.5666 10.6250 10.4664 10.5178 10.5310 10.6010 10.6495 10.5698
> ESTP3[ 1: 10,]$cwnd
 [1] 14.7050 14.1224 14.4186 14.8024 14.4461 14.8559 14.5405 14.9094 14.4441 14.7855
> ESTP4[ 1: 10,]$cwnd
 [1] 11.0134 10.7138 10.7314 10.7838 10.8264 10.5595 10.5342 10.9930 10.6539 10.7254
```

Figura N19. Datos a ser evaluados en el test.

```
# Importacion de datos
#
# ESTP RTT 50ms perdidas constantes del 1%
ESTP1 = read.table("/home/owl-wkstn-tcp/SIMULACION/datatcp_ESTP1_perdidasconstantes.csv", sep=",")
# ESTP RTT 50ms perdidas constantes del 2%
ESTP2 = read.table("/home/owl-wkstn-tcp/SIMULACION/datatcp_ESTP2_perdidasconstantes.csv", sep=",")
# ESTP RTT 100ms perdidas constantes del 1%
ESTP3 = read.table("/home/owl-wkstn-tcp/SIMULACION/datatcp_ESTP3_perdidasconstantes.csv", sep=",")
# ESTP RTT 100ms perdidas constantes del 2%
ESTP4 = read.table("/home/owl-wkstn-tcp/SIMULACION/datatcp_ESTP4_perdidasconstantes.csv", sep=",")
#
colnames(ESTP1)= c("N","thrput_tcpprobe","thrput_tshark","cwnd","beta_ejecucion","decremento_real")
colnames(ESTP2)= c("N","thrput_tcpprobe","thrput_tshark","cwnd","beta_ejecucion","decremento_real")
colnames(ESTP3)= c("N","thrput_tcpprobe","thrput_tshark","cwnd","beta_ejecucion","decremento_real")
colnames(ESTP4)= c("N","thrput_tcpprobe","thrput_tshark","cwnd","beta_ejecucion","decremento_real")
# Valores a ingresar al test
# Rendimiento calculado con datos de TCP Probe
ESTP1[ 1: 10,]$thrput_tcpprobe
ESTP2[ 1: 10,]$thrput_tcpprobe
ESTP3[ 1: 10,]$thrput_tcpprobe
ESTP4[ 1: 10,]$thrput_tcpprobe
# Rendimiento calculado con datos de TSHARK
ESTP1[ 1: 10,]$thrput_tshark
ESTP2[ 1: 10,]$thrput_tshark
ESTP3[ 1: 10,]$thrput_tshark
ESTP4[ 1: 10,]$thrput_tshark
# Ventana de congestion promedio
ESTP1[ 1: 10,]$cwnd
ESTP2[ 1: 10,]$cwnd
ESTP3[ 1: 10,]$cwnd
ESTP4[ 1: 10,]$cwnd
# Test de proporciones entre valores de rendimiento calculado por TcpProbe/cwnd en cada muestra
prop.test(ESTP1[ 1: 10,]$thrput_tcpprobe,ESTP1[ 1: 10,]$cwnd, conf.level = 0.95)
prop.test(ESTP2[ 1: 10,]$thrput_tcpprobe,ESTP2[ 1: 10,]$cwnd, conf.level = 0.95)
prop.test(ESTP3[ 1: 10,]$thrput_tcpprobe,ESTP3[ 1: 10,]$cwnd, conf.level = 0.95)
prop.test(ESTP4[ 1: 10,]$thrput_tcpprobe,ESTP4[ 1: 10,]$cwnd, conf.level = 0.95)
# Test de proporciones entre valores de rendimiento calculado por Tshark/cwnd en cada muestra
prop.test(ESTP1[ 1: 10,]$thrput_tshark,ESTP1[ 1: 10,]$cwnd, conf.level = 0.95)
prop.test(ESTP2[ 1: 10,]$thrput_tshark,ESTP2[ 1: 10,]$cwnd, conf.level = 0.95)
prop.test(ESTP3[ 1: 10,]$thrput_tshark,ESTP3[ 1: 10,]$cwnd, conf.level = 0.95)
prop.test(ESTP4[ 1: 10,]$thrput_tshark,ESTP4[ 1: 10,]$cwnd, conf.level = 0.95)
```

Figura N20. Script ~/ EVALUACION_R/test_ESTP_constante.R

5.3. Script en R.

El script en R que evalúa los datos del archivo .csv anterior es:

~/ EVALUACION_R/test_ESTP_constante.R (Ver Figura M10).

5.4. Resultados en R.

Los resultados observados en la consola de R se muestran en las siguientes imágenes:

