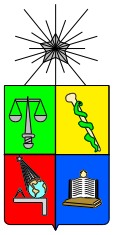


UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

IMPLEMENTACION DE ETHERNET-SERVICES TRANSPORT PROTOCOL EN LINUX: UN METODO DE CONTROL DE CONGESTION DE TRAFICO ORIENTADO A REDES CARRIER-ETHERNET

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRICISTA

SERGIO FERNANDO ANGULO CÁCERES

PROFESOR GUÍA:
CLAUDIO ESTÉVEZ MONTERO

MIEMBROS DE LA COMISIÓN
ALBERTO CASTRO ROJAS
RODRIGO ARENAS ANDRADE

SANTIAGO DE CHILE
2013

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE:
INGENIERO CIVIL ELECTRICISTA
POR: SERGIO ANGULO CACERES.
FECHA: 01/10/2013
PROF. GUÍA: DR. CLAUDIO ESTÉVEZ MONTERO

IMPLEMENTACION DE ETHERNET-SERVICES TRANSPORT PROTOCOL EN LINUX: UN METODO DE CONTROL DE CONGESTION DE TRAFICO ORIENTADO A REDES CARRIER-ETHERNET

Uno de los protocolos de la capa de transporte más importantes, Transport Control Protocol (TCP), garantiza que los paquetes sean recibidos en forma ordenada e integra y aborda el problema de la congestión evitando transmitir paquetes nuevos si el receptor no ha notificado que los paquetes enviados con anterioridad han sido recibidos. Se observa en su comportamiento, por una parte, que la velocidad de transmisión de datos es afectada negativamente respecto a la distancia entre el transmisor y receptor. Por otra parte para evitar la congestión el protocolo solo detecta la existencia de tráfico, sin la capacidad de estimar su magnitud. Con el fin de abarcar las debilidades de TCP, en relación al efecto provocado por la distancia entre transmisor y receptor, la eficiencia de la transmisión y el problema de la congestión de tráfico, el Dr. Claudio Estévez ha desarrollado la teoría de un protocolo de transporte orientado a redes Carrier Ethernet, que mejora la eficiencia de la capa de transporte, llamado ESTP (Ethernet-Services Transport Protocol).

Para llevar a la realidad la propuesta recién descrita, surge el problema de investigación de esta memoria: "¿La implementación de ESTP valida el modelo teórico presentando un mejor comportamiento que los métodos de congestión tradicionales?". Así, el propósito de esta memoria es concretar la codificación de un modulo de congestión en Linux para implementar ESTP y posteriormente compararlo con la teoría planteada por el Dr. Claudio Estévez y con otros protocolos existentes, particularmente BIC, CUBIC y RENO.

Los principales resultados de este trabajo caracterizan el comportamiento experimental de ESTP comparándolo con la teoría en que se basa la implementación, en cuanto al comportamiento en el tiempo y a variaciones de Round Trip Time (RTT), poniendo énfasis en la reducción de la ventana de congestión, el cumplimiento de las condiciones de Carrier-Ethernet y la reducción del cuello de botella impuesto por la capa de transporte para valores altos de RTT. Además los resultados experimentales se contrastan con el funcionamiento general de otros métodos de congestión bajo un ambiente de pruebas controlado, demostrando la superioridad de ESTP en la medida que los valores de RTT aumentan. Así, se observa que a un valor de RTT de 70[ms], en un ambiente de pérdida variable, el flujo de datos de ESTP es superior a BIC en un 35,7% , a CUBIC en un 41,1% y a RENO en un 71,8%.

Cabe señalar que a raíz de esta investigación se publicaron tres papers titulados: "A Carrier-Ethernet Oriented Transport Protocol with a Novel Congestion Control and QoS Integration: Analytical, Simulated and Experimental Validation"(ICC 2012), "Ethernet-Services Transport Protocol Design oriented to Carrier Ethernet Networks" (GLOBECOM 2012) y "Ethernet-Services Transport Protocol forCarrier Ethernet Networks"(ICCCN 2012). Además de un journal enviado, denominado: "QoS-supporting Transport Protocol with Congestion Intensity Estimation".

“*A mis Padres y a Isabel por su infinita paciencia y amor incondicional*”

Agradecimientos

Al laboratorio *Optical & Wireless Lab* (OWL) del Departamento de Ingenieria Electrica de la Universidad de Chile, por facilitar espacio, equipos y el factor humano, de quienes me llevo gratos momentos y recuerdos.

A mi querido profesor Guía, el Doctor Claudio Estévez por su tiempo, dedicación, confianza y apoyo durante todo mi proceso de memoria y mi estadía en OWL. Por sobre todo agradezco el haberme enseñado a conocer, comprender y valorar el mundo de la investigación y las telecomunicaciones.

A la comisión correctora de esta memoria, profesores: Claudio Estevez, Alberto Castro y Rodrigo Arenas. Por su buena disposición a la revisión de mi memoria y participación en la defensa de la misma.

A mis amados padres, Jaime e Isolina, por el cariño, comprensión, contensión, apoyo y amor incondicional que me han entregado durante toda mi vida. A mi padre quien ha sido mi gran modelo de ingeniero y a mi madre por creer en mi hasta en los momentos más adversos. En definitiva por se los pilares fundamentales de mi vida.

A mis queridos hermanos Jaime, Gabriel y Sebastian y mis sobrinos, Paulina, Philippe, Renata, Gaspar, Rebeca, Joaquin y Javiera, quienes me acompañaron a lo largo de mi carrera y mi vida, les agradezco sinceramente su apoyo y alegría.

A mi amada Isabel, quien me ha acompañado, apoyado y levantado en cada uno de los momentos, con palabras, paciencia, consejos y sonrisas. Muchas gracias por todo el amor que me has entregado y por decidir acompañarme.

A Jorge e Isabel Urrutia Avendaño, Juanita Naranjo, Familia Egaña Urrutia y Urrutia Román, por hacerme parte de su familia y por todo su cariño y comprensión.

A Zweicom, donde me desempeño laboralmente, en especial a Carlos Fangmeier, por brindarme los tiempos para concretar este proceso.

A todos aquellos compañeros, familiares y amigos con los que compartí más que un cigarrito durante mi largo periodo universitario.

A los docentes del Departamento de Ingeniería Eléctrica y a mi querida Facultad de Injeniería, que con motivación y sangre me formaron como profesional.

Y por sobre todo gracias a Dios por poder compartir mi vida con todos ellos.

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

CAPÍTULO 1: INTRODUCCIÓN

1.1 Fundamentación del problema

Desde la creación de las comunicaciones a través de paquetes de datos, se han establecido distintos protocolos que aseguran la integridad de la información transmitida y abordan el problema del control de congestión, con el fin de mejorar la velocidad de transferencia de datos en redes de computadores.

TCP (Transport Control Protocol), es un protocolo de transporte de datos orientado a la conexión, que garantiza que los datagramas sean recibidos en forma ordenada e integra y aborda el problema de la congestión no transmitiendo paquetes nuevos si es que el receptor no ha notificado que los datagramas enviados con anterioridad han sido recibidos. Entre las limitaciones de TCP se observa que la velocidad de transmisión de datos es afectada negativamente con respecto a la distancia entre el transmisor y receptor y además para evitar la congestión el protocolo solo detecta la existencia de tráfico sin la capacidad de estimar su magnitud.

Con la evolución de la tecnología y los servicios de comunicación, se requieren mayores velocidades de transmisión donde el protocolo de transporte de datos toma relevancia. Los cuellos de botella que TCP provoca debido a la distancia entre receptor y transmisor, no permiten obtener altas velocidades de transmisión. En este sentido el Dr. Claudio Estevez ha desarrollado la teoría de un protocolo de transporte orientado a redes Carrier Ethernet, que mejora la eficiencia de la capa de transporte, llamado ESTP (Ethernet-Services Transport Protocol). ESTP reduce el efecto provocado por la distancia entre transmisor y receptor, incorpora información conocida de la red mejorando la eficiencia de la transmisión y aborda el problema de la congestión de tráfico midiendo el nivel de congestión y no solo la existencia de esta.

Este nuevo método de control de congestión de tráfico difiere de los métodos tradicionales, por lo que la comparación es necesaria para la valoración del protocolo desarrollado.

La importancia de la implementación de Ethernet-Services Transport Protocol radica en la validación experimental de lo antes desarrollado teóricamente, logrando que los conceptos asociados a él se sitúen en la realidad.

En relación a lo antes expuesto, se estructura el siguiente problema de investigación:

¿La implementación de ESTP valida el modelo teórico presentando un mejor comportamiento que los métodos de congestión tradicionales?

Para evaluar la importancia potencial de la investigación, se utilizarán los criterios propuestos por Hernández Fernández y Baptista (2006), en su libro Metodología de la Investigación, tal como se detalla a continuación:[5]

Conveniencia: Esta investigación lleva a la realidad un nuevo protocolo de transporte que pretende aumentar el nivel de eficiencia en la transmisión de datos, a través de un mejor uso del ancho de banda de las redes actuales que por su naturaleza física es limitado.

Relevancia Social: El nuevo protocolo se orienta a redes Carrier-Ethernet, que en la actualidad se están implementando y son parte de una nueva generación de redes de alta velocidad. La validación de ESTP, puede llevar a la implementación en cada servidor que soporte Carrier-Ethernet, en este sentido los usuarios de éste servicio serán los principales beneficiados. Además si se observa que en la realidad, ESTP presenta mejores prestaciones que los métodos de control de congestión tradicionales, podría implementarse como estándar en servidores que soporten el protocolo TCP/IP, puesto que para la implementación de ESTP no es necesario hacer ningún cambio en los nodos de la red, solamente basta la implementación en los servidores, de este modo los beneficiados con la validación del protocolo serían todos aquellos usuarios que ocupen el esquema cliente servidor en sesiones orientadas a la conexión donde es necesaria la transmisión integra de los datos o dicho de otro modo, todos los usuarios que ocupen el protocolo TCP.

Implicaciones Prácticas: La investigación apunta a validar un protocolo que supone el uso eficiente del ancho de banda disponible, por lo que se pueden obtener mayores velocidades de transmisión de datos, logrando la posible inclusión de ESTP como protocolo estándar. Además usando este método de control de tráfico puede mejorar la calidad de servicio de prestaciones que usualmente son no orientadas a la conexión, como por ejemplo servicios de streaming.

Valor teórico: La implementación de ESTP, valida la teoría en la que se basa. El método de control de congestión propuesto puede ser generalizado y aplicado en otros escenarios de transmisión, donde exista congestión. En particular, se analizará el efecto de la pérdida y el retraso de paquetes de datos asociados a la velocidad de transmisión en diferentes escenarios tanto para el protocolo ESTP como para los protocolos de transporte más tradicionales. Como consecuencia se estudiará el efecto conjunto de estas variables en la transmisión de datos, generando antecedentes que pueden servir para el análisis del rendimiento de los actuales protocolos en uso o para investigaciones futuras relacionadas con protocolos asociados a la capa de transporte. De la comparación de ESTP con otros métodos de congestión, pueden surgir nuevas ideas que lleven a un comportamiento conjunto o por fases entre protocolos, generando un mejor protocolo de transmisión de datos.

Utilidad metodológica: En este estudio se crearán rutinas, que automaticen la toma de datos, de manera que todas las pruebas tengan la misma validez estadística. Además sugiere una manera de estudiar una red de datos basada en parámetros característicos como la pérdida y el retraso.

1.2 Objetivos del Trabajo de Memoria

1.2.1 Objetivos Generales

De lo expuesto con anterioridad surgen los siguientes objetivos generales de investigación:

1. Contrastar los datos experimentales con los resultados validados por teoría.
2. Comparar el comportamiento de ESTP con los protocolos BIC, CUBIC y RENO en relación con el aumento de Round Trip Time.

1.2.2 Objetivos Específicos

Para la consecución de los objetivos generales antes expuestos, se tienen los siguientes objetivos específicos:

1. Analizar la metodología de implementación de un módulo de congestión de tráfico en el kernel de Linux.
2. Codificar en lenguaje de programación en C el módulo ESTP en Linux para su implementación.
3. Analizar los resultados obtenidos por ESTP y métodos de evitación de tráfico tradicionales para elaborar su comparación.

1.3 Hipótesis

Las hipótesis de este trabajo son las que se presentan a continuación:

"Los datos experimentales de ESTP validan el modelo teórico del protocolo".

"Al aumentar el Round Trip Time, ESTP tiene un mejor comportamiento que BIC, CUBIC y RENO"

Según Hernández Fernández y Baptista (2006), el enfoque de la investigación es de tipo cuantitativo y de diseño experimental, especificamente experimento puro. Lo anterior se definió ante las siguientes descripciones:

- *Cuantitativo*: Usa la recolección de datos para probar hipótesis, con base en la medición numérica y el análisis estadístico, para establecer patrones de comportamiento y probar teorías.

- *Experimental*: se manipulan intencionalmente una o más variables independientes en experimentos y se miden las variables dependientes.
- *Experimento Puro*: Se tienen grupos de comparación y equivalencia entre ellos, controlando las variables en estudio e influyendo en varias variables.

1.4 Estructura de la Memoria

El trabajo de memoria desarrollado, está constituido por cinco capítulos. A continuación se detalla el contenido de cada uno de ellos.

Capítulo 1: Introducción, permite conocer la fundamentación del problema, objetivos e hipótesis del trabajo. En consecuencia, explica el qué, cómo y para qué del trabajo realizado.

Capítulo 2: Revisión Bibliográfica, permite conocer las referencias teóricas que sustentan el trabajo realizado, explicita cada uno de los conceptos relacionados con el desarrollo de esta memoria.

Capítulo 3: Implementación, explica el proceso de implementación experimental del trabajo realizado, donde se explica la codificación del protocolo ESTP, las pruebas realizadas explicitando el diseño del ambiente experimental.

Capítulo 4: Análisis de resultados, se explicitan en forma gráfica los resultados obtenidos de las pruebas experimentales. Además se triangulan y analizan los datos obtenidos en función de las hipótesis que sustentan el trabajo realizado.

Capítulo 5: Conclusiones, presenta las conclusiones obtenidas en función de la implementación, las pruebas realizadas y el análisis de las mismas en relación con las hipótesis de investigación planteadas inicialmente.

A continuación de los cinco capítulos recién descritos se muestran las referencias bibliográficas utilizadas y los anexos del trabajo realizado.

CAPÍTULO 2: REVISIÓN BIBLIOGRÁFICA

2.1 Marco Teórico

La primera transmisión de paquetes de datos fue realizada el día 29 de Octubre de 1969 a las 22:30 horas, bajo la supervisión del Doctor Leonard Kleinrock [15], quien desarrolló anteriormente la teoría de conmutación de paquetes. Esta transmisión fue realizada como parte del proyecto ARPANET, que es el predecesor del actual Internet[11].

"Internet es una red de computadoras que interconecta cientos de millones de dispositivos informáticos a lo largo de todo el mundo" (Kurose, 2010) [13]. Según Internet Society [11], tiene la capacidad de transmitir a todo el mundo, siendo un mecanismo de diseminación de la información y un medio de colaboración e interacción entre individuos sin importar su ubicación geográfica, convirtiéndose en uno de los ejemplos más exitosos de los beneficios de la inversión sostenida, el compromiso con la investigación y el desarrollo de la infraestructura de la información.

Esta gran red interconectada se compone físicamente de equipos terminales, dispositivos de conmutación de paquetes y distintos tipos de medios para los enlaces entre sus elementos. Estos ejecutan protocolos que controlan el envío y la recepción de la información dentro de Internet, lo que permite proporcionar servicios a aplicaciones distribuidas ejecutadas en los terminales de red.

2.1.1 Protocolo de Red

Según Kurose (2010) [13], un protocolo se caracteriza por:

*"Un **protocolo** define el formato y el orden de los mensajes intercambiados entre dos o más entidades que se comunican, así como las acciones tomadas en la transmisión y/o la recepción de un mensaje u otro suceso."*

Luego, un protocolo de red se entiende como la forma en que deben comunicarse los distintos elementos de la red para que estos puedan intercambiar información.

2.1.2 Arquitectura de Internet

La arquitectura de Internet puede entenderse en base a un modelo de capas, donde el hardware y/o software de cada capa implementa un protocolo adecuado para interactuar entre elementos de la misma capa, prestando servicio a las capas inmediatamente superiores y siendo servidas por las capas inferiores. La visión de un sistema por capas, presenta una forma estructurada y modular, para el análisis y modificación de los componentes de un sistema.

Una pila de protocolos, es el conjunto de protocolos de las distintas capas definidas en un sistema. Durante los años 70, el organismo internacional International Organization for Standardization (ISO), propuso un modelo de 7 capas para las redes de computadores, denominado modelo OSI (Open System Interconection). Posteriormente se desprende un modelo de 5 capas, que corresponde a la pila de protocolos de Internet.

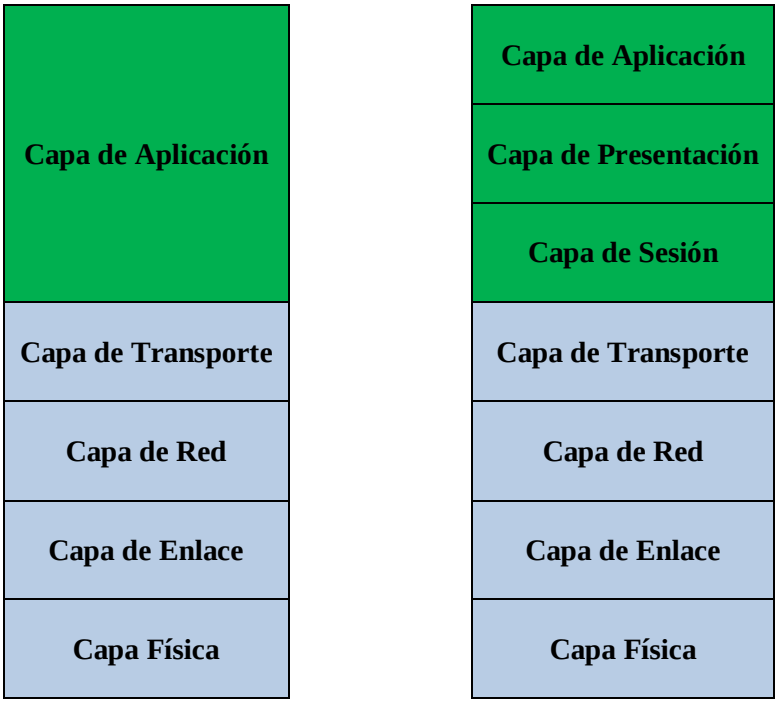


Figura 1. Protocolos de Internet y protocolos del modelo OSI.

En la pila de protocolos de internet, las tres capas más altas del modelo OSI han sido unidas en una sola. La razón de la eliminación de estas capas es que en la arquitectura de internet, pueden omitirse y es problema de los desarrolladores de software si son implementadas o no como parte de la aplicación.

2.1.3 Modelo de capas de Internet

Para entender la arquitectura de internet, se explicarán las distintas capas del modelo que la definen.

Capa de Aplicación: En esta residen las aplicaciones de red y sus protocolos, como por ejemplo HTTP (Hipertext Transfer Protocol), SMTP (Simple Mail Transfer Protocol), FTP (File Tranfer Protocol), entre otros.

Capa de Transporte: Se ocupa de transportar los mensajes de la capa de aplicación entre puntos terminales de la aplicación, es decir de extremo a extremo. Los protocolos de la capa de transporte son dos, UDP (User Datagram Protocol) y TCP (Transmission Control Protocol). UDP está enfocado a servicios no orientados a la conexión, por lo que no asegura que la información llegue en forma correcta, no ofrece control de flujo, ni control de congestión. Por su parte TCP ofrece un servicio orientado a la conexión, presentando todas las ventajas que no posee UDP.

Capa de Red: Es responsable de transportar los paquetes de la capa de red o datagramas, entre equipos terminales. El protocolo más importante de esta capa es el IP (Internet Protocol), de manera que todos los elementos que tienen capa de red deben ejecutar este protocolo.

Capa de Enlace: Se encarga de transportar un paquete desde un nodo (host o router) al siguiente en la ruta que une los nodos transmisor y receptor. Entre los protocolos de esta capa se encuentra Ethernet, WIFI , PPP(Point to Point Protocol) , entre otros.

Capa Física: Se encarga de transmitir los bits individuales entre dos nodos conectados físicamente a través de algún medio de transmisión, existiendo protocolos específicos definidos para cada medio.

Este modelo permite que cada capa se comporte en forma independiente, de tal manera que cualquier cambio puede ser realizado en capas específicas sin que las capas inferiores o superiores tengan que realizar un cambio en particular.

En la transmisión, cada capa encapsula la información proveniente de la inmediatamente superior y luego la entrega a la inmediatamente inferior. El proceso inverso ocurre en la recepción donde la información de cada capa es desencapsulada y entregada a la capa inmediatamente superior. Esta encapsulamiento consiste en agregar una cabecera a la unidad de datos con información relevante donde generalmente se indica cual es el tamaño total del encapsulamiento específico, este conjunto de información propia de cada protocolo es llamado PDU (Protocol Data Unit). En este sentido suele llamarse Payload, a la información procedente de la capa de aplicación; Segmento a los datos manejados por la capa de transporte; Paquete a la información conducida por la capa de Red; Marcos o Frames a la información de la capa de enlace de datos y finalmente Tramas a la información que maneja la capa física.

2.1.4 Transport Control Protocol

El protocolo de la capa de transporte TCP, fue diseñado tempranamente en los primeros años de la década del 70 en conjunto con el protocolo IP, con el propósito de interconectar las incipientes redes de conmutación de paquetes.

La estructura de un segmento TCP se muestra a continuación:

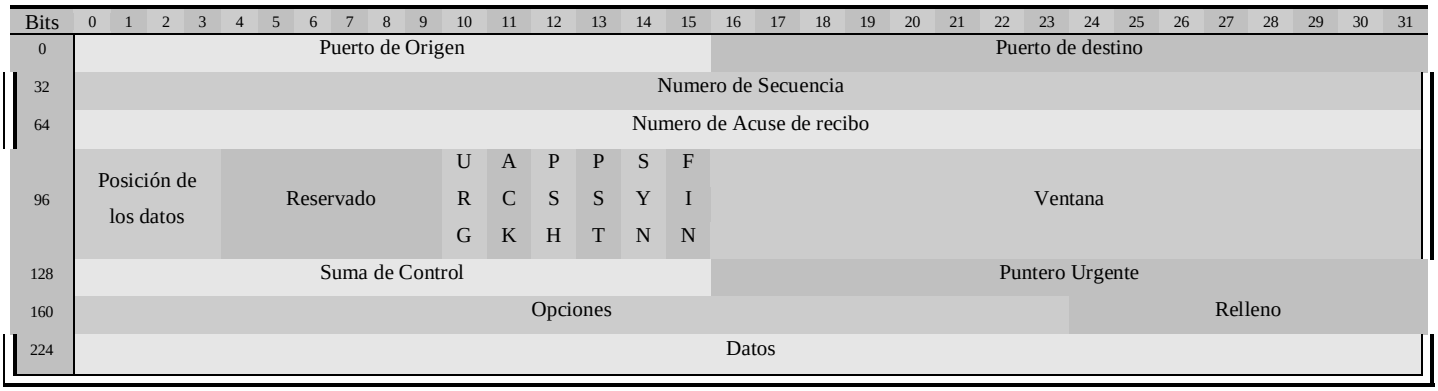


Figura 2. Estructura de un segmento de TCP

Según [10] (…) el propósito principal de TCP consiste en proporcionar un servicio de conexión o circuito lógico fiable y seguro entre pares de procesos. Para proporcionar este servicio encima de un entorno de internet menos fiable, el sistema de comunicación requiere de mecanismos relacionados con las siguientes áreas:

Transferencia básica de datos: Permite la transferencia de un flujo continuo de datos Full-Duplex, empaquetando la información en segmentos para su transmisión eligiendo cuando bloquear y enviar datos, de pendiendo de su conveniencia.

Fiabilidad: Para que la entrega de datos sobre una conexión TCP sea fiable, se usan los números de secuencia y de acuse de recibo de la cabecera del segmento TCP. A cada byte u octeto de datos a transmitir se le asigna un número de secuencia. El número de secuencia del segmento es el número de secuencia del primer byte de datos, el cual es transmitido en la cabecera TCP. Por otro lado el número de acuse de recibo, indica que todos los bytes anteriores a ese número de secuencia han sido recibidos por el modulo TCP receptor, e indica cual es el número de secuencia que se espera recibir a continuación. Así para cada byte transmitido se exige un número de acuse de recibo. Si este acuse de recibo o ack no es recibido dentro de un plazo preestablecido, los datos se retransmiten. Por otra parte los números de secuencia en el receptor se utilizan para reordenar los paquetes que hayan llegado en desorden y eliminar la información duplicada. Además es utilizado el campo Checksum de la cabecera TCP, para comprobar que los datos no se encuentren corruptos. De esta manera es posible ordenar, reconocer y recuperar los datos que se corrompen, pierden, dupliquen o se entreguen en forma desordenada.

Multiplexamiento: TCP entrega información entre dos procesos de distintos sistemas finales. El protocolo permite que un conjunto de procesos simultáneos en un mismo sistema se comuniquen con su respectivo par en algún sistema exterior. Para esto se utiliza el campo puerto de destino y puerto origen de la cabecera TCP. De esta manera a cada proceso que desee comunicarse se le asigna un número de puerto específico.

Conexiones: TCP mantiene información de estado para cada flujo de datos entre procesos. Cuando dos procesos desean comunicarse, se debe inicializar la información de estado en cada lado de la conexión, mantener la comunicación mientras esta se desarrolla y terminar la comunicación cuando esta termina para liberar recursos.

Prioridad y seguridad: Se puede indicar el nivel de seguridad y prioridad de la comunicación entre usuarios. Si esto no es necesario se indican valores por defecto.

Control de flujo: El receptor es capaz de controlar el flujo de datos a través del campo Ventana del segmento TCP, la que en cada ack indica el número de bytes que el receptor está dispuesto a aceptar más allá del último segmento recibido con éxito. Además cuenta con un sistema de control de congestión que será detallado a continuación.

2.1.4.1. Mecanismos de Evitación de Congestión en TCP

La forma en que TCP aborda el problema de la congestión, es través de 4 algoritmos (RFC5681): Slow Start; Congestion Avoidance; Fast Retransmit y Fast Recovery. A continuación se explicaran los mecanicos atingentes al modelo a trabajar.

2.1.4.1.1. Slow start

El algoritmo de Slow Start, es ocupado al comenzar el traspaso de datos en una conexión en la que no se conocen las características de la red, por lo que es necesario inyectar paquetes lentamente a esta para probar la conexión y determinar su capacidad disponible.

Para implementar Slow Start, se crean dos variables de estado por conexión: cwnd (congestion window) y rwnd (receiver's advertised window). La variable cwnd, es un límite del transmisor sobre la cantidad de datos que pueden ser inyectados a la red después de recibir un ack. Rwnd es un límite por el lado del receptor sobre la cantidad de datos pendientes o por recibir. El mínimo entre cwnd y rwnd, define el flujo de datos que será transmitido.

Además es necesario que exista la variable ssthreshold (slow start threshold) que define en qué momento debe dejar de usarse slow start y debe comenzar a utilizarse Congestion Avoidance para el control de la transmisión de datos.

La ventana inicial o IW debe cumplir los siguientes requerimientos dependiendo del tamaño del segmento más largo que el transmisor pueda enviar (SMSS: Sender Maximum Segment Size)

Si $SMSS > 2190\,bytes$, entonces $IW = 2 * SMSS\,bytes$ y no debe ser mayor que 2 segmentos.

Si $1095 < SMSS \leq 2190\,bytes$, entonces $IW = 3 * SMSS\,bytes$ y no debe ser mayor que 3 segmentos.

Si $SMSS \leq 1095\,bytes$, entonces $IW = 4 * SMSS$ y no debe ser mayor que 4 segmentos.

Una vez definida la ventana inicial, se recomienda que la implementación de este algoritmo incremente su ventana según la siguiente expresión:

$$cwnd_{n+1} = cwnd_n + min\,(N, SMSS) \qquad (1)$$

Donde N es el número de bytes que anteriormente no habían recibido acks y de los que se ha recibido acuse de recibo en el último ack. Esto supone un aumento exponencial de la ventana de congestión.

Una vez que $cwnd > sstrheshold$, el algoritmo Congestion Avoidance toma el control del flujo de datos.

2.1.4.1.2. Congestion Avoidance

Este algoritmo se divide en dos partes: Incremento Aditivo y Decrecimiento Multiplicativo (AIMD, Additive Increase Multiplicative Decrease). Si no existen pérdidas, entonces la ventana de congestión será incrementada linealmente (Additive Increase), por que el algoritmo asume que existe ancho de banda disponible. La ventana de congestión aumentará según la siguiente relación:

$$cwnd = cwnd + SMSS*SMSS/cwnd \quad (2)$$

Sin embargo si el algoritmo detecta una pérdida, es decir si recibe 3 ack iguales, entonces el valor de ssthreshold se reduce a la mitad del valor de la ventana de congestión actual y el valor de la ventana de congestión se iguala al valor del sstrheshold (Multiplicative Decrease). La información que no recibió ack, es decir que se perdió, es inmediatamente reenviada sin esperar que se cumpla el tiempo para retransmisión (Fast Retransmit). Luego la ventana de congestión sigue creciendo linealmente puesto que se ha superado el valor del ssthreshold artificialmente (Fast Recovery).

Un caso especial ocurre cuando la pérdida no se debe a la recepción de tres ack iguales consecutivos, sino a pérdida por finalización del tiempo de espera. Cuando el tiempo de espera es alcanzado, el valor de ssthreshold se iguala a la mitad de la ventana de congestion actual y posteriormente, la ventana de congestión se reduce a 1MSS (máximum segment size). Puesto que el valor de cwnd es menor que ssthrehold, se inicia el mecanismo de Slow Start, hasta que el nuevo ssthreshold es alcanzado.

Ethernet-Services Transport Protocol (ESTP), ocupa la misma cabecera del segmento TCP. El protocolo puede explicarse en tres partes: algoritmo de detección del nivel de congestión, función de asignación de MD (Multiplicative Decrease) y algoritmo de evitación de congestión. A continuación se detalla cada uno de sus partes.

A. Mecanismo de detección de congestión

El principio de este método está basado en la estimación del nivel de congestión de la red. Este nivel está relacionado con la cantidad de paquetes satisfactoriamente entregados entre 2 pérdidas consecutivas de paquetes. Esta cantidad es definida como un parámetro α, el cual es asignado a un perfil exponencial para determinar un multiplicador menos agresivo para la ventana de congestión definido como $map(\alpha)$. Entonces el valor de α es la distancia entre dos pérdidas de paquetes consecutivas tal como se muestra en la siguiente figura.

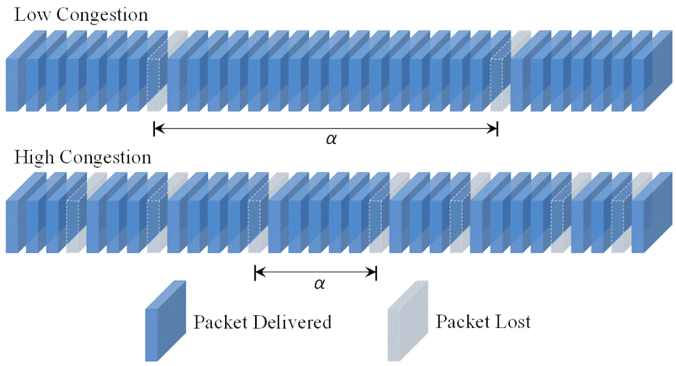


Figura 3. Detección de congestión

Se elige un perfil exponencial con el fin de relacionarlo con la distribución de probabilidad exponencial que exhibe el intervalo entre la pérdida de paquetes. De esta manera se tiene una ventana que es dependiente del nivel de congestión de la red, de manera que puede ser controlada de forma más eficiente.

El valor de α varía en el rango de 1 a ∞, incluyendo así los dos casos extremos donde 1 representa dos perdidas consecutivas e ∞, representa una conexión sin pérdidas. El valor de α, es el dominio de la función $map(\alpha)$, la cual varía en el rango [1;2]. Esto significa que el protocolo puede disminuir la ventana de congestión en un factor no mayor que 2 y no más pequeño que 1, en efecto se define que $map(\alpha = 1) = 2$ y $map(\alpha = \infty) = 1$, se observa que un factor 1 es seleccionado para una transmisión sin pérdidas, el cual naturalmente nunca será elegido por el protocolo. Este esquema de asignación es eficiente porque si α toma un valor pequeño, se asume que es debido al resultado de una red altamente congestionada, con lo que la ventana de congestión es reducida agresivamente. A su vez si α toma un valor alto, se asume que la red no está experimentando altos niveles de congestión y la ventana es reducida ligeramente.

B. Función de asignación

Para obtener la función de asignación del protocolo es necesario relacionar esta con la función de densidad de probabilidad (fdp) del periodo de tiempo entre dos pérdidas de paquetes. La probabilidad de que se pierda un paquete en la red sigue una distribución de Bernoulli. Si se asume que una pérdida de paquete es un evento exitoso, entonces el número de eventos exitosos en un periodo fijo de tiempo sigue una distribución de Poisson, es decir, la pérdida de paquetes puede ser modelada como una distribución de Poisson. Luego, el tiempo entre 2 eventos que siguen una distribución de Poisson, puede ser modelado como un proceso de distribución exponencial:

$$f(\alpha;\tau) = \begin{cases} \frac{1}{\tau} e^{-\frac{\alpha}{\tau}} & \alpha \geq 0 \\ 0 & \alpha < 0 \end{cases} \quad (3)$$

Donde τ es la esperanza de α ($E[\alpha]$).

Puesto que el principal objetivo es tener un control efectivo de la congestión, mientras se incrementa el rendimiento, la función de asignación es elegida para que calce con la fdp de la distancia métrica entre 2 paquetes. El factor MD de TCP, como se mencionó anteriormente es ½. La unidad de distancia de esta métrica, es definida como un solo paquete, independiente de su tamaño en bytes. Según esta definición, la mínima distancia entre dos paquetes es 1, puesto que la medida es tomada desde el comienzo de un paquete hasta el comienzo del siguiente. Esta función de asignación es puesta en el denominador del factor MD, de esta manera ESTP tiene como cota inferior el comportamiento del tradicional TCP, puesto que una de las condiciones de bordes es que si la red es altamente congestionada el factor debe ser 2, por lo que al dividir la ventana de congestión en 2, se tiene el comportamiento tradicional de TCP. A su vez, si nunca se tiene una pérdida, de tal manera que la distancia es infinita, el factor de ajuste deseado tenderá 1.

Como se dijo anteriormente, se desea que la función de asignación calce con la distancia entre 2 eventos. Puesto que $exp(-\infty) = 0$, entonces sumando 1 al termino exponencial, resulta:

$$map(\alpha) = \exp[-(\alpha\text{-}1)/\tau_\alpha] + 1 \quad (4)$$

Esta expresión también satisface que $map(1) = 2$, por lo que no es necesario realizar más cambios y la expresión final se muestra a continuación:

$$map(\alpha) = e^{-\frac{\alpha-1}{\tau\alpha}} + 1 \quad (5)$$

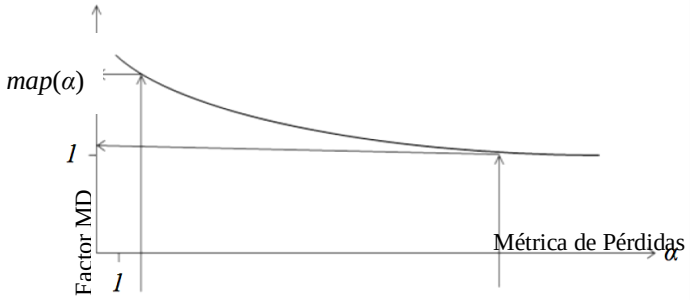


Figura 4. ESTP *map*(*α*) vs. *α*

Se observa que para el caso extremo de dos pérdidas consecutivas de paquetes (α = 1), ESTP se comporta como el protocolo TCP, es decir que la velocidad de transmisión de datos esta acotada por el rendimiento de velocidad de TCP. Puesto que el protocolo disminuye drásticamente la tasa de envío de datos cuando detecta una alta congestión, el protocolo actúa de manera justa, se comporta de forma amigable con TCP.

C. Control de evitación de congestión

Incorporando información de Ethernet-Services, en suma con la dinámica del ajuste del factor MD, ESTP, puede además mostrar mejoras en términos de su eficiencia en la velocidad de

transmisión, puesto que esas técnicas son independientes, unas de otras. Combinando el mecanismo de control de congestión, con características de QoS propias de Ethernet-Services, a saber CIR (Commited Information Rate) y EIR(Excess Information Rate) , se deriva la siguiente expresión para el control de congestión.

$$cwnd_{n+1} = \frac{cwnd_n - cwnd_{MIN}}{map(\alpha)} + cwnd_{MIN} \tag{6}$$

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

Esta técnica aprovecha toda la ventaja del ancho de banda provista al subscriptor, manteniendo la velocidad de flujo de datos sobre CIR. La velocidad de transferencia, puede además mejorarse utilizando la información de EIR. La ventana de transmisión máxima $cwnd_{MAX}$ puede ser obtenida similarmente a la ventana de congestión mínima, usando EIR en vez de CIR:

$$cwnd_{MAX} = RTT \cdot EIR \tag{7}$$

Con este valor, la cota superior de la ventana de congestión puede ser controlada. Una cota superior puede ser impuesta porque una vez que cwnd excede $cwnd_{MAX}$, la velocidad de transferencia de datos excede EIR y la política de tráfico fuerza a las capas más bajas a descartar los paquetes, que exceden esta tasa. Una vez que esto ocurre, el protocolo de capa 4, comenzará su mecanismo de control de congestión, disminuyendo la ventana. Esto es innecesario ya que no existe congestión en la red. Si se mantiene la ventana de congestión en su máximo valor, la velocidad de transferencia de datos no será reducida por el protocolo de control de congestión y el subscriptor obtendrá el máximo throughput, hasta que exista una pérdida aleatoria o que ocurra una pérdida por congestión. Puesto que la ventana de congestión es controlada por el servidor, ESTP solo necesita ser implementado en este, para observar un incremento en la velocidad de transmisión de datos, eliminando la necesidad de mejorar el equipamiento en redes privadas.

Entonces el incremento aditivo (AI) de ESTP se expresará como:

$$cwnd_{n+1} = \min\left(cwnd_{MAX}, cwnd_n + 1/cwnd_n\right) \tag{8}$$

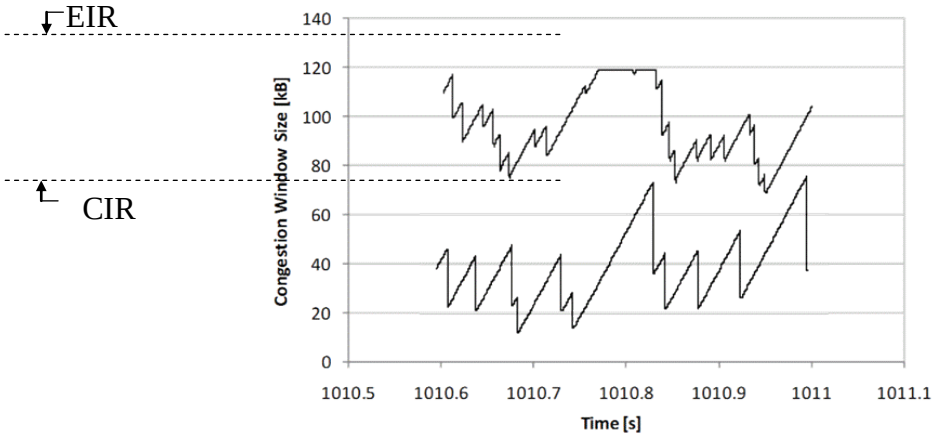


Figura 5. Comportamiento de la ventana de congestión de ESTP y TCP-SACK

2.1.4.2. Fairness

Se entiende por "Fairness", la medida de cuan justo es un protocolo al compartir el ancho de banda con otros a través del mismo enlace. Luego de discutir los mecanismos de ESTP, es necesario discutir brevemente cuan justo es este. La definición de fairness en protocolos de transporte que realizan control de congestion AIMD basados en la detección de congestion esta dado por la siguiente expresión:

$$\gamma = 4 \cdot \frac{1 - \Omega^2}{3} \tag{9}$$

Donde γ es el factor de incremento aditivo (additive increase) y Ω corresponde al factor de decrecimiento multiplicativo (multiplicative decrease). Esta definición se basa en los parámetros de TCP tradicional, donde $\gamma = 1$ y $\Omega = 1/2$. Esta igualdad asegura que otros protocolos de transporte no superen a los considerados justos tal como TCP. Lo que debe ser enfatizado es que esta condición aplica a los protocolos que solo realizan detección de congestión y no a los que realizan estimación de la intensidad de congestion. TCP tradicional reacciona ante un único evento de perdida de paquete, caso en el cual el algoritmo de control de congestión reduce la ventana de congestión a la mitad, es decir tiene un factor de decrecimineto multiplicativo fijo. Esto es análogo a un conductor en una carretera donde cada vez que se encuentra un grupo de vehiculos, sin tener en cuenta la cantidad de ellos, reaccione reduciendo su velocidad a la mitad. Si la intensidad de congestión es estimada, entonces la definición de fairness mostrada anteriormente no aplica porque el factor de reducción multiplicativo es ajustado dinámicamente evitando sobrepasar otros protocolos. Cuando el nivel de congestión es bajo, el protocolo se comporta de manera más agresiva que cuando el nivel de congestión es alto, donde el protocolo se comporta de manera amistosa. En el contexto de la analogía anterior ESTP se comporta como un conductor que reduce su velocidad suavemente cuando se encuentra con grupos pequeños de vehículos (bajo tráfico) y reduce su velocidad agresivamente al encontrarse con un grupo denso de vehículos (alto tráfico).

2.2. Estado del Arte[1]

El interés de los proveedores de servicios acerca de las utilidades de las redes Carrier Ethernet ha ido en aumento durante los últimos años. Esta valoración se debe a la alta tasa de transmisión de datos, flexibilidad y a la relación costo efectividad que proveen este tipo de redes, lo que conduce a un aumento en la inversión en negocios relacionados con Ethernet. Se espera que para fines del 2013, los ingresos globales en este tipo de servicios alcance los $US38.9 billones de dólares.

Las redes Carrier Ethernet, son diseñadas para cubrir áreas grandes de cientos de kilómetros a grandes tasas de transmisión (100Gbps). Por su naturaleza son consideradas redes de alto BDP (Bandwidth Delay Product), en las cuales los protocolos de transporte tienen problemas para sostener una tasa de transmisión alta.

Como resultado del creciente acceso a este tipo de redes, surge la necesidad de mejorar aspectos como la calidad de servicio (QoS) y con ello los métodos de control de congestión de tráfico en redes BDP. Otros protocolos han sido sugeridos para superar este problema, entre los que se encuentran Highspeed TCP, Scalable TCP y XCP, cada uno con ciertos inconvenientes.

En el caso de Highspeed TCP este se comporta al igual que TCP en un ambiente donde existe alta pérdida de paquetes y se observa un mejor comportamiento en transmisiones con una baja tasa de pérdidas.

Scalable TCP por su parte, aumenta la ventana de congestión de manera exponencial y evita la congestión disminuyéndola, multiplicando por un factor inferior a uno (Multiplicative Decrease). Este método provoca ráfagas de datos, debido al comportamiento agresivo en el tamaño de la ventana de congestión. Si las ráfagas de datos son monitorizadas y controladas a través de políticas de datos, pueden resultar en pérdidas. Además el control del tamaño de la ventana de congestión debe ser optimizado en cada red para que se observen las ventajas de este protocolo.

En el caso de XCP, el método descubre la información de congestión de la red, sin embargo este proceso no es compatible con todos los tipos de redes, por ejemplo con redes virtuales privadas (VPN).

Existen otros protocolos que reducen los efectos de un alto BDP, los cuales principalmente alteran los valores del algoritmo AIMD (Additive Increase, Multiplicative Decrease), del método de evitación de congestión.