```
> # Test de proporciones entre valores de rendimiento calculado por TcpProbe/cwnd en cada muestra
> prop.test(ESTP1[ 1: 10,]$thrput_tcpprobe,ESTP1[ 1: 10,]$cwnd, conf.level = 0.95)

        10-sample test for equality of proportions without continuity correction

data:  ESTP1[1:10, ]$thrput_tcpprobe out of ESTP1[1:10, ]$cwnd
X-squared = 3.1508e-09, df = 9, p-value = 1
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4    prop 5    prop 6    prop 7    prop 8    prop 9   prop 10
0.2316829 0.2316779 0.2316781 0.2316791 0.2316807 0.2316811 0.2316766 0.2316787 0.2316803 0.2316764

Warning message:
In prop.test(ESTP1[1:10, ]$thrput_tcpprobe, ESTP1[1:10, ]$cwnd,  :
  Chi-squared approximation may be incorrect
> prop.test(ESTP2[ 1: 10,]$thrput_tcpprobe,ESTP2[ 1: 10,]$cwnd, conf.level = 0.95)

        10-sample test for equality of proportions without continuity correction

data:  ESTP2[1:10, ]$thrput_tcpprobe out of ESTP2[1:10, ]$cwnd
X-squared = 4.619e-09, df = 9, p-value = 1
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4    prop 5    prop 6    prop 7    prop 8    prop 9   prop 10
0.2316787 0.2316847 0.2316828 0.2316800 0.2316842 0.2316834 0.2316779 0.2316763 0.2316822 0.2316789

Warning message:
In prop.test(ESTP2[1:10, ]$thrput_tcpprobe, ESTP2[1:10, ]$cwnd,  :
  Chi-squared approximation may be incorrect
> prop.test(ESTP3[ 1: 10,]$thrput_tcpprobe,ESTP3[ 1: 10,]$cwnd, conf.level = 0.95)

        10-sample test for equality of proportions without continuity correction

data:  ESTP3[1:10, ]$thrput_tcpprobe out of ESTP3[1:10, ]$cwnd
X-squared = 3.8678e-09, df = 9, p-value = 1
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4    prop 5    prop 6    prop 7    prop 8    prop 9   prop 10
0.1158382 0.1158373 0.1158365 0.1158393 0.1158375 0.1158395 0.1158420 0.1158397 0.1158397 0.1158365

Warning message:
In prop.test(ESTP3[1:10, ]$thrput_tcpprobe, ESTP3[1:10, ]$cwnd,  :
  Chi-squared approximation may be incorrect
> prop.test(ESTP4[ 1: 10,]$thrput_tcpprobe,ESTP4[ 1: 10,]$cwnd, conf.level = 0.95)

        10-sample test for equality of proportions without continuity correction

data:  ESTP4[1:10, ]$thrput_tcpprobe out of ESTP4[1:10, ]$cwnd
X-squared = 4.2158e-09, df = 9, p-value = 1
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4    prop 5    prop 6    prop 7    prop 8    prop 9   prop 10
0.1158407 0.1158413 0.1158376 0.1158404 0.1158372 0.1158388 0.1158417 0.1158374 0.1158355 0.1158372

Warning message:
In prop.test(ESTP4[1:10, ]$thrput_tcpprobe, ESTP4[1:10, ]$cwnd,  :
  Chi-squared approximation may be incorrect
```

Figura N21. Resultados de test de hipótesis (Parte 1)

```
> # Test de proporciones entre valores de rendimiento calculado por Tshark/cwnd en cada muestra
> prop.test(ESTP1[ 1: 10,]$thrput_tshark,ESTP1[ 1: 10,]$cwnd, conf.level = 0.95)