Por otra parte, existe un tópico que ha despertado interés en el campo del diseño de la capa de transporte, esto es la estimación de intensidad de congestión. En particular, dos técnicas han sido sugeridas. La primera es una métrica Φ = goodput – throughput, mientras que la segunda

[1] El desarrollo del estado del Arte se basa en los papers realizados en torno a este trabajo, referenciados como: [2], [3] y [4]

es la distancia entre dos perdidas de paquetes. En ambos casos, la información es retroalimentada a algoritmos de incremento o disminución que mejoran el mecanismo de control de congestión.

La estimación de congestión basada en goodput-throughput (G-T) es discutida por Jung, Kim, Yeom, Kang, & Libman en 2011, aunque el concepto G-T fue sugerido con anterioridad en Jung, Kim, Yeom & Kang, 2010. El protocolo asociado a este trabajo es llamado Adaptive end-to-end Congestion control Protocol (ACP). Este protocolo mide las tasas de goodput y de throughput. El goodput, se entiende como la cantidad de datos que ha recibido acuse de recibo dentro de un rango de tiempo. El throughput corresponde a la cantidad total de datos enviados, independiente de su éxito para alcanzar el destino en un rango de tiempo. Se define la métrica $\phi = goodput - throughput$, como una medida del retraso observado en la red en un intervalo de tiempo para determinar la intensidad de congestion de la red. Si $\phi > 0$, significa que la cantidad de datos recibida es mayor que la cantidad de datos enviados en un mismo rango de tiempo y por lo tanto existe una alta probabilidad de que la red no se encuentre congestionada. Inversamente si $\phi < 0$, significa que la cantidad de datos de los que se han recepcionado acuses de recibo es mas baja que la cantidad que ha sido enviada en el mismo rango de tiempo, esto sugiere una alta probabilidad que la red se encuentre congestionada.

Junto con la estimación de la intensidad de congestión, el protocolo realiza una medida de fairness. El umbral de equidad (U) para determinar si el protocolo se comporta de manera agresiva o amistosa se calcula dividiendo la capacidad del enlace, por el número de conexiones que activamente envian datos a través del mismo (C/n). Si se define que el flujo i de datos ocupado en el enlace (w_i), es igual a $\frac{cwnd_i}{RTT}$, entonces se tiene la siguiente medida de fairness.

$$F_i = \frac{w_i}{U} \tag{10}$$

Al estimar F_i con parámetros obtenidos de la red se tiene que si la expresión anterior es mayor que 1, el ancho de banda ocupado por el enlace es mayor que el umbral de equidad, por lo que el protocolo se comporta de manera agresiva respecto a otros protocolos. Inversamente si F_i es menor que 1, quiere decir que el protocolo se comporta de manera amistosa puesto que ocupa un ancho de banda inferior que el umbral de equidad.

Cabe destacar que el mecanismo ocupado por ACP resulta ser muy sensitivo a los cambios de retraso y necesita un banco de pruebas muy especializado donde RTT es configurado en base al nivel de congestión, por lo que no se consideró la comparación de ESTP con este protocolo.

Se decide comparar ESTP con CUBIC, RENO y BIC, considerando que estos protocolos son parte del stack de métodos de control de congestión de tráfico incluidos por defecto en el kernel de Linux. En este sentido RENO ha sido considerado, ya que el comportamiento tradicional de TCP se ve reflejado en este protocolo. Este, disminuye la ventana de congstión a la mitad en cada evento de pérdida, presentando una reacción agresiva pero justa en el tratamiento de la congestión. Por otra parte, BIC posee un algoritmo de búsqueda binaria, donde la ventana de congestión crece al valor medio entre el último valor que tuvo cuando ocurrió una pérdida de paquete y su última medida. De esta manera prueba el ancho de banda, alcanzando altos niveles de Throughput, sin embargo no se comporta de manera justa frente a otros protocolos. Por último

CUBIC, es una modificación de BIC que lo transforma en un protocolo justo, posee un comportamiento cúbico en el cálculo de la ventana de congestión; además disminuye la misma, dependiendo del tiempo existente entre pérdidas.

En síntesis, este trabajo se centra en la implementación de ESTP, protocolo que colecta el nivel de congestión de la red basado en una métrica asociada a la distancia entre dos eventos de perdida de paquetes utilizando solamente información de la capa de transporte. Con lo anterior ajusta de forma inteligente la ventana de congestión, reduciendo el impacto de un alto BDP. Además implementa un algoritmo que solo necesita información disponible en la capa de transporte, por lo que su implementación solamente requiere que el servidor disponga del protocolo para ver incrementada la tasa de transmisión de datos, siendo de esta manera un protocolo escalable.

CAPÍTULO 3: IMPLEMENTACIÓN

3.1. Implementación de ESTP

La implementación de ESTP se efectuará en el sistema operativo Linux, específicamente en la versión de distribución Ubuntu v. 11.04.

Se elige Ubuntu como sistema operativo porque es posible instalar el método de congestión ESTP a través de la codificación de un módulo, el cual es compilado de manera independiente al sistema operativo completo, en conjunto con el beneficio de poder ser cargado o descargado dinámicamente. Además, Ubuntu Linux cuenta con distintos tipos de mecanismos de evitación de tráfico ya compilados, con los que se pueden realizar comparaciones, además de la facilidad en la creación de un método de pruebas, debido a las herramientas de compilación integradas al sistema.

3.1.1. Instalación de recursos.

Para la compilación e instalación de módulos es necesario instalar algunos recursos que no existen por defecto en la distribución de Ubuntu. A continuación se explica cómo se obtienen los recursos necesarios para la implementación, lo que es independiente del tipo de distribución que se disponga.

3.1.1.1. Cabeceras de Código Fuente

Se descargan las cabeceras de código fuente en caso de que no existan, para esto en un terminal de Linux se escribe la siguiente sentencia:

```
$ sudo apt-get install linux-headers-$(uname –r) linux-libc-dev kernel-package
```

Las cabeceras son desargadas en usr/src/

3.1.1.2. Código Fuente de la Distribución

Se descarga el código fuente de la distribución de Linux instalada, de esta manera se tiene acceso a los archivos fuentes de los módulos de congestión que ya existen en la distribución. Para la instalación, se ejecuta en una terminal la siguiente sentencia:

```
$ sudo apt-get install linux-source.<version>
```

El código fuente de la distribución es descargado en la ubicación usr/src/

3.1.1.3. Module Assistant

Para preparar el equipo para operar con módulos, es necesario descargar y configurar la herramienta module-assistant. Par esto se escribe la siguiente sentencia en un terminal:

```
$ sudo apt-get install module-assistant
```

Una vez instalado, se ejecuta la herramienta a través de la siguiente sentencia:

```
 $sudo module-assistant
```

Una vez ejecutado, se observa la siguiente imagen en pantalla:

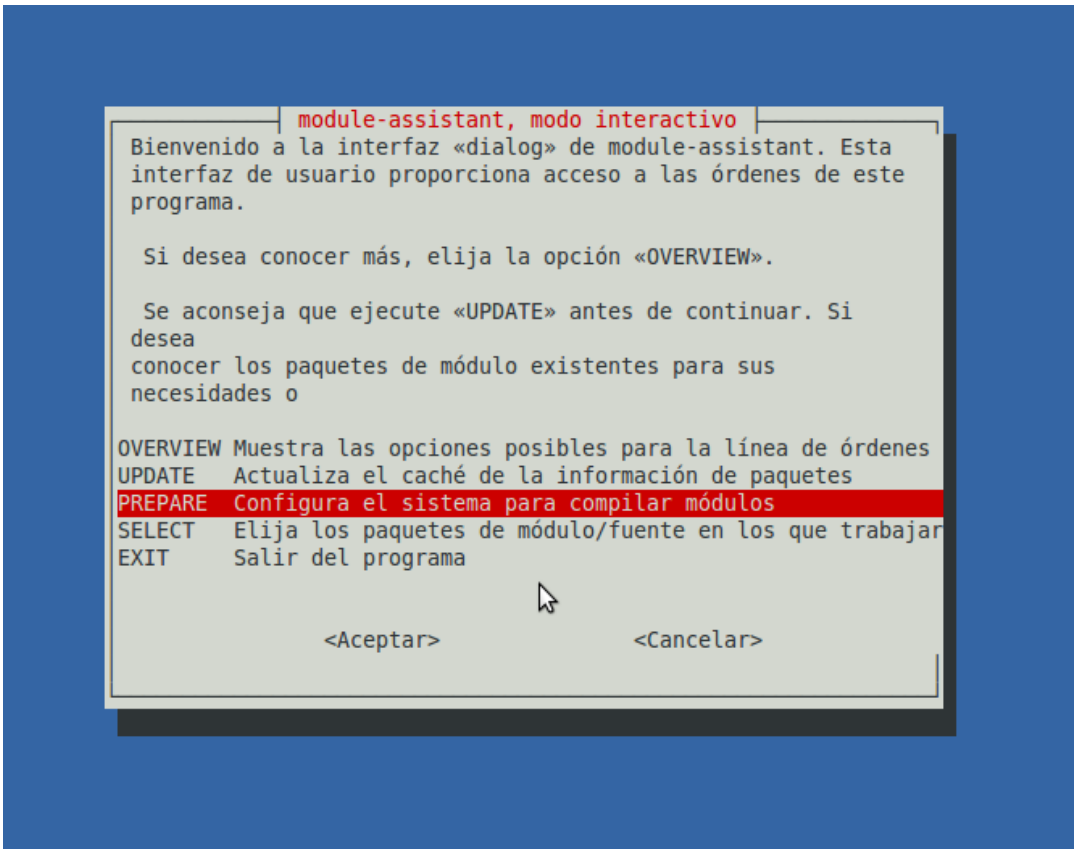


Figura 6. Menú de module assistant.

Posteriormente, en el menú gráfico se ejecuta “PREPARE”, la herramienta descargará todo lo necesario para compilar módulos.

3.1.2. Compilación de Módulos de evitación de tráfico por defecto

Para compilar los módulos de congestión de tráfico incluidos en los archivos fuente, se debe ingresar a la siguiente ubicación:

```
$ cd /usr/src/Linux-source-xxx/net/ipv4
```

Una vez dentro de esta carpeta, se modifica el archivo Makefile, agregando las siguientes sentencias.

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

Finalmente se compila, escribiendo en la ubicación de la carpeta la sentencia que se observa a continuación:

```
$sudo make
```

La acción anterior genera la compilación de la carpeta completa, que genera código binario de todos los archivos. En particular, solo interesan los archivos relacionados con los métodos de congestión, a saber: TCP_BIC.c, TCP_CUBIC.c, TCP_westwood.c, TCP_highspeed.c, TCP_hybla.c, TCP_hTCP.c, TCP_vegas.c, TCP_veno.c, TCP_scalable.c, TCP_yeah.c y TCP_illinois.c. Para cada uno de los archivos fuente anteriores, se generan archivos binarios .o y .ko. Los archivos .ko son los únicos requeridos para la carga dinámica de los métodos de congestión.

3.1.3. Carga y ejecución de módulos de congestión.

Para montar un modulo de congestión de tráfico .ko ya compilado en el sistema operativo, se utiliza el utilitario insmod como se muestra a continuación.

```
$ sudo insmod /ubicacion/NombreModulo.ko
```

Con la acción anterior el módulo queda montado en el sistema operativo. A través del comando "lsmod", se verifica que el modulo se encuentre cargado. Para que TCP ocupe el método de congestión del módulo, es necesario cargarlo ocupando el utilitario "sysctl" como se muestra a continuación.

```
$ sudo sysctl –w net.ipv4.TCP_congestion_control=modulo
```

Con lo anterior para cada socket TCP, se utilizará el método de congestión seleccionado. Es importante destacar que al llamar al módulo por medio de "sysctl", se debe ocupar el nombre definido dentro del archivo fuente, a continuación se muestra el fragmento de código de TCP_CUBIC.c donde se define el nombre de llamada, en este caso "CUBIC".

```
static struct TCP_congestion_ops TCP_CUBIC = {
.init = CUBIC_init,
.ssthresh = CUBIC_ssthresh,
.cong_avoid = CUBIC_cong_avoid,
```

```
.min_cwnd = TCP_RENO_min_cwnd,
.owner = THIS_MODULE,
.name = "CUBIC"
```

Para desmontar el modulo de congestión de tráfico se utiliza el comando “rmmod” como se muestra a continuación:

```
$ sudo rmmod modulo
```

Es importante considerar que esta operación puede fallar si existen sockets ocupando el módulo de congestión de tráfico, por lo que se recomienda cerrar todas las conexiones TCP existentes antes de bajar el módulo, o en último caso forzar el retiro del módulo con el flag -f.

3.1.4. Código Fuente de ESTP

Debido a que el kernel de Linux está programado en lenguaje de programación C, el modulo de congestión es también programado en este lenguaje. En esta sección se presenta el aspecto final del código fuente, explicado en sus partes funcionales, mientras que en la sección Anexos, se muestra el código fuente completo.

3.1.4.1. Librerías

Las siguientes librerías son incluidas.

```
#include <linux/module.h>
#include <net/TCP.h>
#include <linux/time.h>
```

3.1.4.2. Variables de Entrada

Al cargar el modulo ESTP es posible definir 4 variables de entrada, Estas son “cir”, “promedio_flag”, “debug” y “MSS” las que tienen valor por defecto 500, 0, 0 y 1460 respectivamente. La variable “cir”, tiene relación con CIR (2.1.4.1.2 sección C), esta es expresada en Mbits, y corresponde a la tasa mínima de velocidad permitida. Por otra parte la variable promedio_flag, se utiliza para indicar al módulo si debe calcular un promedio de los RTT obtenidos en la conexión para ajustar la ventana o bien siempre ocupar el RTT medido. Además se define de manera externa el valor de MSS en 1460 para todas las aplicaciones. Por último se crea una variable externa llamada “id_test”, con el fin de tener un identificador de conexión de transporte.

```
int cir=500;
int promedio_flag=0;
int debug=0;
int MSS=1460;
module_param(cir,int,S_IRUGO);
module_param(promedio_flag,int,S_IRUGO);
module_param(debug,int,S_IRUGO);
module_param(MSS,int,S_IRUGO);
```

```
int id_test;
```

3.1.4.3. Implementación de función map(α)

La ecuación para $map(\alpha)$, mostrada en la ecuación (5) es una función de carácter exponencial de acuerdo a la función de asignación expuesta.

$$map(\alpha) = e^{-\frac{(\alpha-1)}{\tau_\alpha}} + 1 \tag{5}$$

El kernel de Linux tiene la limitación de trabajar solo con aritmética de enteros, por lo que no es posible trabajar con esta expresión directamente. Ante esto, se divide la función en rangos de operación para los distintos valores de α. El recorrido de map(α), se encuentra acotado en el intervalo [1,2] debido al término constante 1. Así es posible dividir el intervalo [1,2] a valores regulares, para los que existe un único valor de α que satisface la función anterior. Estos valores críticos de alpha, logran cuantizar el dominio de $map(\alpha)$ en una cantidad fija de valores determinando intervalos de operación. Luego, para cualquier alpha, es posible determinar a qué intervalo de operación pertenece y es posible asignarle el valor de alpha crítico superior del intervalo al cual pertenece.

Debido a que $map(\alpha)$ depende de τ_α y al ser este la esperanza de α, la dependencia con la tasa mínima garantizada (CIR), es directa. Asumiendo que $\tau_\alpha = CIR * MSS$, entonces se tiene la expresión completa para el cálculo de los alphas.

$$map(\alpha) - 1 = e^{\frac{-(\alpha-1)}{(CIR*MSS)}} \tag{11}$$

La implementación entonces, debe depender tanto del valor de CIR como de MSS. Lo anterior se logra creando una estructura, que guarde los valores de los α críticos que dividen el intervalo para cada CIR a MSS fijo. En particular se divide el intervalo [1,2] obteniendo 32 valores intermedios para $map(\alpha)$, con estos se calculan 34 valores de alphas críticos de los que se rescatan 33 útiles (se omite el caso en que alpha=Inf). Debido a la dependencia de los alpha críticos con CIR se crea una función que para cada CIR calcula los alpha criticos.

Para implementar lo anterior, se escribe la siguiente rutina en MATLAB, que escribe el código para la función Obtener_segmentacion_alphas en formato C, la cual se agrega como función del módulo.

```
function [alphas] = Escribe_funcion_segmentacion(mss,nombre_archivo_out)
% La función escribe en el archivo de salida una funcion c, que segmenta
% los valores de map(alpha), dependendo del rango de alpha
% Crea casos de cir desde 10 hasta 1000 (de 10 en 10),con mss 1460
% genera función c:  char alpha_to_punt(u32 alpha, int cir), donde cir
% es el valor que se ingresa como argumento al cargar modulo ESTP

map = 3-(1:1/33:2)';
fo=fopen(nombre_archivo_out,'W');
fprintf(fo,'void Obtener_segmentacion_alphas(struct ESTP_estructura_datos m[32],int
cir){\n\t');
fprintf(fo,'switch (cir){\n')
```

```
for j=1:100
    alphas = round(1-log(map-1)*j*10*mss);
    fprintf(fo,'\t\tcase %d:\n',j*10);
    for i=0:length(alphas)-2
        fprintf(fo,'\t\t\tm[%d].valor=%d;\n',i,alphas(i+1));
        if (i==(length(alphas)-2))
            fprintf(fo,'\t\tbreak;\n');
        end
    end
end
    fprintf(fo,'\t}\n}');
    fclose(fo);
end
```

La función anterior genera para cada valor de CIR desde 10 hasta 1000 (de 10 en 10), los valores críticos para la función de asignación $map(\alpha)$ y crea la función Obtener_segmentacion_alphas. Esta función recibe una estructura vacía de 33 valores a los que les asigna valores críticos de alpha dependientes de CIR. A continuación se muestra un extracto de la función creada.

```
void Obtener_segmentacion_alphas(struct ESTP_estructura_datos m[33],int cir){
      switch (cir){
            case 10:
                  m[0].valor=1;
                  m[1].valor=450;
                  m[2].valor=914;
                  m[3].valor=1393;
                  m[4].valor=1887;
                  m[5].valor=2400;
                  m[6].valor=2931;
                  m[7].valor=3482;
                  m[8].valor=4054;
                  m[9].valor=4650;
                  m[10].valor=5272;
                  m[11].valor=5921;
                  m[12].valor=6600;
                  m[13].valor=7312;
                  m[14].valor=8061;
                  m[15].valor=8851;
                  m[16].valor=9685;
                  m[17].valor=10570;
                  m[18].valor=11512;
                  m[19].valor=12520;
                  m[20].valor=13602;
                  m[21].valor=14770;
                  m[22].valor=16041;
                  m[23].valor=17432;
                  m[24].valor=18971;
                  m[25].valor=20690;
                  m[26].valor=22640;
                  m[27].valor=24890;
                  m[28].valor=27552;
                  m[29].valor=30810;
                  m[30].valor=35010;
                  m[31].valor=40930;
                  m[32].valor=51050;
```

```
            break;
            case 20:
                  m[0].valor=1;
                  m[1].valor=900;
            .
            .
            .
```

Cada valor de alpha se relaciona con un intervalo entre valores críticos y a cada valor crítico se le asigna un valor de la siguiente estructura de datos.

```
static const struct ESTP_aimd_val {
       unsigned int md;
}AIMD_ESTP[]={
{128,/*(1) 128/(2^8) = 0.500 */},
{124,/*(2) 124/(2^8) = 0.484 */},
{120,/*(3) 120/(2^8) = 0.469 */},
{116,/*(4) 116/(2^8) = 0.453 */},
{112,/*(5) 112/(2^8) = 0.438 */},
{108,/*(6) 108/(2^8) = 0.422 */},
{104,/*(7) 104/(2^8) = 0.406 */},
{100,/*(8) 100/(2^8) = 0.391 */},
{96, /*(9)  96/(2^8) = 0.375 */},
{92, /*(10) 92/(2^8) = 0.359 */},
{88, /*(11) 88/(2^8) = 0.344 */},
{84, /*(12) 84/(2^8) = 0.328 */},
{80, /*(13) 80/(2^8) = 0.313 */},
{76, /*(14) 76/(2^8) = 0.297 */},
{72, /*(15) 72/(2^8) = 0.281 */},
{68, /*(16) 68/(2^8) = 0.266 */},
{64, /*(17) 64/(2^8) = 0.250 */},
{60, /*(18) 60/(2^8) = 0.234 */},
{56, /*(19) 56/(2^8) = 0.219 */},
{52, /*(20) 52/(2^8) = 0.203 */},
{48, /*(21) 48/(2^8) = 0.188 */},
{44, /*(22) 44/(2^8) = 0.172 */},
{40, /*(23) 40/(2^8) = 0.156 */},
{36, /*(24) 36/(2^8) = 0.141 */},
{32, /*(25) 32/(2^8) = 0.125 */},
{28, /*(26) 28/(2^8) = 0.109 */},
{24, /*(27) 24/(2^8) = 0.094 */},
{20, /*(28) 20/(2^8) = 0.078 */},
{16, /*(29) 16/(2^8) = 0.063 */},
{12, /*(30) 12/(2^8) = 0.047 */},
{8,  /*(31) 8/(2^8) = 0.031  */},
{4,  /*(32) 4/(2^8) = 0.016  */},
{0,  /*(33) 0/(2^8) = 0.000  */},
};
```

El cálculo del valor de reducción de la ventana de congestión se obtiene restando a 256 el valor de esta estructura y posteriormente dividiendo en 2^8. Con esto se obtienen valores entre [0.5; 1], que corresponden al recorrido de $1/map(\alpha)$

La función obtener_indice_AIMD es quien relaciona los valores de alpha con su correspondiente valor en la estructura AIMD_ESTP. Esta función se observa a continuación.

```
char Obtener_indice_AIMD(u32 alpha, struct ESTP_estructura_datos segmentacion_alphas[33]){
	int i=0;
	char punt=32;
	for (i=0;i<33;i++){
		if (alpha < segmentacion_alphas[i].valor){
			punt=i;
			i=33;
		}
	}
	return punt;
}
```

3.1.4.4. Inicialización de variables y estructuras

Se utiliza una estructura común de datos llamada "ESTP_estrctura_datos".

```
static struct ESTP_estructura_datos{
	 u32 valor;
}
```

Esta estructura posee un único miembro llamado valor. Posteriormente se generan 2 arreglos basados en esta estructura, "rtts_ultimos_valores[10]" y "segmentación_alphas[33]". El arreglo "rtts_ultimos_valores" se utiliza solo si el el flag para el promedio de RTTs está activo. En este caso la estructura se utilizará como un buffer donde cada valor obtenido de RTT es guardado. En el caso de "segmentación_alphas" se utiliza para guardar los valores críticos de alpha dependientes de CIR.

Por otra parte, se genera una estructura de datos para mantener en memoria las variables obtenidas de eventos anteriores de pérdida y se crea una instancia de esta estructura llamada datos para la conexión actual.

```
static struct ESTP {
	int id_test;
	int contador_rtts;
	u32 seq_number_anterior;
	u32 ultimo_alpha;
	int primer_intento_flag;
	int estructura_rtts_llena;
	unsigned int rtts_anterior;
	int no_hay_seq_number_flag;
} datos;
```

Luego al iniciarse una conexión de transporte, se invoca la función "ESTP_init", donde se inicializan las variables de la estructura anterior. Además se asigna la ventana de congestión máxima como el menor valor entre el valor actual y 2^{25}. En la iniciación se guarda en segmentacion_alphas, los valores de alpha críticos basados en CIR. Además se obtiene un

sequence number de referencia para el nivel de pérdida, este número se asocia al "snd_nxt" que corresponde al siguiente sequence number que será enviado en la conexión de transporte.

Finalmente, se escribe información de debug que muestra datos relevantes acerca de los parámetros de inicialización del protocolo.

```
static void ESTP_init(struct sock *sk)
{
	struct TCP_sock *tp = TCP_sk(sk);
	struct ESTP *ca = inet_csk_ca(sk);
	datos.contador_rtts=0;
	datos.estructura_rtts_llena=0;
	datos.no_hay_seq_number_flag=0;
	datos.seq_number_anterior=tp->snd_nxt;
	datos.rtts_anterior=0;
	datos.ultimo_alpha=0;
	Obtener_segmentacion_alphas(segmentacion_alphas,cir);
	tp->snd_cwnd_clamp = min_t(u32, tp->snd_cwnd_clamp, 0xffffffff/128);
	//Elige el mínimo entre  snd_cwnd_clamp y un
	//numero de 32 bits con solo unos
	id_test+=1;
	datos.id_test=id_test;

	printk(KERN_INFO "**************NUEVA PRUEBA %d**************\n",datos.id_test);
	printk(KERN_INFO "*\n*\tSe definio cir=%d\n",cir);
	printk(KERN_INFO "*\n*\tSe definio promedio_flag=%d\n",promedio_flag);
	printk(KERN_INFO "*\tEl equipo funciona con HZ = %d", HZ);
	printk(KERN_INFO "************************************************");
}
```

3.1.4.5. Factor Additive Increase

En la función ESTP_cong_avoid, se decide si se aplica slow_start o additive increase. Esta función es llamada cuando llega un ACK y si existen al menos tantos paquetes en vuelo como el tamaño de la ventana de congestión.

```
static void ESTP_cong_avoid(struct sock *sk, u32 ack, u32 in_flight)
{
	struct TCP_sock *tp = TCP_sk(sk);
	if (!TCP_is_cwnd_limited(sk, in_flight)){
		return;
	}
	if (tp->snd_cwnd <= tp->snd_ssthresh){
		TCP_slow_start(tp);
	}else{
		/* Do additive increase */
		if (tp->snd_cwnd < tp->snd_cwnd_clamp) {
			tp->snd_cwnd_cnt += 1;
			if (tp->snd_cwnd_cnt >= tp->snd_cwnd) {
				tp->snd_cwnd_cnt -= tp->snd_cwnd;
				tp->snd_cwnd++;
			}
		}
	}
```

```
}
```

Como se observa en la implementación, esta función recibe tres parámetros, un puntero a a una estructura sock, el ack recibido y la cantidad de paquetes en vuelo.

De la estructura sock se obtiene un puntero a una estructura TCP_sock que está definida en TCP.h. Si la ventana de congestion no está limitada, simplemente retorna. En otro caso si la ventana de congestión (snd_cwnd) es menor que el umbral de slow start (snd_ssthresh), se realiza slow_start. Si la ventana de congestion es mayor que slow start threshold entonce se analiza si la ventana de congestion es menor que la ventana de congestion máxima (snd_cwnd_clamp), si es cierto se le suma uno al contador snd_cwnd_cnt. Posteriormente se analiza si el contador es mayor que la ventana de congestion actual. En caso afirmativo, al contador se le resta la ventana de congestion actual y este último se acrecenta en 1. El comportamiento es idéntico al presentado por TCP_RENO.

3.1.4.6. Factor Multiplicative Decrease

Basado en la implementación de TCP_highspeed, se agrega el factor MD en la función ESTP_ssthresh. A continuación se explicará el código asociado a esta, la cual es invocada cuando se detecta un evento de pérdida debido a la obtención de un tercer ack duplicado. Se comienza con la inicialización de las siguientes variables.

```
static u32 ESTP_ssthresh(struct sock *sk){
	int i;
	unsigned char indice_aimd_ESTP;
	const struct TCP_sock *tp = TCP_sk(sk);
	u32 promedio_rtts;
	u32 seq_number_actual,alpha,rtt,cwnd_min,rtts;
```

Posteriormente se calcula el valor de RTTs. Por defnición este valor es un múltiplo de 8 que depende de la granularidad de tiempo del procesador, con lo que al dividir por la cantidad de HZ se tiene el tiempo en unidades de segundo. Por lo anterior y debido que el valor es múltiplo de 8, se corren 3 bits hacia la derecha que es semejante a dividir por 2^3 para tenerlo en unidades normalizadas de tiempo. Luego se multiplica por 10^6 para convertirlo en unidades de us.

```
	rtts = ((tp->srtt)>>3 ) *(1000000/ HZ); //us
	/*srtt es un multiplo de 8 por definición*/
```

Es importante destacar que el orden de las operaciones es relevante en el momento de la realización de cálculos debido a la restricción del kernel para trabajar en números enteros.

En el caso que se active la bandera promedio_flag en tiempo de carga del modulo de congestión, los RTTs se tratarán como se muestra a continuación.

```
	if(promedio_flag!=0){
		datos.rtts_anterior=datos.rtts_anterior+rtts-
					(int)rtts_ultimos_valores[datos.contador_rtts].valor;
		if(!datos.estructura_rtts_llena){
			promedio_rtts=datos.rtts_anterior/(datos.contador_rtts+1);
```

```
            }else{
                  promedio_rtts=datos.rtts_anterior/10;
            }
            rtts_ultimos_valores[datos.contador_rtts].valor=rtts;
            datos.contador_rtts++;
            if(datos.contador_rtts==10){
                  datos.contador_rtts=0;
                  datos.estructura_rtts_llena=1;
            }
      }
```

La variable rtts_anterior corresponde a la sumatoria de los RTTs guardados anteriormente más el valor actual obtenido menos el primer valor guardado. En otras palabras, el RTT obtenido actualmente reemplaza uno de los valores que se tenían almacenados con anterioridad. En la medida que se va llenando esta estructura se calcula el promedio con los RTTs que se han obtenido hasta el momento. Una ves que se ha llenado el buffer, se comienza a sobrescribir el valor más antiguo con el valor actual, de tal manera que siempre se calcula el RTT con los últimos 10 valores obtenidos.

Finalmente como se observa en el siguiente fragmento de código, si promedio_flag es activado, entonces el valor de RTT se obtendrá del promedio de los RTTs obtenidos, en otro caso se utilizará el valor obtenido en esa iteración.

```
if(promedio_flag!=0){
      rtt = promedio_rtts;
}else{
      rtt = rtts;
}
```

Para calcular el valor de alpha es necesaria la obtención del sequence number donde se produjo la pérdida. Con este fin se utiliza la función obtener_seq_number que analiza la existencia del valor snd_una en la estructura TCP_sock. Si existiera un problema con la obtención de este valor entonces se activa el flag “no_hay_seq_number_flag”.

Luego se analiza el valor obtenido, en funcionamiento normal, “seq_number_actual” es mayor que “seq_number_anterior”, en este caso el valor de alpha se calcula como la diferencia entre estos. Esta diferencia retorna el número de bytes entre pérdidas, luego al dividir por MSS se obtiene el número de paquetes entre ambas pérdidas, lo que corresponde a alpha.

```
d     seq_number_actual = Obtener_seq_number(sk,&datos.no_hay_seq_number_flag);
      if((seq_number_actual>=datos.seq_number_anterior) &&!datos.no_hay_seq_number_flag){
            alpha = (seq_number_actual - datos.seq_number_anterior)/MSS;
            datos.seq_number_anterior=seq_number_actual;
            datos.ultimo_alpha=alpha;
      }else if(!datos.no_hay_seq_number_flag){
            alpha = ((0xffffffff-datos.seq_number_anterior)+seq_number_actual)/MSS;
            datos.seq_number_anterior=seq_number_actual;
            datos.ultimo_alpha=alpha;
            if(debug!=0) printk(KERN_INFO "%d)SEQ NUMBER SOBREPASADO\n",datos.id_test);
      }else{
            alpha=datos.ultimo_alpha;
            datos.no_hay_seq_number_flag=0;
            if(debug!=0) printk(KERN_INFO "%d)NO HAY SEQ NUMBER ESTRUCTURA
VACIA\n",datos.id_test);
```

```
        }
```

Debido a que los sequence number son número aleatorios que crecen en el tiempo, se da el caso en que "sequence_number_actual" es menor que "sequence_number_anterior". Esto se da porque el máximo valor del tipo u32 (32 bits sin signo) ha sido superado. En este caso se calcula esta diferencia en bytes como la resta entre el valor máximo "0xffffffff" y "sequence_number_anterior" y posteriormente sumando "sequence_number_actual".

En el caso que no exista la estructura de datos, se asigna a alpha su último valor calculado. Este comportamiento no debería darse nunca y experimentalmente no se dio el caso. Sin embargo se mantiene en el código en caso que ocurra una falla extraña y exista un evento de pérdida.

Como ya se tiene el valor de alpha, entonces es posible asignar el valor de la estructura AIMD_ESTP que le corresponde. Este valor es guardado en la variable "índice_aimd_ESTP" como se observa en la siguiente sentencia.

```
        indice_aimd_ESTP=Obtener_indice_AIMD(alpha,segmentacion_alphas);
```

En el siguiente fragmento de código se muestra la obtención de la ventana de congestión mínima.

```
        cwnd_min = cir*rtt/(8*MSS); //CIR[Mbits], rtt[us],MSS[bytes] => [Packets * s]
        //: cwnd_min=(CIR[bytes/s]/MSS[bits])*(rtt[us]/1000000)
        //1000000 es para dejar RTT en segundos
```

La ventana de congestión mínima como se comentó en el capítulo 2, se calcula según la expresión, mostrada en el código anterior. Dado que CIR se encuentra en unidades de [Mbits] y RTT en [us], los magnificadores se cancelan. Por otra parte dividiendo por MSS en bits, se tiene el tamaño de la ventana de congestión en unidad de [$paquetes \cdot segundo$].

Con todos los datos anteriores, ahora es posible ejecutar el cálculo para la reducción de la ventana de congestión. El siguiente fragmento de código muestra como esta reducción es realizada.

```
if(tp->snd_cwnd > cwnd_min){
        u32 delta_cwnd=(tp->snd_cwnd - cwnd_min);
        /*256 Por que si AIMD_ESTP= 128 => (2⁸-2⁷)/2⁸=1-1/2=0.5 .
          Si AIMD_ESTP=0 (2⁸-2⁰)/2⁸ = 1      */
        return max(delta_cwnd * (( 256 - AIMD_ESTP[indice_aimd_ESTP].md) >> 8) + cwnd_min,
                    cwnd_min);//a>>b => a/(2^b)
        }else{
                return cwnd_min;
        }
}
```

En caso que la ventana de congestion a enviar "snd_cwnd" sea menor que "cwnd_min" calculado anteriormente, entonces se obliga al protocolo a enviar "cwnd_min" calculado. Esto responde a la necesidad de mantener el troughput sobre "cir". En caso contrario, se calcula la diferencia de "snd_cwnd" con "cwnd_min" que llamaremos "delta_cwnd". Es a esta diferencia, a la que se aplicará la reducción.

Luego la ventana de congestión que se enviará, corresponde al valor máximo entre la ventana de congestión mínima y la apropiada reducción de "delta_cwnd", sumando "cwnd_min". Se tiene que, si existe el más alto nivel de pérdida "índice_aimd_ESTP" tomará el valor más bajo posible, es decir 0. Luego como "AIMD_ESTP[0]" es igual a 128, al realizar la resta y dividiendo por 2^8, se tiene que la reducción para delta_cwnd es:

$$factor\ de\ reduccion = \frac{256 - 128}{2^8} = \frac{2^8 - 2^7}{2^8} = 0.5 \quad (12)$$

Que corresponde a una reducción de ventana idéntica a la de TCP_RENO en el peor de los casos.

Por otra parte en el caso en que la distancia entre pérdidas de paquetes sea muy grande, "índice_aimd_ESTP", tomará el valor más alto posible, es decir 32. Luego como "AIMD_ESTP[32]" es igua a 0, al realizar la resta y dividiendo por 2^8, se tiene que la reducción para delta_cwnd es:

$$factor\ de\ reduccion = \frac{256 - 128}{2^8} = \frac{2^8 - 2^7}{2^8} = 0.5 \quad (13)$$

$$factor\ de\ reduccion = \frac{256 - 0}{2^8} = \frac{2^8 - 0}{2^8} = 1 \quad (14)$$

Es decir, no existe reducción en la ventana si existe una gran distancia entre las pérdidas medidas.

Es importante notar la diferencia entre una conexión con Carrier-Ethernet y una conexión TCP normal. Olvidando por un instante el cálculo del nivel pérdida de la conexión de transporte reflejado por el cálculo de alpha[2], las condiciones especificadas para Carrier-Ethernet, solo se asemejarían a las condiciones de una conexión normal TCP, si CIR tomara el valor 0. De esta manera, se igualaría el punto de referencia para la reducción de ventana y ante la existencia de un alto nivel de pérdida de paquetes, ESTP se comportaría como TCP_RENO, reduciendo la ventana de congestión a la mitad.

3.2. Implementación de un marco de pruebas

Una vez diseñado el protocolo, es necesario crear una ambiente controlado, que permita la repetitividad e independencia estadística de los pruebas que permitirán analizarlo para observar si cumple con las hipótesis planteadas en este documento. La implementación del marco de pruebas se dividirá en cuatro etapas para su explicación.

1. Topología de Red

[2] La función $map(\alpha)$ deja de tener sentido si $CIR = 0$

2. Configuración de variables del sistema operativo.
3. Configuración de un ambiente de pruebas
4. Configuración de rutinas para la generación de datos.

3.2.1. Topología de Red

La topología de red que se utiliza para la obtencion de datos consiste, en 2 computadores conectados a través de cable cruzado categoría UTP-6 para conexiones Gigabit Ethernet. Ambos equipos cuentan con tarjetas de red de 1[GB/s], memoria RAM de 4[GB] y sistema operativo Ubuntu 12.04.

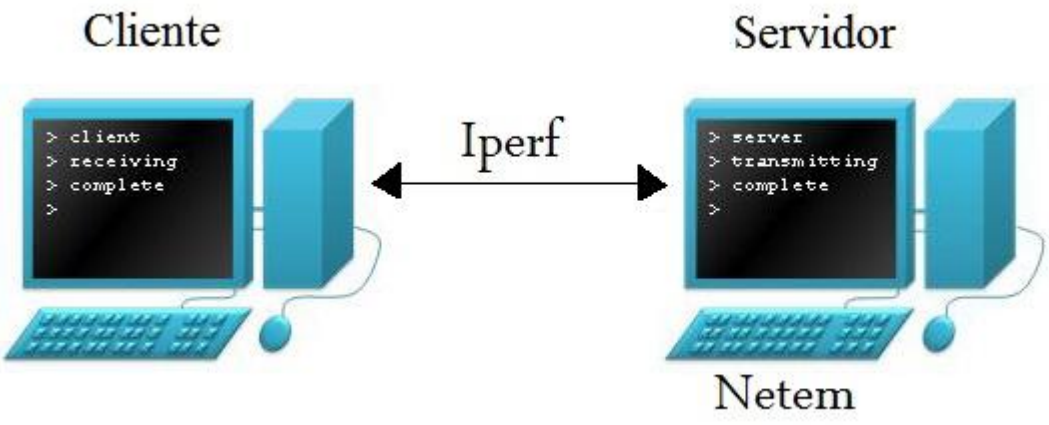


Figura 7. Topología de red para la obtención de pruebas

3.2.2. Configuración de variables del sistema operativo

El núcleo del sistema operativo cuenta con una serie de variables que pueden ser modificadas dinámicamente en tiempo de ejecución, sin la necesidad de recompilar el sistema. Estos parámetros modificables, son todos aquellos que se encuentran en la carpeta /proc/sys/ y se modificarán con el comando "sysctl".

En particular, se modifican los parámetros asociados a networking que se relacionan con net/ipv4 y algunas variables del nucleo asociadas a tamaños del buffer de transmición.

Se opta por crear una rutina bash, que modifique estas variables de sistema, luego, cualquier aplicación de test que se realice en el equipo, debe considerar correr esta rutina, con el fin de que las pruebas se generen en un mismo ambiente y por tanto puedan ser comparables y repetibles.

A continuación se presentan todas las modificaciones y la explicación de las variables modificadas.[3]

1. **net.ipv4.TCP_abc=1**: Controla el Appropiate Byte Count. Incrementa el contador de ventana de congestion cuando se recibe ack de un segmento completo.

[3] Las modificaciones especificadas en esta sección y su explicación, están basadasen: [6], [7] y [9]

2. **net.ipv4.TCP_adv_win_scale=7**: Cuenta el almacenamiento en buffer de sobrecarga como bytes/(2^7)
3. **net.ipv4.TCP_abort_on_overflow=1**: Permite resetear las conexiones en caso que la conexión se muy lenta o si se produce una sobrecarga por ráfaga de datos
4. **net.ipv4.TCP_fack=0**: Se desactiva el soporte para Forward Acknowledgment
5. **net.ipv4.TCP_moderate_rcvbuf=1**: Si esta activo TCP realiza una auto configuración del buffer de recepción no mas grande que TCP_rmem.
6. **net.ipv4.TCP_no_metrics_save=1**: TCP guarda distintas métricas cuando una conexión es realizada para que las conexiones en el futuro se establezcan con condiciones iniciales. Si esta activo, no se guardan estas métricas.
7. **net.ipv4.TCP_orphan_retries=0**: Número de veces que se sondea una conexión cerrada por en el propio host.
8. **net.ipv4.TCP_retrans_collapse=0**: No envía paquetes completos en las retransmisiones.
9. **net.ipv4.TCP_syncookies=0**: Se desactiva el envío de syncookies, que viola el protocolo.
10. **net.ipv4.TCP_mem='65535 1048575 16777215'**: Estos tres valores [bajo,presión,alto] permiten a TCP saber cuanta memoria puede usar. El primer valor, indica que TCP no regula la asignación de memoria bajo este valor. El segundo parámetro indica a TCP si excede esta cantidad de memoria, TCP debe moderar el consumo de esta hasta alcanzar el primer valor. El ultimo, indica cual es la máxima memoria que TCP puede asignar globalmente.
11. **net.ipv4.TCP_rmem='65535 1048575 16777215'**: Estos tres valores [min,default,max], son usados por TCP para ajustar dinámicamente el tamaño de buffer de recepción, dependiendo de la memoria disponible del sistema. Min, es la minima ventana de congestión utilizada por TCP. Default, corresponde al tamaño default del buffer de recepción. Este valor sobrescribe net.core.rmem_default, definido para todos los protocolos. Max, es el máximo tamaño del buffer, este valor no sobrescribe a net.core.rmem_max
12. **net.ipv4.TCP_wmem='65535 1048575 16777215'**: Semejante a TCP_rmem, controla el buffer de envío.
13. **net.ipv4.TCP_max_ssthresh=1661992959**: configura el valor máximo del umbral.
14. **net.core.wmem_max=131071**: Tamaño máximo en bytes, del buffer del socket de envío.
15. **net.core.rmem_max=131071**: Tamaño máximo en bytes, del buffer de recepción.

3.2.3. Configuración de un ambiente de pruebas

El ambiente de pruebas diseñado, consiste en una red con pérdida de paquetes y delay variable. Se definen para este experimento tres niveles de pérdida: alto, medio y bajo que corresponden a un 1%, 0.01% y 0.0001% de pérdida respectivamente. Se definen periodos de 10 segundos, en estos la pérdida se mantendrá en un nivel medio durante los primeros 8 segundos mientras que en los 2 segundos restantes se aplicará un nivel de pérdida alto. El nivel de pérdida bajo será configurado dinámicamente, en caso que el flujo de datos sea menor que CIR. En la siguiente figura se observa una representación del esquema antes presentado.