        10-sample test for equality of proportions without continuity correction

data:  ESTP1[1:10, ]$thrput_tshark out of ESTP1[1:10, ]$cwnd
X-squared = 0.010786, df = 9, p-value = 1
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4    prop 5    prop 6    prop 7    prop 8    prop 9   prop 10
0.2132216 0.2221905 0.2226035 0.2266406 0.2256975 0.2233845 0.2242634 0.2196774 0.2236156 0.2221438

Warning message:
In prop.test(ESTP1[1:10, ]$thrput_tshark, ESTP1[1:10, ]$cwnd, conf.level = 0.95) :
  Chi-squared approximation may be incorrect
> prop.test(ESTP2[ 1: 10,]$thrput_tshark,ESTP2[ 1: 10,]$cwnd, conf.level = 0.95)

        10-sample test for equality of proportions without continuity correction

data:  ESTP2[1:10, ]$thrput_tshark out of ESTP2[1:10, ]$cwnd
X-squared = 0.001305, df = 9, p-value = 1
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4    prop 5    prop 6    prop 7    prop 8    prop 9   prop 10
0.2135747 0.2164765 0.2173736 0.2172141 0.2192253 0.2185343 0.2181369 0.2168003 0.2172402 0.2175538

Warning message:
In prop.test(ESTP2[1:10, ]$thrput_tshark, ESTP2[1:10, ]$cwnd, conf.level = 0.95) :
  Chi-squared approximation may be incorrect
> prop.test(ESTP3[ 1: 10,]$thrput_tshark,ESTP3[ 1: 10,]$cwnd, conf.level = 0.95)

        10-sample test for equality of proportions without continuity correction

data:  ESTP3[1:10, ]$thrput_tshark out of ESTP3[1:10, ]$cwnd
X-squared = 0.00073092, df = 9, p-value = 1
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4    prop 5    prop 6    prop 7    prop 8    prop 9   prop 10
0.1141721 0.1130969 0.1144910 0.1141234 0.1144115 0.1125210 0.1137994 0.1136800 0.1125027 0.1142674

Warning message:
In prop.test(ESTP3[1:10, ]$thrput_tshark, ESTP3[1:10, ]$cwnd, conf.level = 0.95) :
  Chi-squared approximation may be incorrect
> prop.test(ESTP4[ 1: 10,]$thrput_tshark,ESTP4[ 1: 10,]$cwnd, conf.level = 0.95)

        10-sample test for equality of proportions without continuity correction

data:  ESTP4[1:10, ]$thrput_tshark out of ESTP4[1:10, ]$cwnd
X-squared = 0.00047007, df = 9, p-value = 1
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4    prop 5    prop 6    prop 7    prop 8    prop 9   prop 10
0.1135707 0.1131065 0.1120264 0.1128545 0.1134819 0.1129031 0.1137533 0.1121441 0.1136861 0.1119585