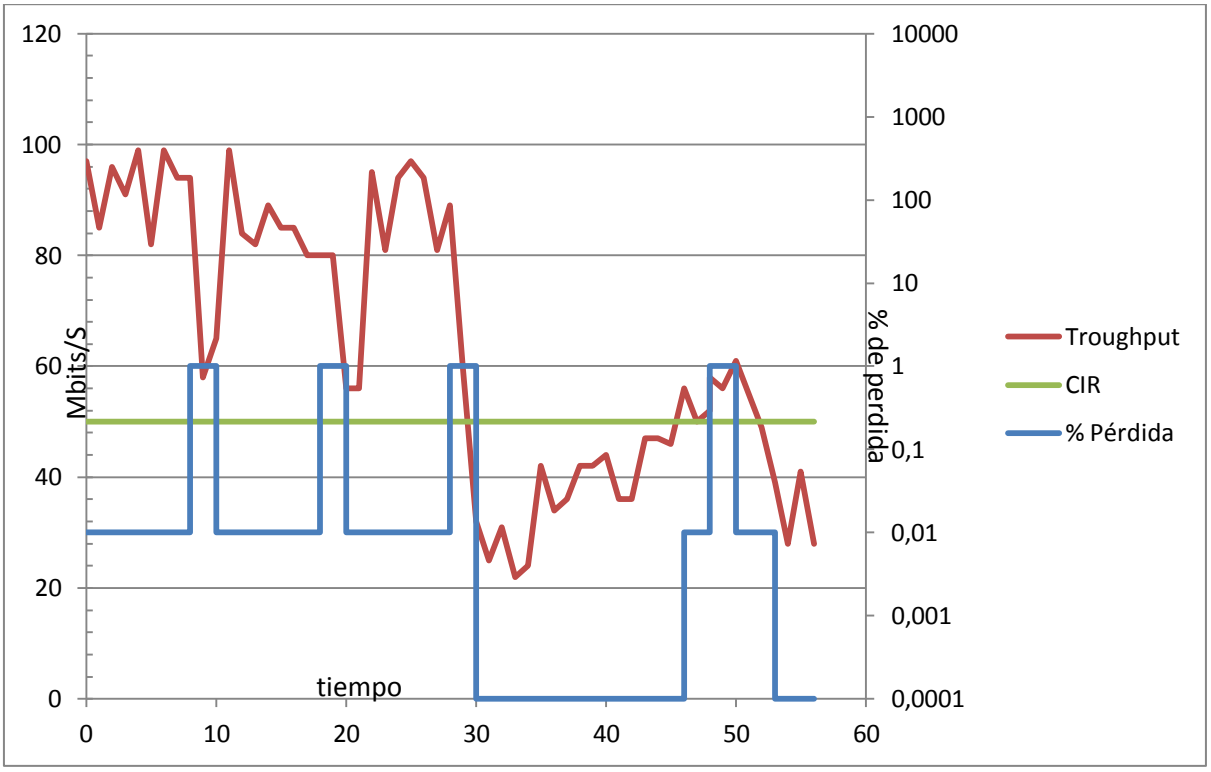


Figura 8. Gráfico de ambiente de pruebas.

Como se observa en la figura, el caudal de datos es representado por la curva roja (magnitudes en eje izquierdo) mientras que el porcentaje de pérdida es representado por la línea azul(magnitudes en eje derecho) mientras que el eje de las abcisas representa el tiempo en segundos. En el segundo 29, se observa que el throughput de datos, disminuye bajo el umbral CIR, en este caso el nivel de pérdida es reducido abruptamente. Este comportamiento representa un caudal de datos garantizado en que la pérdida es muy baja. Por otra parte, el nivel medio de pérdida representa un esquema de mejor esfuerzo en la entrega de datos. El nivel alto de pérdida permite caracterizar el comportamiento del protocolo en un ambiente de alta congestión. De esta manera a través de este testbed, se caracteriza una conexión en los tres casos posibles de un servicio Carrier Ethernet, es decir un ancho de banda garantizado, un ancho de banda de máximo esfuerzo y por último un ancho de banda máximo caracterizado por la capacidad física de la conexión.

Los protocolos que se prueban son: BIC, CUBIC, ESTP y RENO. Cada test se configura para que tenga una duración total de 300[s], a RTT constante. Cada experimento se repetirá 2 veces para valores de RTT que variarán entre 1 y 100 [ms] (entre 1 y 10 con un paso de 1[ms] y entre 10 y 100 con un paso de 10[ms]). Además cada prueba se realizará con la configuración de variables especificada en la sección anterior.

3.2.3.1. Software utilizados.

Para el esquema de pruebas antes presentado, se utiliza una versión de iperf modificada, un programa escrito en C que modifica el nivel de pérdida dinámicamente dependiente del caudal de datos obtenido por iperf y una rutina escrita en bash, que permite la coordinación entre ambos programas y el módulo TCPprobe modificado, las repeticiones por prueba, la variación del delay y las configuraciones tanto de conexión, como del sistema operativo, del protocolo a probar y el guardado de los datos para su posterior análisis.

El software iperf, se modifica para que escriba en un segmento de memoria compartido el troughput obtenido mientras que la rutina escrita en C, lee el mismo segmento detectando cualquier variación en el caudal de datos. En el caso que este disminuya bajo el valor de CIR, se disminuirá el porcentaje de pérdida a un nivel más bajo, en caso contrario operará según el esquema periódico definido en la sección anterior. El cambio en el nivel de pérdida y delay es realizado a través de llamadas a sistema, a través del software netem.

El configurador de pérdida llamado “testbed_loss”, recibe 6 argumentos:

1. Periodo de congestión
2. Sub periodo de alta pérdida
3. RTT
4. % pérdida baja
5. % pérdida media
6. % pérdia alta

Por otra parte, el programa rescata desde variables de ambiente previamente configuradas, el valor de CIR, EIR, y la ubicación de la instalación donde se encuentra el segmento compartido, configurado en 1Kb

```
#define SHM_SIZE 1024  /* make it a 1K shared memory segment */
#define SIZE 512
```

Posteriormente se crea una llave que apunta a la ubicación del segmento.

```
  #include <sys/ipc.h>
  #include <sys/shm.h>

    key_t key;
    /* Se crea la llave para el segment compartido */
    if ((key = ftok(file_key, 'R')) == -1) {
        perror("ftok");
        exit(1);
    }
```

Luego se conecta y si es necesario se crea el segmento.

```
    if ((shmid = shmget(key, SHM_SIZE, 0644 | IPC_CREAT)) == -1) {
        perror("shmget");
        exit(1);
    }
```

Posteriormente se une el segmento a un puntero adecuado, en este caso el puntero es de tipo “double”.

```
    double *data;
    int shmid;
    data = shmat(shmid, (void *)0, 0);
    if (data == (double *)(-1)) {
        perror("shmat");
        exit(1);
    }
    int mode;
```

Por comodidad se asigna el valor de “data” a la variable “rate”

```
    double rate;
    rate = * (double *) data;
```

Con lo anterior se tiene la implementación de lectura del segmento compartido. Una vez terminada la aplicación se elimina la unión al segmento a través del siguiente código.

```
if (shmdt(data) == -1) {
        perror("shmdt");
        exit(1);
}
```

Con esto solo basta implementar la lógica explicada anteriormente dependiente de los valores de throughput obtenidos dinámicamente. A continuación se muestra el caso en que CIR es menor que 0 para evidenciar cómo se implementa el uso de Netem.

```
while(rate>-2) {
   rate = * (double *) data;
   if (rate<CIR){
      sprintf(palabra,"sudo tc qdisc replace dev eth0 root netem delay %dms loss %s%%\n",
              delay, l_loss);
      system(palabra);
      printf("%s \n", palabra);
   }else{
       .
       .
       .
```

Como se observa, cuando la variable “rate” es menor que “CIR”, se crea un string con la operación a enviar al sistema operativo. Las variables “delay” y “l_loss” son obtenidas en tiempo de invocación del programa via argumentos. Se observa una llamada a system(), que invoca una terminal y ejecuta la sentencia

```
sudo tc qdisc replace dev eth0 root netem delay %dms loss %s%%\n
```

Esto indica que se debe reemplazar la regla aplicada a la interfaz eth0 con permisos de super usuario a través de netem para asignar un cierto valor de delay en [ms] y una pérdida en %. De esta manera se implementa el configurador de pérdida asociado a la conexión.

Es importante destacar que se desarrolla un pequeño programa utilitario, con una estructura similar al archivo presentado actualmente llamdo "set_rate", con la finalidad de escribir un valor de troughput arbitrariamente en el segmento que es recibido como argumento.

```
* (double*) datac = atoi(argv[1]);
```

La sentencia anterior escribe en el segmento el argumento recibido por el programa utilitario. Es la rutina bash quien invoca este utilitario y al terminar la conexiónde iperf escribe el valor -3 en el segmento compartido, lo que permite cerrar el configurador de pérdida dinámicamente.

La modificación del archivo iperf, es bastante sencilla y sigue el mismo esquema de conexión al segmento compartido. En particular se modifica el archivo ReportDefault.c, con la intención de que además de escribir en pantalla la información del troughput de datos de la conexión, escriba el mismo valor en el segmento compartido en unidades de Mbits/s, de tal manera que la comparación entre CIR y el flujo de datos sea consistente. La sentencia que calcula el troughput y lo escribe en el segmento se muestra a continuación:

```
* (double*) datac = (stats->TotalLen / (stats->endTime - stats->startTime))*8/1000000;
```

La rutina de datos en bash define cuales protocolos serán ejecutados en la ronda de pruebas. Para esto se crea un arreglo con los nombres de los protocolos a ejecutar.

```
#! /bin/bash
#typeset cong_name[15]
. ../../lyd.env
d="_d_"
t="_t_"
cong_name[0]="ESTP"
cong_name[1]="CUBIC"
cong_name[2]="RENO"
cong_name[3]="BIC"
cong_name[4]="highspeed"
```

Luego para cada protocolo en el arreglo, se ejecutará una cantidad de 2 tests para cada RTT en el rango entre 10 y 100 [ms]. Esto puede observarse en el siguiente fragmento.

```
for cong in  "${cong_name[@]}"
do
       for (( delay=10; delay<=100; delay=delay+10 ))
       do
              for (( test=1; test<=2 ; test++ ))
              do
              .
              .
              .
              done
       done
done
```

La rutina espera un tiempo definido externamente como variable de ambiente, para ejecutar la siguiente prueba y posteriormente ejecuta la configuración de ambiente del sistema operativo.

```
sleep $LyD_SLEEP_TIME
sudo $LyD_HOME_PATH/Testbed/Common/./opciones_buff
```

Luego el Protocolo ESTP es cargado dentro de los protocolos disponibles

```
sudo /sbin/./insmod $LyD_HOME_PATH/Testbed/Common/Protocol/$LyD_ESTP_KO $LyD_ESTP_CONFIG
```

Posteriormente se asigna al test actual el protocolo que se usa y se guarda un archivo de "log" indicando la hora en que se realiza el test.

```
sudo sysctl -w net.ipv4.TCP_congestion_control=$cong
FECHA=$(date +%F--%T)
$FECHA >> $LyD_HOME_PATH/Testbed/LyD_Testbed/TCP_probe/log/$cong$d$delay$t$test.log
```

A continuación se utiliza el utilitario para escribir en el segmento compartido un valor mayor que -3 de tal manera que testbed_loss tenga un valor que permita la iniciación correcta de la rutina.

```
$LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./set_rate 10
```

Luego se invoca la rutina testbed_loss en background con parámetros definidos como variables de ambiente y con el delay asociado a la prueba actual.

```
$LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./testbed_loss $LyD_T_CONG $LyD_T_TOTAL $delay
$LyD_LOW_LOSS $LyD_MEDIUM_LOSS $LyD_HIGH_LOSS &
```

Posteriormente se inicia el modulo TCP_probe que escribirá la información relevante de cada paquete en el archivo TCPprobe.out en modo silencioso y se guarda el número de proceso asociado para finalizarlo más tarde.

```
sudo /sbin/./insmod $LyD_HOME_PATH/Testbed/Common/Protocol/TCP_probe.ko
sudo chmod 640 /proc/net/TCPprobe
sudo cat /proc/net/TCPprobe > $LyD_HOME_PATH/Testbed/LyD_Testbed/Temp/TCPprobe.out &
TCPCAP=$!
```

A continuación se inicia la conexión entre hosts con el archivo iperf modificado entregando como argumento variables asociadas al test. El flag –f le indica al programa que muestre los valores de flujo de datos en unidades de Mbits/s.

```
$LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./iperf -i $LyD_DISPLAY_TIME -t $LyD_TEST_TIME
-f m -c $LyD_HOST -p $LyD_PUERTO
```

Una vez que la conexión ha terminado se aborta el proceso TCPprobe que escribe en el archivo de salida y se escribe en el segmento compartido el valor -3 para que testbed_loss termine de ejecutarse.

```
sudo kill $TCPCAP
$LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./set_rate -3
```

Luego se copia el archivo originado a una carpeta predefinida con un nombre acorde para su posterior análisis. El nombre del archivo tiene relacion con el modulo de congestion ejecutado, el delay asociado a la prueba y el valor asociado a la repetición del test.

```
sudo cp $LyD_HOME_PATH/Testbed/LyD_Testbed/Temp/TCPprobe.out
$LyD_HOME_PATH/Testbed/LyD_Testbed/TCP_probe/$cong$d$delay$t$test.info
```

Finalmente se remueven el modulo TCP_probe y el módulo TCP_ESTP. Antes de remover ESTP, se asigna el modulo de congestion por defecto “CUBIC”.

```
#close all
sudo /sbin/./rmmod TCP_probe
sudo sysctl net.ipv4.TCP_congestion_control=CUBIC
sudo /sbin/./rmmod TCP_ESTP
```

En los test se hace referencia a variables de ambiente cargadas antes de ejecutar las pruebas. El archivo de configuración contiene la caracterización total de las pruebas experimentales, tal como se presenta a continuación.

```
######################### CONFIGURACION IPERF ############################
#puerto de conexion iperf
export LyD_PUERTO="5001"

#Tiempo total del test en segundos
export LyD_TEST_TIME=300

#IP de conexion iperf
export LyD_HOST="10.0.0.2"

#Intervalo de muestreo en pantalla en segundos
export LyD_DISPLAY_TIME=1

################## CONFIGURACION DE PARAMETROS DE TCP ##################
#Maximum Segment size
export LyD_MSS=1460
#Commited Information Rate
export LyD_CIR=500
export LyD_EIR=1000

################ CONFIGURACION MODULOS DE CONGESTION ###################
#Nombre archivo de compilacion de ESTP que se cargara
export LyD_ESTP_KO="TCP_ESTP.ko"

#Nombre modulo de congestion que se usara
export LyD_MODULE_NAME="ESTP"
```

```
#Promedio de rtts. Si es distinto de 0 se calcula el promedio de rtts
export LyD_PROM_FLAG=0

#Informacion de salida del modulo. Si es distinto de 0, el modulo
#escribe informacion extra
export LyD_DEBUG=0

#Nombre de archivo de salida
export LyD_LOG_NAME="ESTP_test"

#Configuracion al cargar modulo ESTP
export LyD_ESTP_CONFIG="cir=$LyD_CIR promedio_flag=$LyD_PROM_FLAG
debug=$LyD_DEBUG MSS=$LyD_MSS"

###################### TESTBED LOSS CONFIGURACION ###################
#Tiempo de congestion
export LyD_T_CONG=2
#Periodo total. Suma entre periodos de perdida alta y media
export LyD_T_TOTAL=10
#% Perdida baja cuando el throughput es menor que la ventana de
#  congestion
export LyD_LOW_LOSS=0.0001

#%Perdida en periodos de baja congestion
export LyD_MEDIUM_LOSS=0.01

#%Perdida en periodos de alta congestion
export LyD_HIGH_LOSS=1

######################### OTRAS CONFIGURACIONES ######################
#Tiempo de espera entre una prueba y otra
export LyD_SLEEP_TIME=10

#Ruta de instalacion
export LyD_HOME_PATH=`echo ~`
```

3.3. Obtención de Resultados

3.3.1. Configuración de rutinas para la obtención de datos

La rutina de pruebas, mediante el modulo TCP_probe, generó un total de 152 archivos.

$$4\ (protocolos) * 19(RTTs) * 2(repeticiones) = 152\ (archivos) \qquad (15)$$

Como cada prueba tiene una duración de 300[s] más un desfaz entre pruebas de 10[s], se tiene que la duración total del test es de 13,08 horas.

$$\frac{310[s] * 152(archivos)}{60[s] * 60[min]} = 13{,}08[horas] \quad (16)$$

Debido a que cada archivo tiene una resolución en el tiempo del orden de los nanosegundos y que cada dato obtenido corresponde a un paquete de datos enviado, esto generó una masa de datos de 10.5[Gbytes].

Un ejemplo de los datos obtenidos se muestra a continuación.

Tiempo	Puerto Origen	Puerto Destino	Largo	SND NXT	SND UNA	SND CWND	SSTHRESHOLD	SND WND	SRTT
0.191133526	45200	5001	32	0x1e72019f	0x1e71ce9f	10	2147483647	65535	250
0.191149867	45200	5001	32	0x1e72019f	0x1e71ceb7	10	2147483647	33292288	222
0.191164333	45200	5001	32	0x1e72183f	0x1e71da07	11	2147483647	33292288	197
0.191249059	45200	5001	32	0x1e72183f	0x1e71e557	12	2147483647	33292288	176

Tabla 1. Ejemplo de datos obenidos

Cabe destacar que de las variables anteriores se utilizarán: tiempo, SND_UNA, SND_CWND y SSTHRESHOLD. La variable SND_UNA indica el último ack recibido, en base a este valor se calculará el flujo de datos de la transmisión en Mbits/s. SSTHRESHOLD da cuenta del umbral entre los mecanismos de Slow Start y Additive Increase. SND_CWND, corresponde a la ventana de congestión calculada por el protocolo.

Debido a la cantidad de datos anterior se genera una rutina en lenguaje C++, con el fin de generar datos analizables. La codificación de la rutina, se encuentra en los anexos.

El primer paso que realiza el programa es generar tres ficheros para cada archivo creado por TCP_probe. Cada uno de ellos, representa una tabla de dos columnas, en las que se guarda el tiempo y el promedio por segundo para SND_CWND, SSTHRESHOLD y el flujo de datos.

El flujo de datos se calcula tomando la diferencia entre el primer y último SND_UNA en el intervalo de un segundo, lo que corresponde a la diferencia en bytes transmitidos hasta ese instante. Multiplicando por 8 para obtener la diferencia en bits y dividiendo por 10^6, se obtiene la tasa de trasnmisión de datos en Mbits/s

Un ejemplo de los archivos generados se muestra a continuación.

.cwnd		.thr		.sst	
TIME	CWND	TIME	Throughput	TIME	SSThreshold
1	2286	1	924,235	1	2,15E+09
2	6675	2	934,434	2	2,15E+09
3	9750,5	3	934,31	3	2,15E+09
4	10818	4	934,554	4	2,15E+09
5	11182,5	5	934,662	5	2,15E+09
6	11319,5	6	934,312	6	2,15E+09

Tabla 2. Ejemplo de archivos generados en primera iteración.

Posteriormente para cada uno de los archivos generados, se realiza el promedio segundo a segundo de las dos repeticiones, generándose un archivo para cada protocolo y RTT. Lo anterior permite obtener el comportamiento en el tiempo de cada uno de los protocolos a RTT fijo. Luego de creados los 19 archivos por variable a considerar, se entrelazan línea a línea a través del comando "paste" disponible en bash, generando en total tres archivos por protocolo que contienen su comportamiento en el tiempo para cada RTT. Un ejemplo para el entrelazado línea a línea de 2 archivos se muestra a continuación donde en "file_out.txt" se guarda el resultado de la operación.

```
paste file_input_1.txt file_input_2.txt >> merged_file.txt
```

Luego se busca caracterizar la relación entre los datos y la variación de RTT, para esto se calcula para cada archivo generado, el promedio de los datos de CWND, Throughput, y Sstrheshold. Con lo anterior se genera, para cada protocolo en una segunda iteración, tres archivos que muestran el comportamiento del protocolo frente a las variaciones de RTT, que siguen la misma estructura de dos columnas indicadas anteriormente. Luego para las tres variables de interés, se mezclan los resultados de cada protocolo en un único archivo, generando así tres archivos de salida.

De esta manera, se han generado 15 archivos de datos útiles para realizar el análisis de los datos obtenidos. En base a estos archivos se obtienen las curvas características, disponibles en en la sección Anexos.

CAPÍTULO 4: DISCUSION DE RESULTADOS

4.1. Comparación de resultados experimentales de ESTP con la teoría

En el capítulo 2, se explicitó el fundamento teórico en que se basa el protocolo implementado. En resumen existen 3 rasgos característicos que debe cumplir el protocolo para estar de acuerdo a la teoría, estos son:

1. El comportamiento del flujo de datos en el tiempo debe ser superior a CIR, es decir, a la tasa garantizada de datos con el fin de cumplir las exigencias del servicio Carrier Ethernet.
2. La ventana de congestión debe ser reducida, dependiendo del nivel de pérdida, calculado en base a la función $map(\alpha)$, la que se comporta en forma agresiva si la distancia en paquetes entre pérdidas es baja y en forma sutil, si esta distancia es grande.
3. El protocolo, reduce el efecto de cuello de botella en la capa de transporte en la medida que aumenta RTT

A continuación, se analizan estas características para mostrar que el protocolo implementado cumple con lo esperado teóricamente.

4.1.1. Comportamiento del flujo de datos en el tiempo

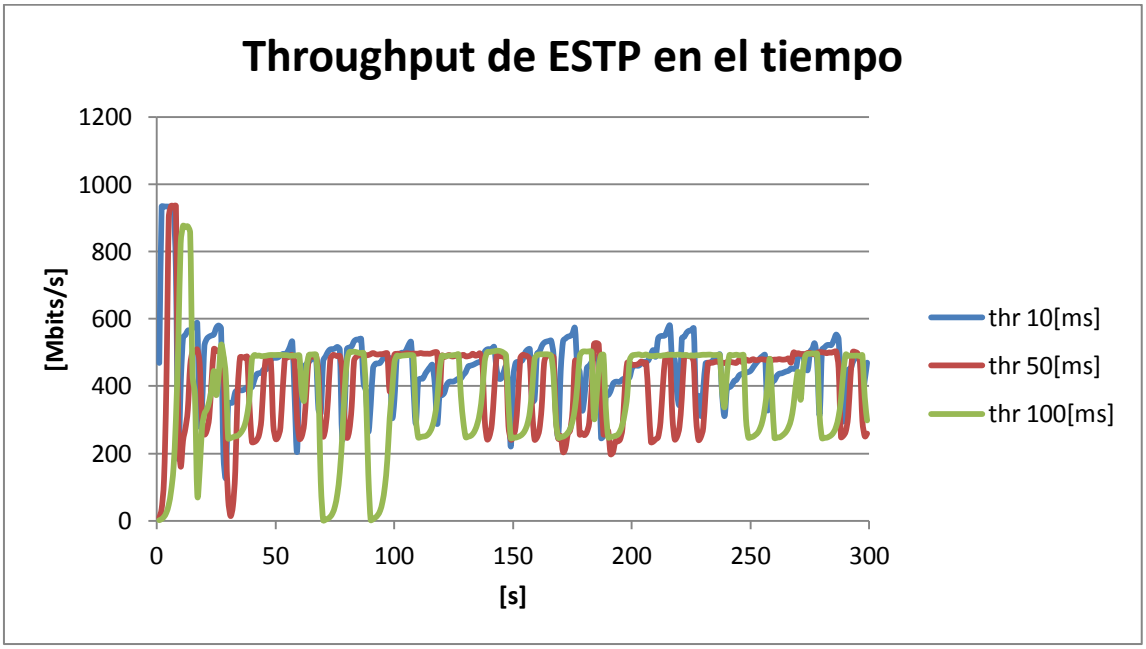


Figura 9: Gráfico de Throughput de ESTP en el tiempo

Se puede observar que el flujo de datos tiende a mantener una velocidad de transferencia cercana a los 500[Mbits/s], que corresponde al CIR configurado en el protocolo. Sin embargo también se muestra que este valor, en la medida que el RTT aumenta, resulta ser una cota superior para el caudal de datos, por lo tanto se ve un problema en la tasa de datos garantizada. Este problema puede ser resuelto para cumplir con la calidad de servicio, aumentando el valor de CIR a una tasa mayor.

Caracterizando aproximadamente la curva obtenida se observa que a partir de los 100 segundos, los valores de throughput oscilan con una amplitud cercana a los 200[Mbits/s], en torno a los 370[Mbits/s]. En general la tasa se mantiene por más tiempo cercana a CIR que en tramos donde la velocidad de datos disminuye de 500 [Mbits/s]. Dada la forma que tiene la oscilación, puede aproximarse de manera burda a una función sinusoidal a la que es posible extraer su valor eficaz, mediante la división de su amplitud por $\sqrt{2}$, semejándose a un escalón que decae a aproximadamente ¾ del valor de amplitud de la oscilación. Ante esta aproximación, se tiene una oscilación de 150[Mbits/s] en torno a los 425[Mbits/s]. Por lo tanto, aumentando el valor de CIR en un 15%, se tendrán oscilaciones alrededor de 500[Mbits/s], lo que en promedio cumple con la tasa máxima garantizada.

En este sentido, el protocolo cumple con la expectativa de tasa garantizada, pero a un nivel inferior que el configurado, posiblemente debido a que el valor de RTT dinámicamente calculado en el protocolo, no se condice al 100% con el RTT configurado. Esta aseveración puede observarse en el grafico de la ventana de congestión promedio para cada RTT y su valor esperado teóricamente.

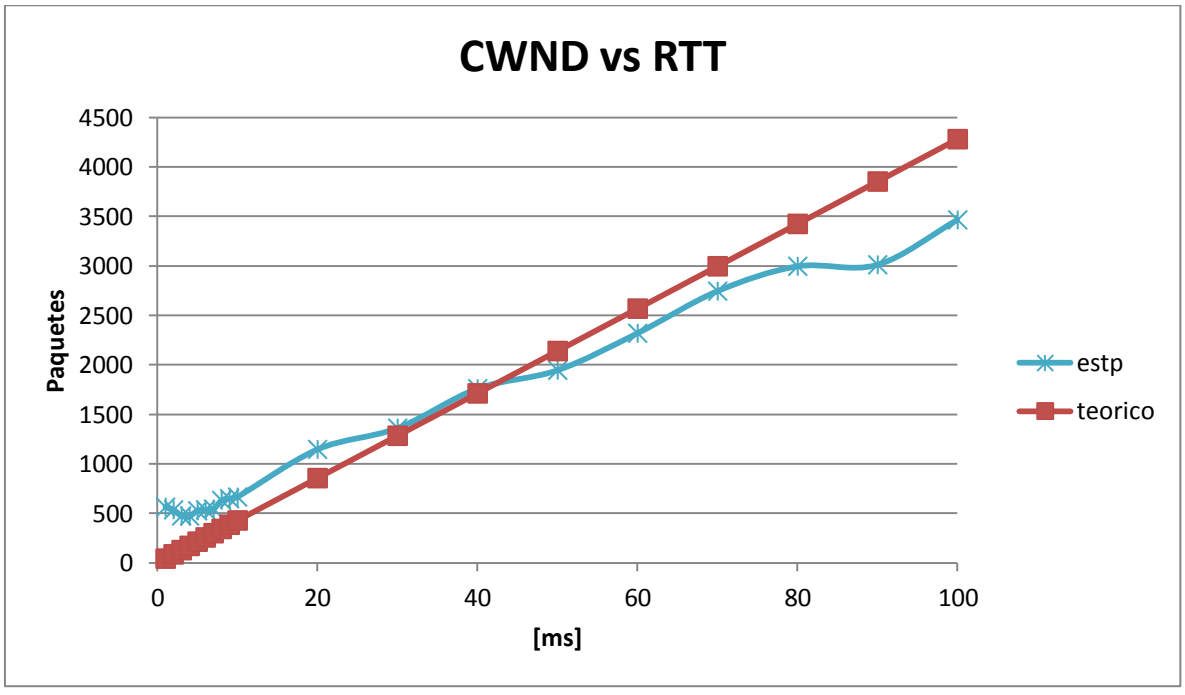


Figura 10. Gráfico CWND vs RTT

La ventana de congestión mínima del protocolo es calculada dinámicamente según la siguiente ecuación:

$$CWND_{min} = \frac{CIR[Mbits/s] \cdot RTT[us]}{MSS[bits]} \quad (15)$$

Dado que tanto MSS y CIR son valores fijos en el cálculo de la ventana, la única manera que la ventana de congestión mínima configurada sea inferior a la curva teórica, es porque el valor obtenido de RTT en el protocolo es inferior al configurado en el testbed de pruebas y por tanto este comportamiento se ve reflejado en un flujo de datos inferior al previsto.

En conjunto puede concluirse que el protocolo implementado se comporta según lo esperado teóricamente, pero para cumplir las exigencias del servicio Carrier Ethernet, debe buscarse una forma en que el RTT sea calculado de mejor manera, o bien aumentar el CIR configurado sobre un 15% para que cumpla con los estándares del servicio.

4.1.2. Comportamiento de la reducción de la ventana.

Se analizará el comportamiento de la ventana de congestión cuando existe un RTT de 10[ms]

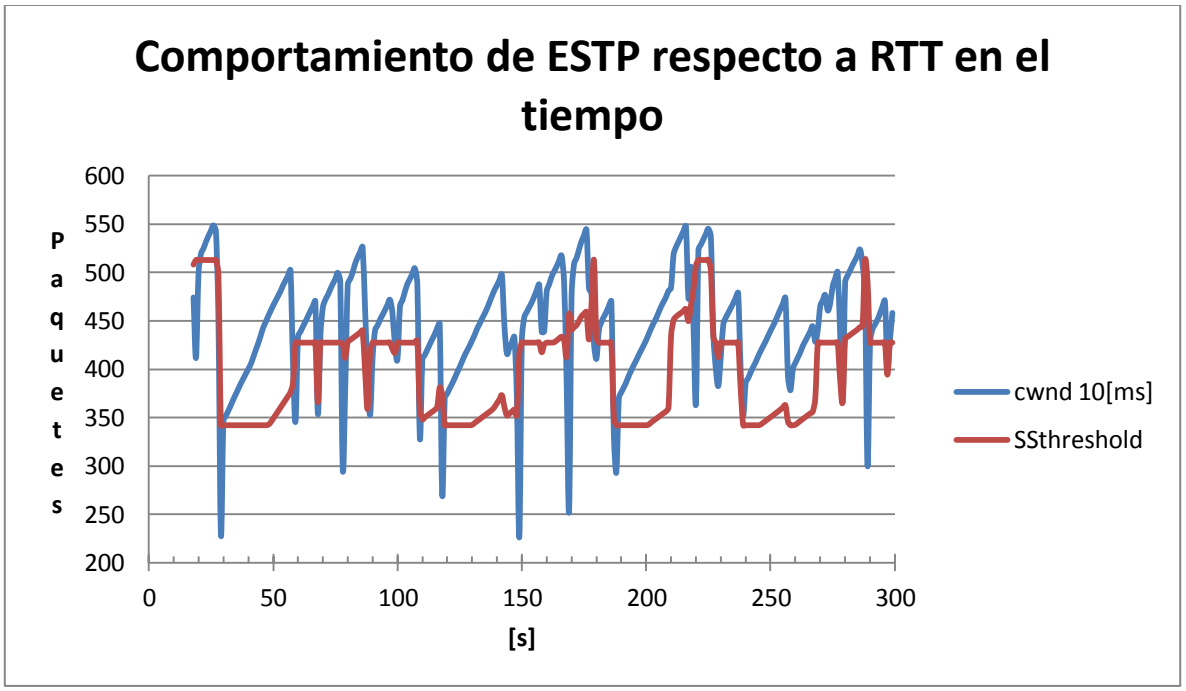


Figura 11. Comportamiento de ESTP respecto a RTT en el tiempo

En la figura se observan los promedios de la ventana de congestión calculados para cada segundo de las pruebas realizadas. Si bien esto corresponde a un promedio y se está omitiendo información dinámica debido a la granularidad del tiempo de cálculo, es posible analizar a grandes rasgos el comportamiento del cálculo de la ventana de congestión. Se centra la atención en el comportamiento observado entre los 110 a 150 segundos.

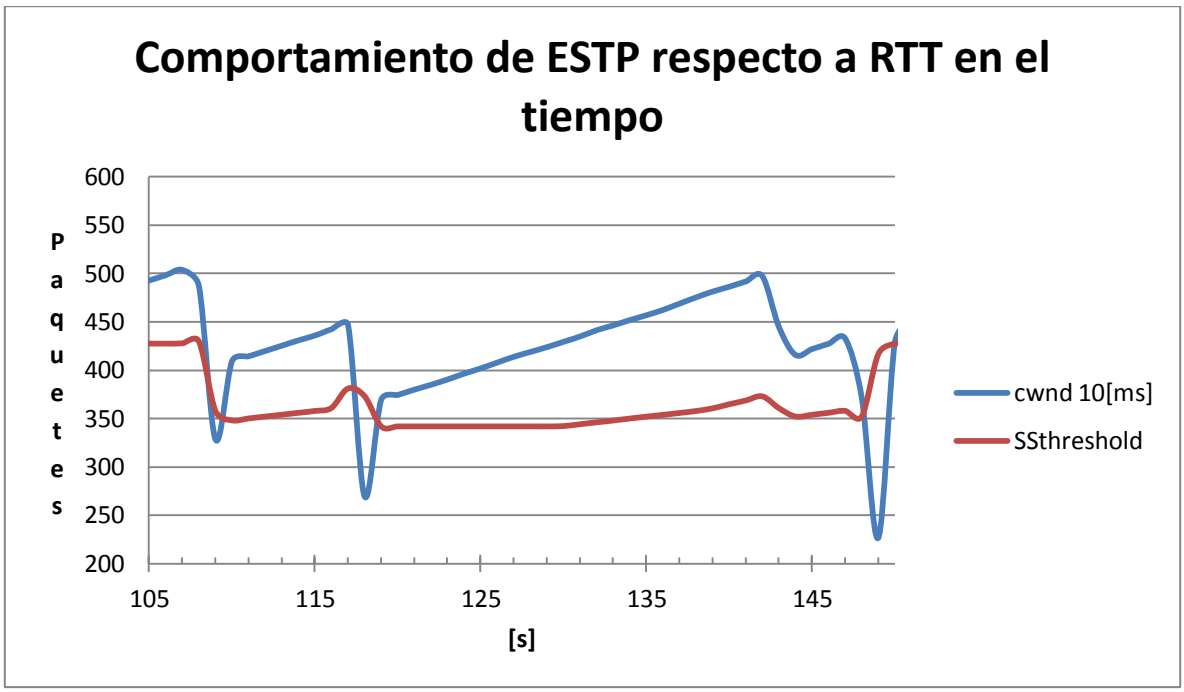


Figura 12. Gráfico Comportamiento de ESTP respecto a RTT en el tiempo

En el gráfico se considerará que la primera pérdida ocurre a los 117 [s] luego de 10 segundos sin pérdida, donde la ventana es reducida en 173 paquetes. luego desde los 120 segundos no existe pérdida y la ventana de congestión crece linealmente. En el segundo 141, se presenta un évento de pérdida y la ventana es reducida ligeramente en 78 paquetes. Posteriormente, 4 segundos más tarde (146[s]) ocurre un nuevo evento de pérdida y la ventana es reducida mucho más drásticamente en 207 paquetes. En el primer evento de pérdida considerado (Evento 1), la ventana es reducida en un 39,8%. Se observa, que en el segundo evento de pérdida (Evento 2) la ventana es reducida en un 15,6%, y finalmente en el tercer evento (Evento 3) la ventana de congestión es reducida en un 47,8%.

En la siguiente tabla se observan estos tres valores ordenados dependiendo del tiempo entre ocurrencias de pérdidas.

Evento	Tiempo entre pérdidas [s]	%Reduccion ventana
3	4	47,8%
1	10	39,8%
2	21	15,6%

Tabla 3. Eventos según ocurrencias de perdidas

Debido a que el tiempo entre pérdidas tiene directa relación, con la cantidad de paquetes enviados en ese transcurso de tiempo, la tabla anterior permite deducir que el comportamiento de la ventana de reducción, es acorde al comportamiento de la función $map(\alpha)$. Luego, en la medida que existe una mayor distancia de paquetes entre pérdidas, la ventana de congestión es reducida ligeramente, mientras que si la distancia entre pérdida de paquetes es baja la ventana es reducida en forma más agresiva.

En conclusión el método de reducción de la ventana de congestión se comporta dentro de lo esperado según el modelo teórico.

4.1.3. Reducción del efecto de cuello de botella

En esta sección se presenta el gráfico resumen que caracteriza el flujo de datos v/s el nivel de RTT para ESTP.

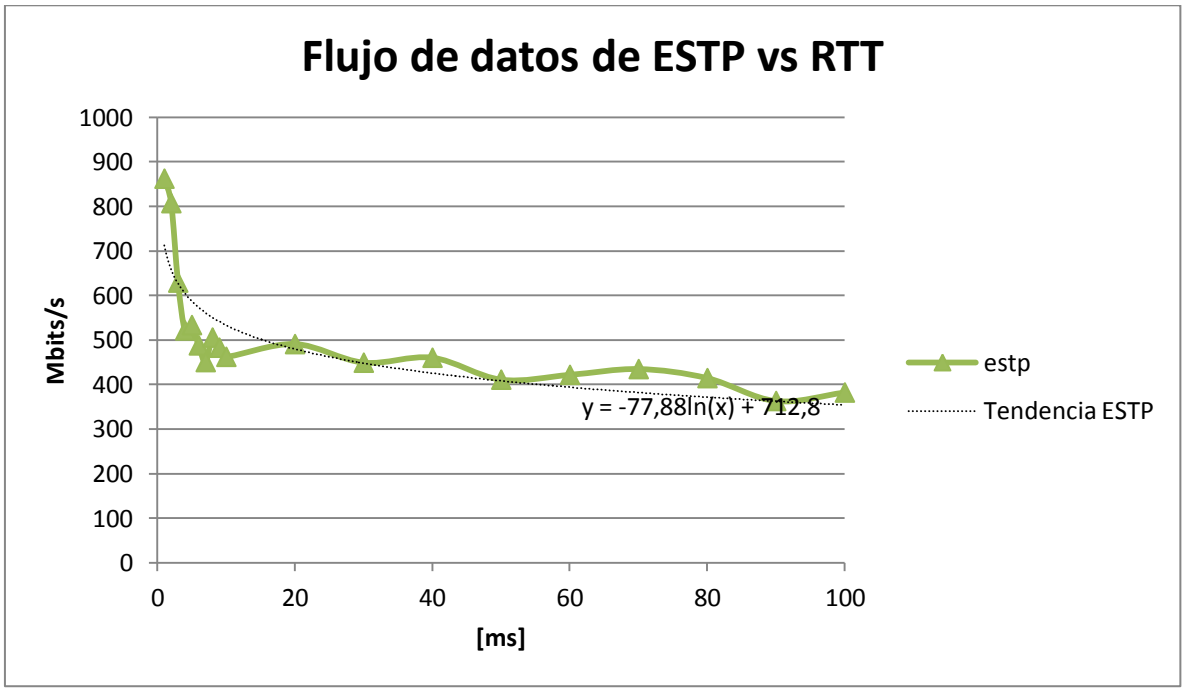


Figura 13. Gráfico Flujo de datos de ESTP vs RTT

La figura anterior muestra que en la medida que RTT aumenta, la reducción de la velocidad de transferencia de datos no disminuye en forma drástica, y tiende hacia un valor constante. Por tanto, aun cuando RTT siga en aumento, el comportamiento de ESTP no debería variar presentando un buen comportamiento, reduciendo de esta manera el cuello de botella debido a las distancias entre emisor y receptor.

Considerando los datos anteriormente expuestos, se puede afirmar que las pruebas experimentales del protocolo ESTP, se condicen con la teoría de la que emana.

4.2. Comparación de los resultados de ESTP con métodos de congestión tradicionales.

En este apartado se muestrán en primera instancia los resultados obtenidos para cada uno de los protocolos (ESTP, BIC, CUBIC, RENO) en relación a su desempeño en el tiempo a RTT fijo con el objetivo de observar su comportamiento. Luego se efectúa la comparación de los resultados obtenidos en cuanto a ventana de congestión, umbral de Slow Start y flujo de datos, en relación a la variación de RTT.

4.2.1. Resultados por protocolo

Con los datos obtenidos, se tienen las curvas para cada RTT en el tiempo. Debido a que son 19 curvas, no es posible realizar un análisis gráfico que muestre claramente su comportamiento.

A raíz de lo anterior se definen tres niveles de RTT, bajo, medio y alto, correspondientes a 10[ms], 50[ms] y 100[ms] respectivamente para efectuar la comparación.

4.2.1.1. ESTP

A continuación se presenta el comportamiento de ESTP para cada nivel de RTT.

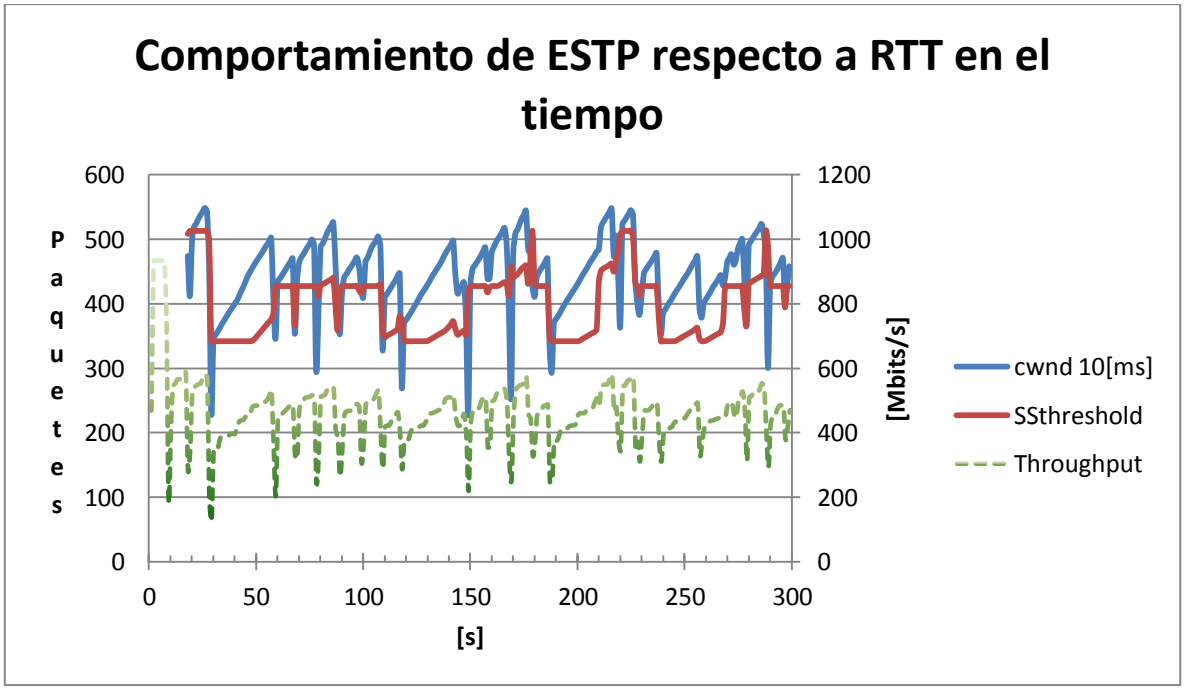


Figura 14. Gráfico Comportamiento de ESTP respecto a RTT en el tiempo

Se observa en el gráfico anterior, el distinto comportamiento (slow start y congestion avoidance), en el momento en que se alcanza el umbral SSThreshold. Se observa claramente el comportamiento lineal una vez que se ha superado el umbral, lo que concuerda con lo esperado

según la codificación y el modelo teórico. Por otra parte se observa una reducción de ventana de congestión más agresiva si es que ha ocurrido una pérdida cercana.

Este comportamiento es el esperado para ESTP, lo que indica que el protocolo está detectando el nivel de pérdida y reduciendo la ventana acorde a su medición.

A continuación se observa el comportamiento a 50 [ms].