Warning message:
In prop.test(ESTP4[1:10, ]$thrput_tshark, ESTP4[1:10, ]$cwnd, conf.level = 0.95) :
  Chi-squared approximation may be incorrect
```

Figura M22. Resultados de test de hipótesis (Parte 2)

6. TEST DE HIPÓTESIS QUE COMPARA ESTP CON GAIMD Y TCP RENO

Las hipótesis a contrastar son:

Hipótesis A: El rendimiento promedio es mayor cuando se emplea ESTP en comparación a TCP RENO y GAIMD

6.1. Datos a evaluar

Los datos de los promedios de rendimiento y ventana de congestión de cada muestra son colocados en el archivo ~/ EVALUACION_R/ESTP_vs_GAIMD_vs_RENO.csv.

	A	B	C	D	E	F	G	H	I
1	RENO	GAIMD	ESTP	RENO	GAIMD	ESTP	RENO	GAIMD	ESTP
2	2.47	3.254	3.425	2.53	3.315	3.526	2.353	3.144	3.526
3	1.77	2.353	2.499	1.844	2.396	2.576	1.726	2.285	2.439
4	1.31	1.663	1.773	1.301	1.693	1.777	1.301	1.649	1.724
5	0.926	1.21	1.334	0.929	1.2	1.322	0.886	1.155	1.281
6									

Figura N23. Archivo ~/ EVALUACION_R/ ESTP_vs_GAIMD_vs_RENO.csv.

En la figura N23:

Columna A : Rendimiento con TCP RENO obtenido desde datos de captura de iPerf.

Columna B : Rendimiento con GAIMD obtenido desde datos de captura de iPerf.

Columna C : Rendimiento con ESTP obtenido desde datos de captura de iPerf.

Columna D : Rendimiento con TCP RENO calculado con datos de captura de TCP Probe

Columna E : Rendimiento con GAIMD calculado con datos de captura de TCP Probe

Columna F : Rendimiento con ESTP calculado con datos de captura de TCP Probe

Columna G : Rendimiento con TCP RENO calculado desde datos de captura de tshark.

Columna H : Rendimiento con GAIMD calculado desde datos de captura de tshark.

Columna I : Rendimiento con ESTP calculado con datos de captura de TCP Probe.

6.2. Script en R para realizar test de hipótesis

El script en R que evalúa los datos del archivo .csv anterior es:

~/ EVALUACION_R/Comparacion_ESTP_GAIMD_RENO.R.

```
RENO_GAIMD =read.table("home/owl-wkstn-
tcp/EVALUACION_R/ESTP_vs_GAIMD_vs_RENO.csv", sep=",", header = TRUE)
#
Muestra = c(1,4)
R_iperf_reno = RENO_GAIMD$RENO
R_iperf_gaimd = RENO_GAIMD$GAIMD
R_iperf_estp = RENO_GAIMD$ESTP
R_probe_reno = RENO_GAIMD$RENO.1
R_probe_gaimd = RENO_GAIMD$GAIMD.1
R_probe_estp = RENO_GAIMD$ESTP.1
R_tshark_reno = RENO_GAIMD$RENO.2
R_tshark_gaimd = RENO_GAIMD$GAIMD.2
R_tshark_estp = RENO_GAIMD$ESTP.2
R_iperf_reno # rendimiento promedio obtenido por iPerf empleando TCP RENO
R_iperf_gaimd # rendimiento promedio obtenido por iPerf empleando GAIMD
R_iperf_estp # rendimiento promedio obtenido por iPerf empleando ESTP
R_probe_reno # rendimiento promedio obtenido de los datos capturados por TCP Probe: TCP RENO
R_probe_gaimd # rendimiento promedio obtenido de los datos capturados por TCP Probe: GAIMD
R_probe_estp # rendimiento promedio obtenido de los datos capturados por TCP Probe: ESTP
R_tshark_reno # rendimiento promedio obtenido de los datos capturados por TSHARK: TCP RENO
R_tshark_gaimd # rendimiento promedio obtenido de los datos capturados por TSHARK: GAIMD
R_tshark_estp # rendimiento promedio obtenido de los datos capturados por TSHARK: ESTP
#
# Test de hipotesis para comprobar si rendimiento de ESTP es mejor que rendimiento de RENO
#
# 1) Empleando datos de iperf
t.test (R_iperf_estp, R_iperf_reno, conf.level = 0.95, paired = TRUE)
cor(R_iperf_estp, R_iperf_reno)
# 2) Empleando datos de tcpprobe
t.test (R_probe_estp, R_iperf_reno, conf.level = 0.95, paired = TRUE)
cor(R_probe_estp, R_probe_reno)
# 3) Empleando datos de tshark
t.test (R_tshark_estp, R_iperf_reno, conf.level = 0.95, paired = TRUE)
cor(R_tshark_estp, R_tshark_reno)
#
# Test de hipotesis para comprobar si rendimiento de ESTP es mejor que rendimiento de GAIMD
#
# 1) Empleando datos de iperf
t.test (R_iperf_estp, R_iperf_gaimd, conf.level = 0.95, paired = TRUE)
cor(R_iperf_estp, R_iperf_gaimd)
# 2) Empleando datos de tcpprobe
t.test (R_probe_estp, R_iperf_gaimd, conf.level = 0.95, paired = TRUE)
cor(R_probe_estp, R_probe_gaimd)
# 3) Empleando datos de tshark
t.test (R_tshark_estp, R_iperf_gaimd, conf.level = 0.95, paired = TRUE)
cor(R_tshark_estp, R_tshark_gaimd)
#
```

Figura N24. Archivo script: ~/EVALUACION_R/Comparacion_ESTP_GAIMD_RENO.R

6.3. Resultados en R

Los resultados observados en la consola de R se muestran en las siguientes imágenes:

```
> # Test de hipotesis para comprobar si rendimiento de ESTP es mejor que rendimiento de
RENO
> #
> # 1) Empleando datos de iperf
> t.test (R_iperf_estp, R_iperf_reno, conf.level = 0.95, paired = TRUE)

        Paired t-test

data:  R_iperf_estp and R_iperf_reno
t = 5.0459, df = 3, p-value = 0.01501
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.2358921 1.0416079
sample estimates:
mean of the differences
                0.63875

> cor(R_iperf_estp, R_iperf_reno)
[1] 0.9988891
> # 2) Empleando datos de tcpprobe
> t.test (R_probe_estp, R_iperf_reno, conf.level = 0.95, paired = TRUE)

        Paired t-test

data:  R_probe_estp and R_iperf_reno
t = 4.434, df = 3, p-value = 0.02132
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.1922923 1.1702077
sample estimates:
mean of the differences
                0.68125

> cor(R_probe_estp, R_probe_reno)
[1] 0.9996327
> # 3) Empleando datos de tshark
> t.test (R_tshark_estp, R_iperf_reno, conf.level = 0.95, paired = TRUE)

        Paired t-test

data:  R_tshark_estp and R_iperf_reno
t = 3.9102, df = 3, p-value = 0.02972
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.1160445 1.1309555
sample estimates:
mean of the differences
                 0.6235

> cor(R_tshark_estp, R_tshark_reno)
[1] 0.9953318
```

Figura N25. Resultados de test de Student y correlación que contrasta si el rendimiento al emplear ESTP es mejor que si se emplea TCP Reno.

```
> # Test de hipotesis para comprobar si rendimiento de ESTP es mejor que rendimiento
de GAIMD
> #
> # 1) Empleando datos de iperf
> t.test (R_iperf_estp, R_iperf_gaimd, conf.level = 0.95, paired = TRUE)

        Paired t-test

data:  R_iperf_estp and R_iperf_gaimd
t = 10.333, df = 3, p-value = 0.001934
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.09532318 0.18017682
sample estimates:
mean of the differences
                0.13775

> cor(R_iperf_estp, R_iperf_gaimd)
[1] 0.9999272
> # 2) Empleando datos de tcpprobe
> t.test (R_probe_estp, R_iperf_gaimd, conf.level = 0.95, paired = TRUE)

        Paired t-test

data:  R_probe_estp and R_iperf_gaimd
t = 4.4954, df = 3, p-value = 0.02055
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.05264505 0.30785495
sample estimates:
mean of the differences
                0.18025

> cor(R_probe_estp, R_probe_gaimd)
[1] 0.9995496
> # 3) Empleando datos de tshark
> t.test (R_tshark_estp, R_iperf_gaimd, conf.level = 0.95, paired = TRUE)

        Paired t-test

data:  R_tshark_estp and R_iperf_gaimd
t = 2.4452, df = 3, p-value = 0.09207
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.0369323  0.2819323
sample estimates:
mean of the differences
                 0.1225

> cor(R_tshark_estp, R_tshark_gaimd)
[1] 0.9976917
```

Figura N26. Resultados de test de student y correlación que contrasta si el rendimiento al emplear ESTP es mejor que si se emplea GAIMD.

Según las figuras N25 y N26:

- Existe una relación lineal entre ESTP y TCP RENO, así como entre ESTP y GAIMD.
- ESTP tiene mayor rendimiento que TCP Reno y un rendimiento mayor o igual a GAIMD.