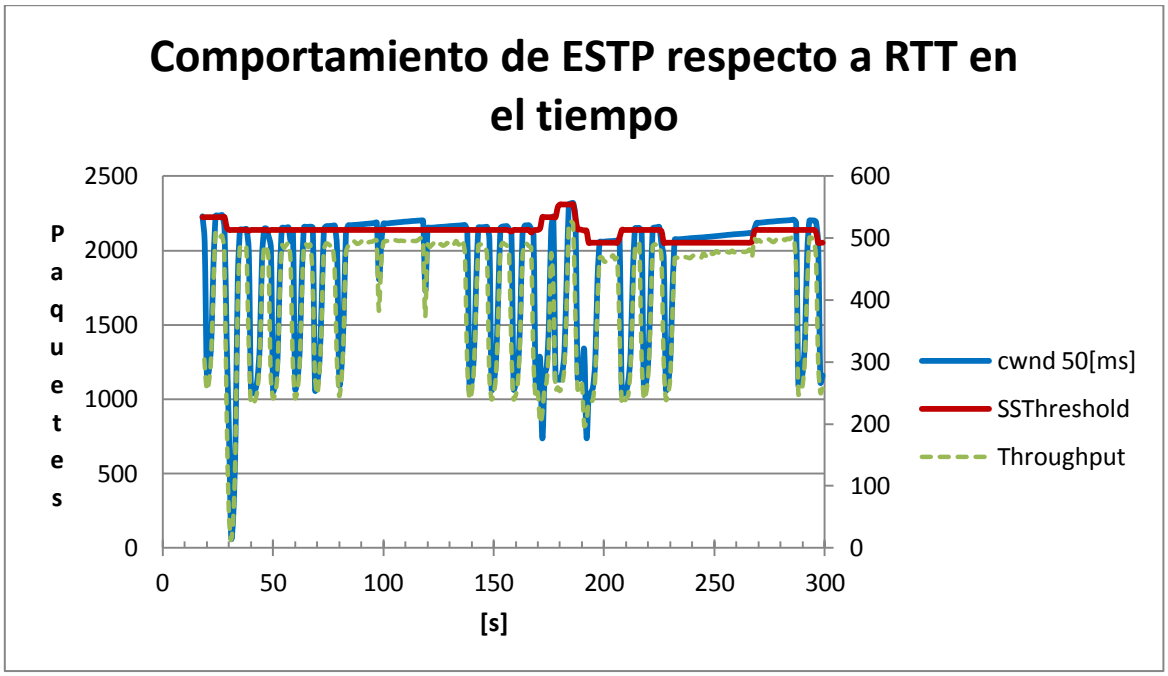


Figura 15. Gráfico Comportamiento de ESTP respecto a RTT en el tiempo

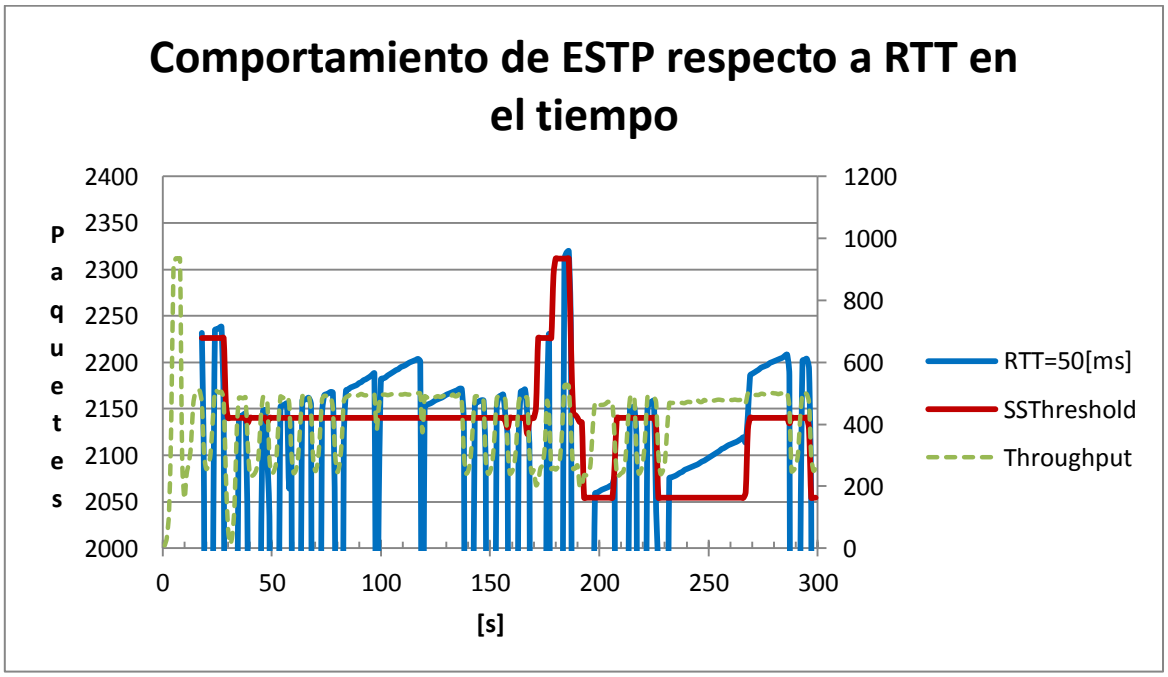


Figura 16. Gráfico Comportamiento de ESTP respecto a RTT en el tiempo

Se observa que el flujo de datos no sufre mayor variación y sigue tendiendo a los 500[Mbits/s].

A continuación se muestra el caso con 100[ms] de RTT, donde se puede apreciar un comportamiento del flujo de datos muy similar.

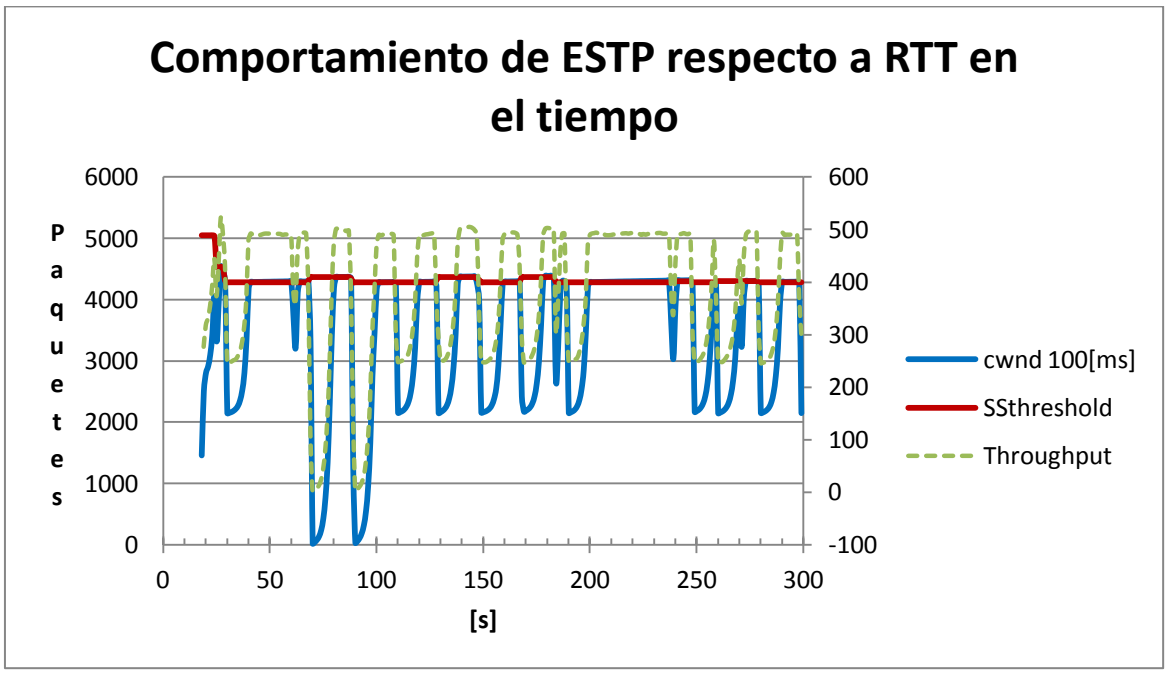


Figura 17. Gráfico Comportamiento de ESTP respecto a RTT en el tiempo

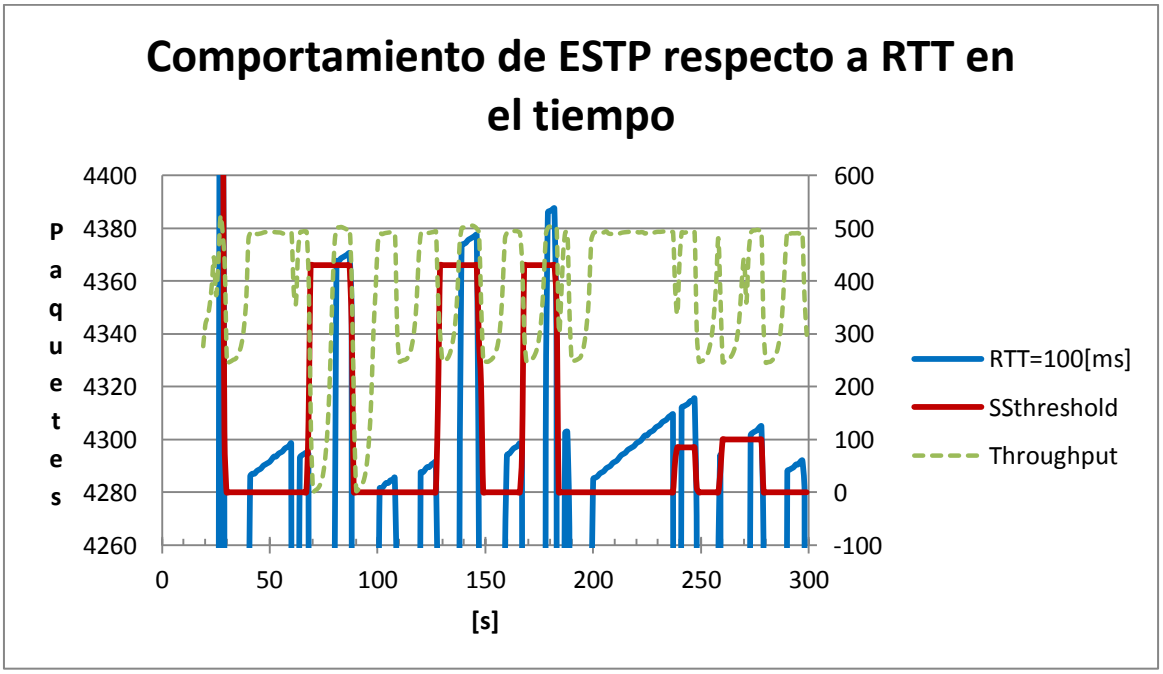


Figura 18. Gráfico Comportamiento de ESTP respecto a RTT en el tiempo

En las figuras anteriores, se observa claramente como la curva de throughput tiene una relación directa con la ventana de congestión. Esta relación proporcional es natural, debido a que la ventana de congestión es simplemente un controlador de cuantos paquetes de datos serán enviados a través de la red. El throughput puede representarse como la conjunción entre el comportamiento de Slow Start y congestion avoidance, por lo que la caracterización del funcionamiento del protocolo puede analizarse en base a esta única variable.

En la siguiente figura se muestra el flujo de datos obtenido para cada segundo en los tres niveles de RTT.

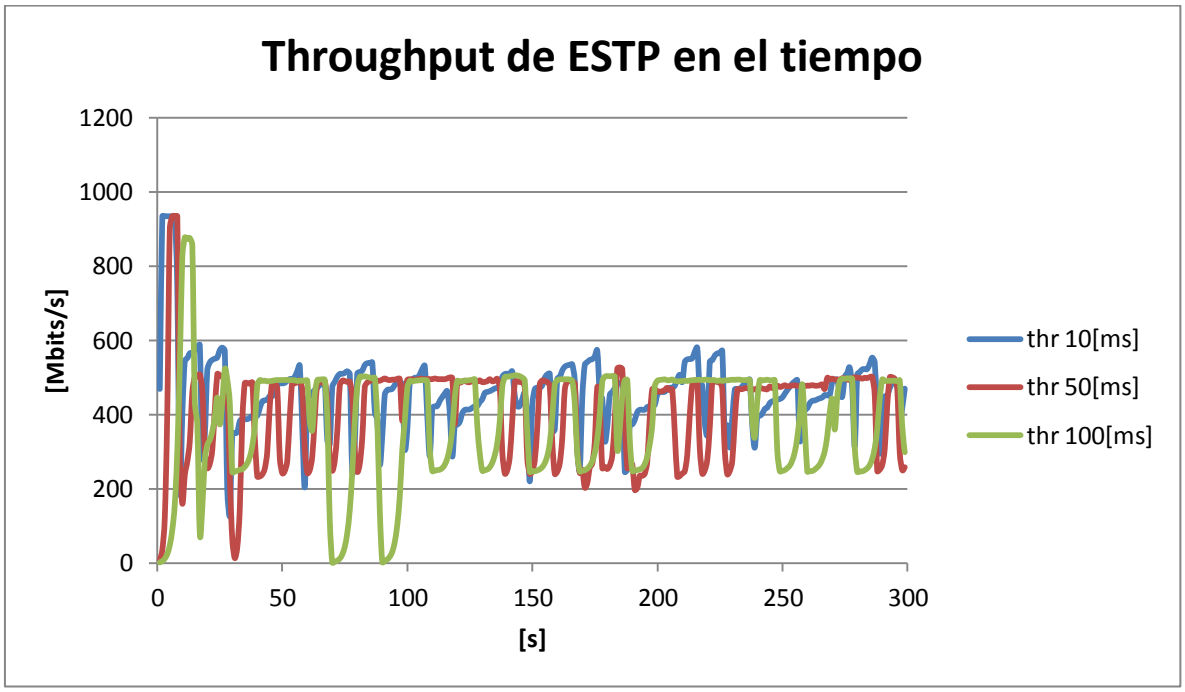


Figura 19. Gráfico Throughput de ESTP en el tiempo

Se puede observar que el flujo de datos tiende a mantener una velocidad de transferencia cercana a los 500[Mbits/s], que corresponde al CIR configurado en el protocolo. Sin embargo también se ve que este valor, en la medida que el RTT aumenta, resulta ser una cota superior para el caudal de datos, por lo tanto se observa un problema en la tasa de datos garantizada. Este problema puede ser resuelto para cumplir con la calidad de servicio, aumentando el valor de CIR a una tasa mayor.

4.2.1.2. RENO

En los siguientes gráficos se presenta el comportamiento de reno para los tres niveles de RTT definidos.

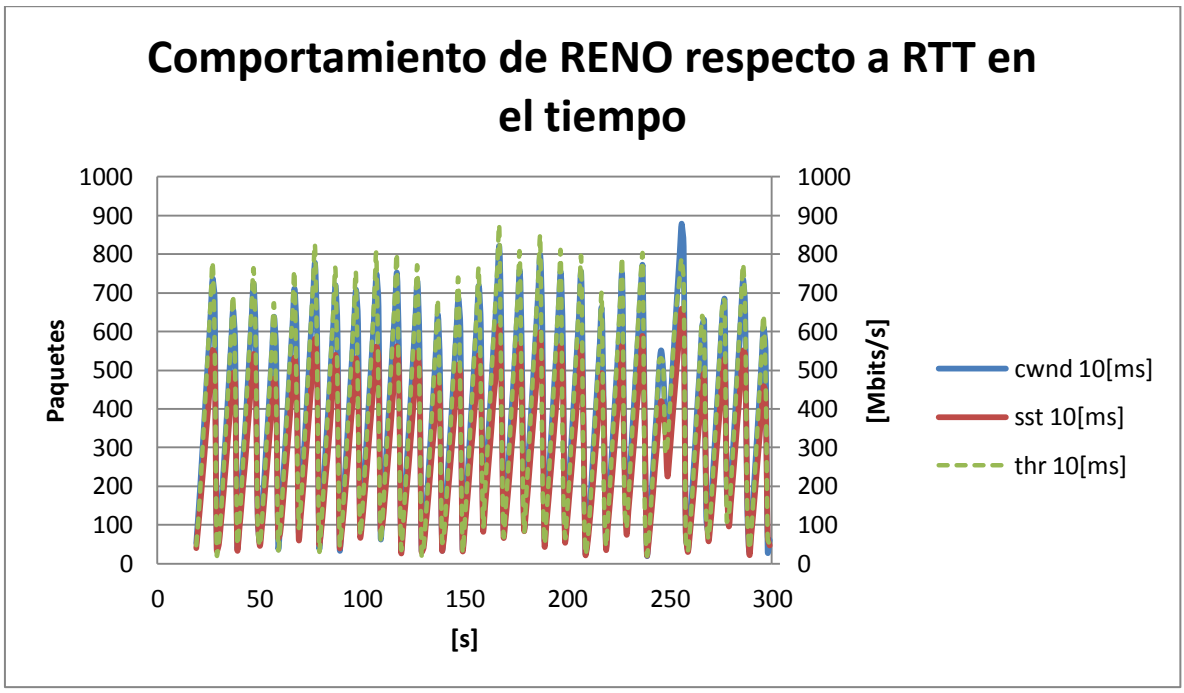


Figura 20. Gráfico Comportamiento de RENO respecto a RTT en el tiempo

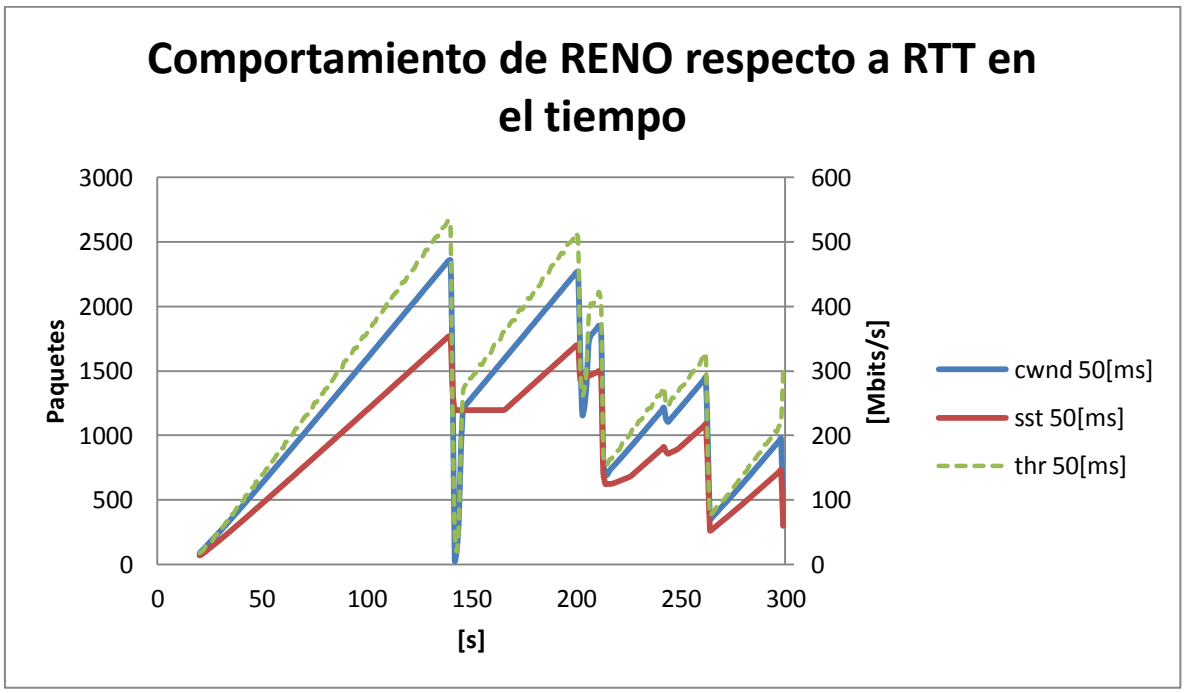


Figura 21. Gráfico Comportamiento de RENO respecto a RTT en el tiempo

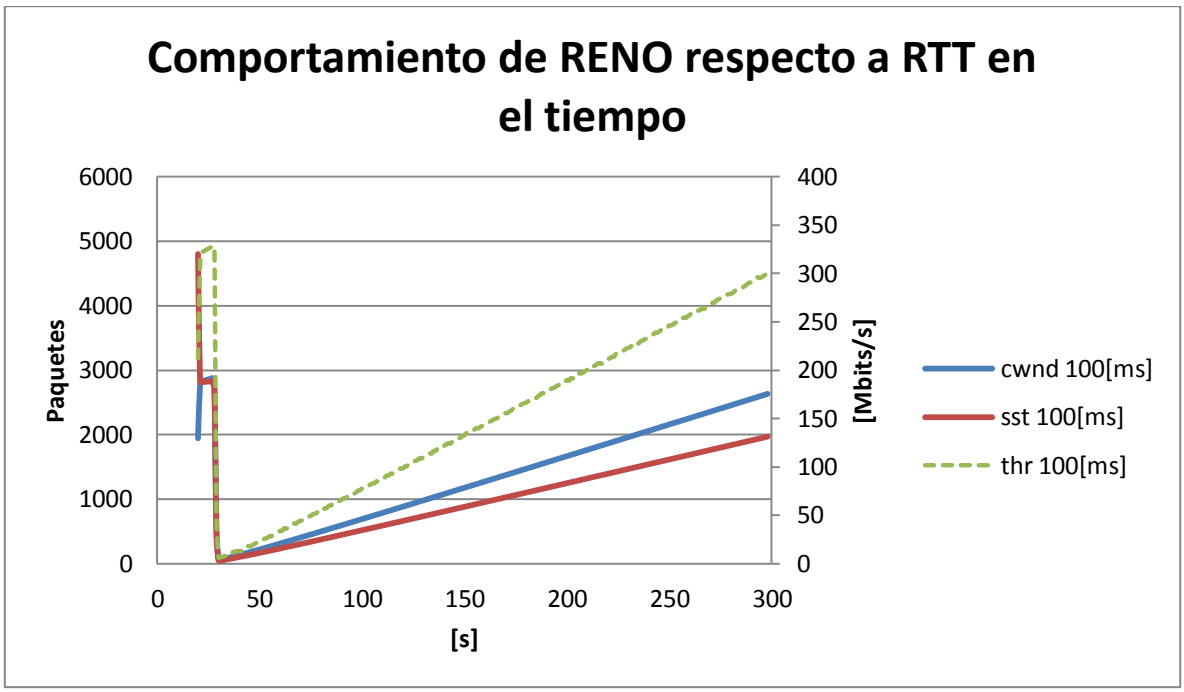


Figura 22. Gráfico Comportamiento de RENO respecto a RTT en el tiempo

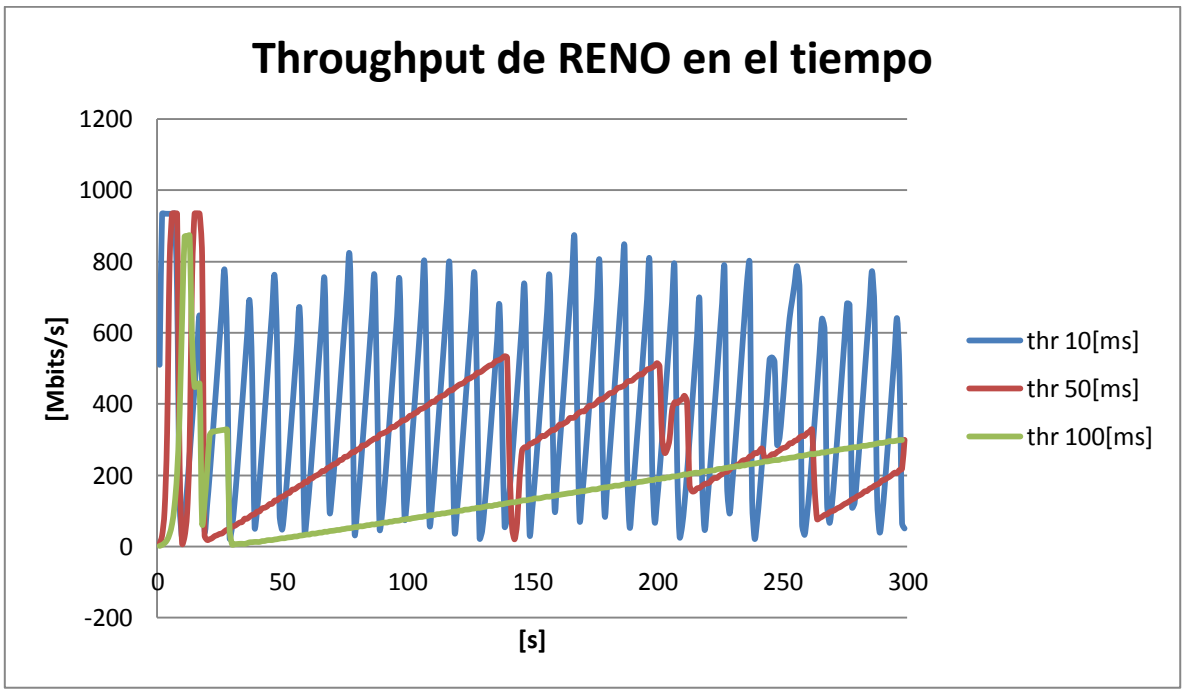


Figura 23. Gráfico Throughput de RENO en el tiempo

El comportamiento clásico de diente de sierra se muestra claramente a un RTT de 50[ms]. Se ve como la ventana de congestión es reducida a la mitad en cada evento de pérdida. Además se observa claramente como el througput de datos es afectado en la medida que aumenta el RTT. En consecuencia, se observa el comortamiento clásico y por tanto esperado de RENO.

4.2.1.3. CUBIC

A continuación se presentan los gráficos obtenidos para el protocolo CUBIC en los ditintos niveles de RTT definidos.

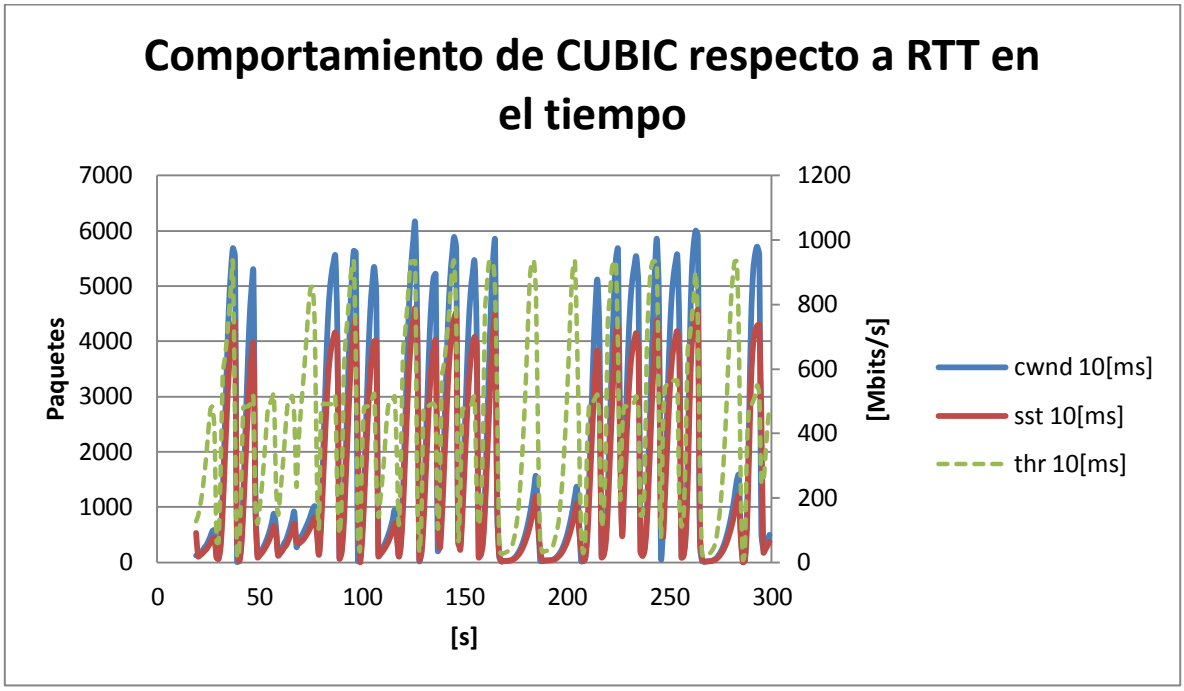


Figura 24. Gráfico Comportamiento de CUBIC respecto a RTT en el tiempo

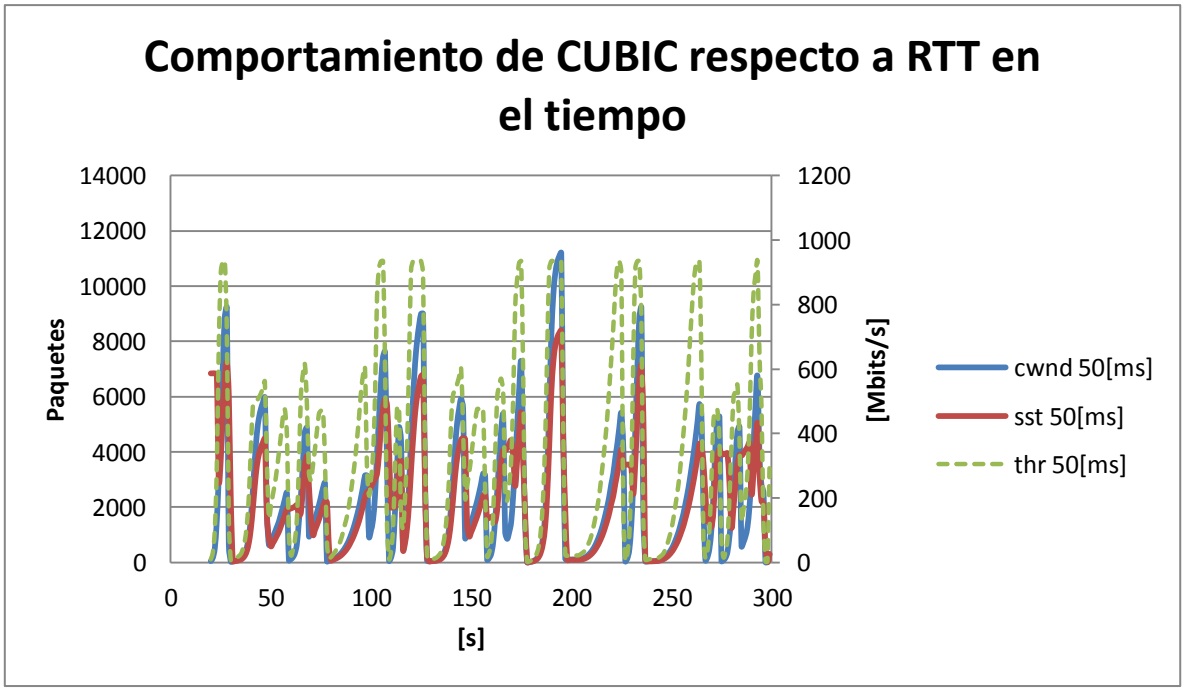


Figura 25. Comportamiento de CUBIC respecto a RTT en el tiempo

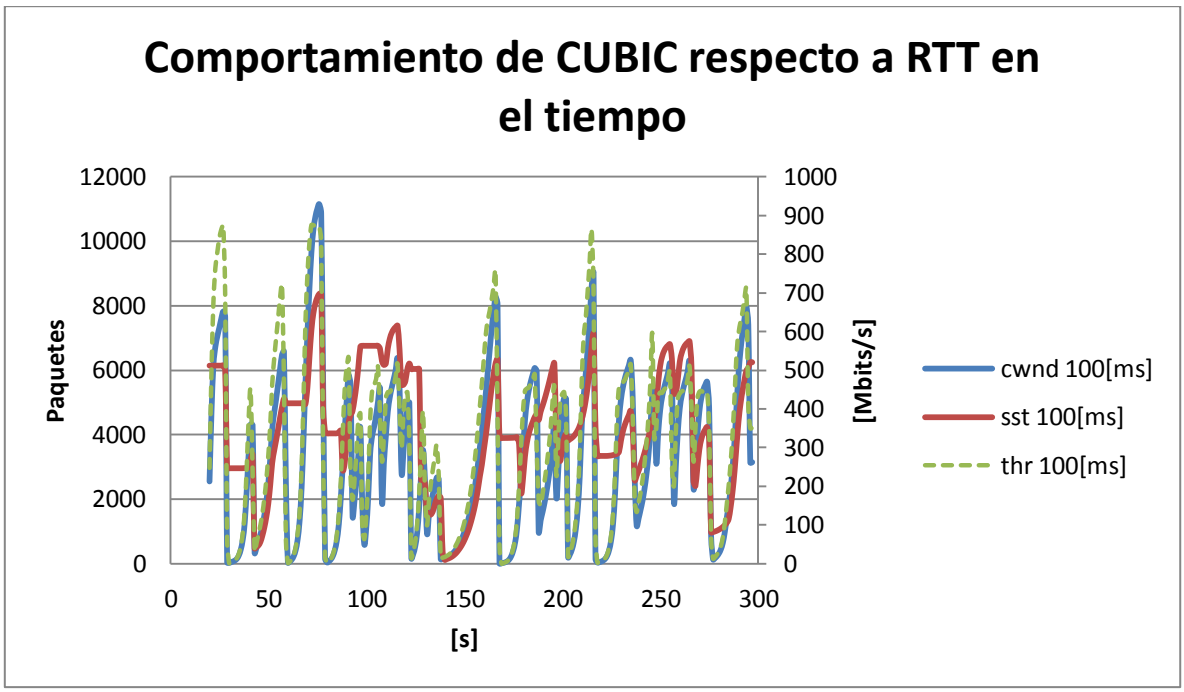


Figura 26. Gráfico Comportamiento de CUBIC respecto a RTT en el tiempo

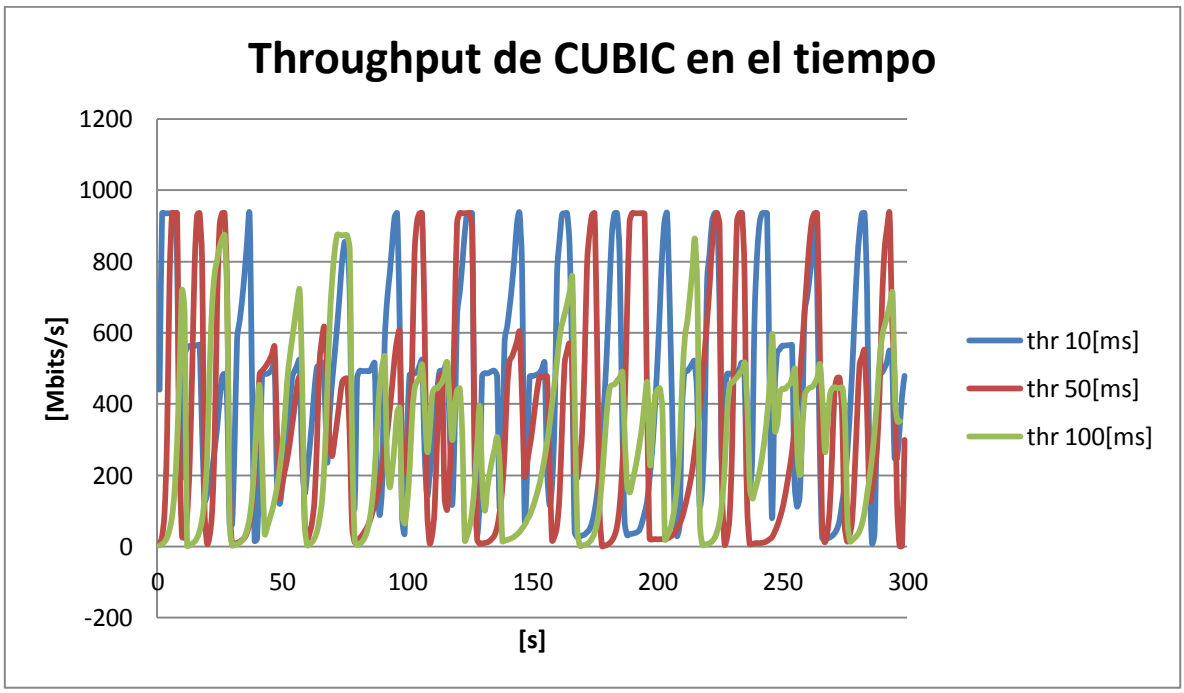


Figura 27. Gráfico Throughput de CUBIC en el tiempo

En los gráficos observados, se aprecia el comportamiento cúbico del cálculo de la ventana de congestión. Se observa que el protocolo disminuye la ventana dependiendo del tiempo existente entre pérdidas. Con respecto al througput, es claro que tiene un comportamiento mejor que reno, en la medida que aumenta RTT, sin embargo las amplitudes de oscilación, son demasiado altas, lo que reduce la eficacia del comportamiento. En particular, se observa un

comportamiento en base a ráfagas de datos las que son amortiguadas fuertemente (de 900[Mbits/s] a 0[Mbits/s]) para comportarse de manera justa.

4.2.1.4. BIC

A continuación se presentan las curvas en el tiempo para el protocolo BIC, en los distintos niveles de RTT definidos.

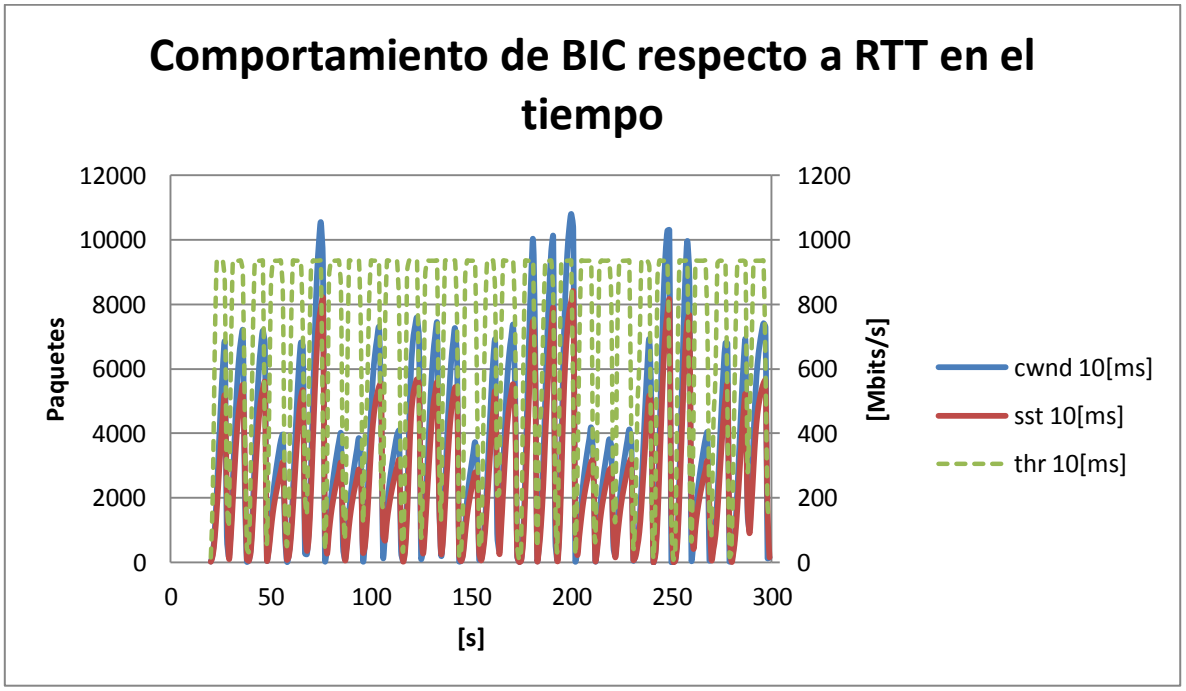


Figura 28. Gráfico Comportamiento de BIC respecto a RTT en el tiempo

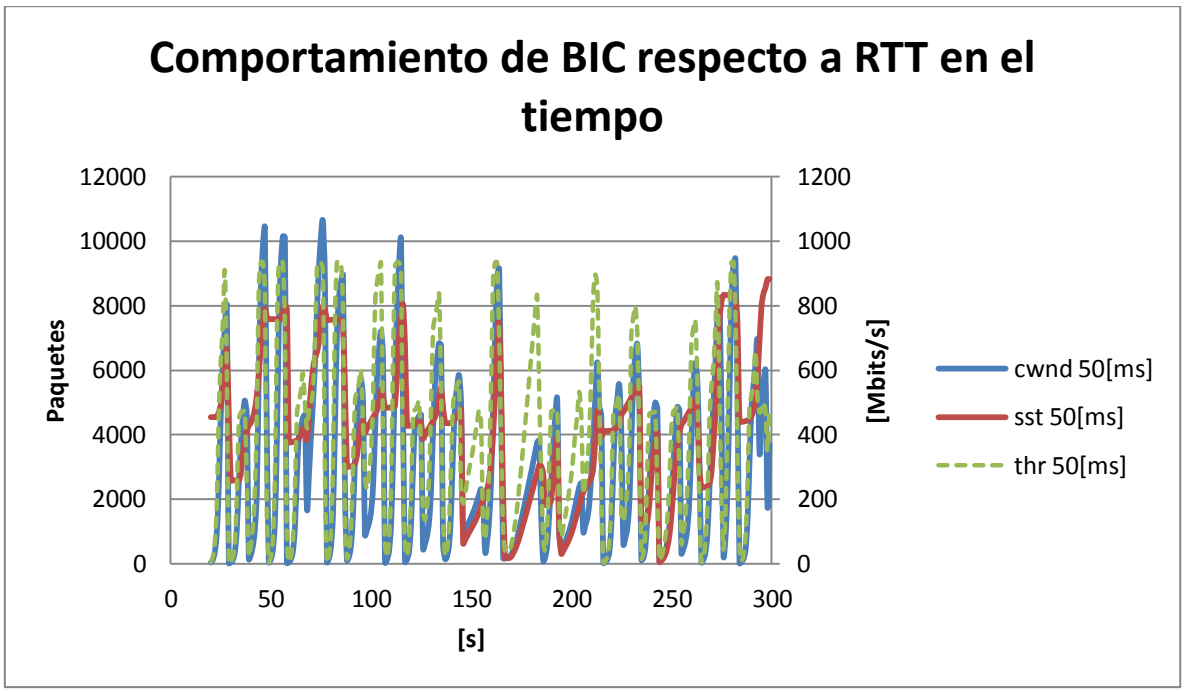


Figura 29. Gráfico Comportamiento de BIC respecto a RTT en el tiempo

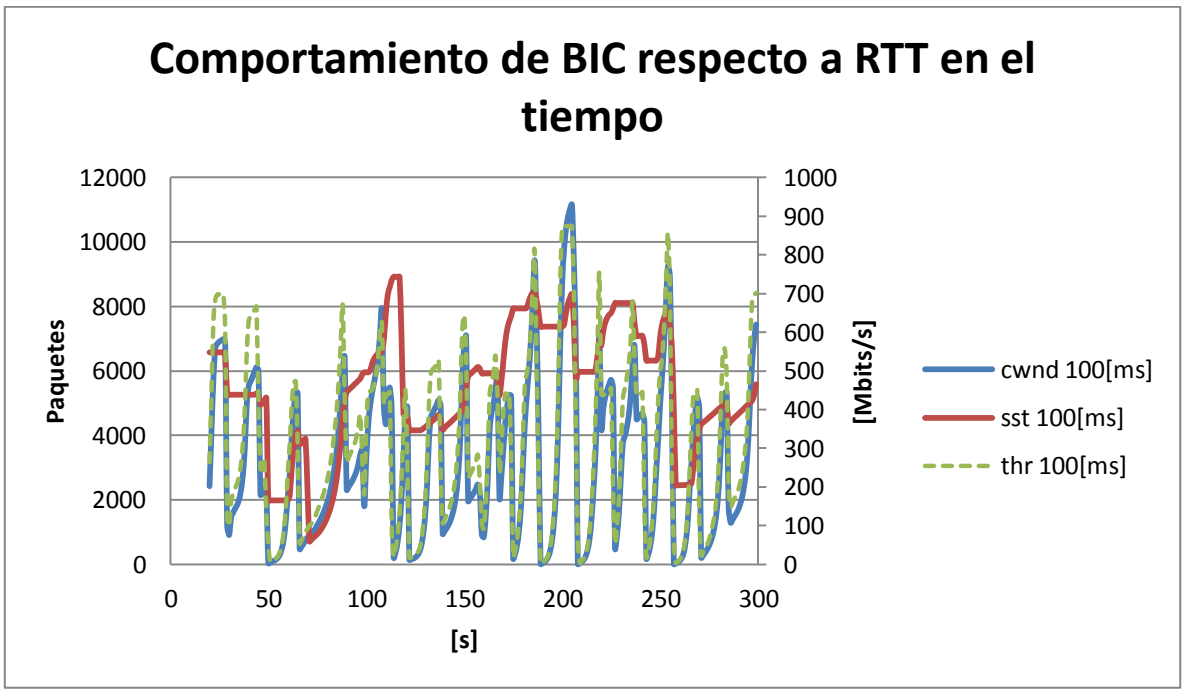


Figura 30. Gráfico Comportamiento de BIC respecto a RTT en el tiempo

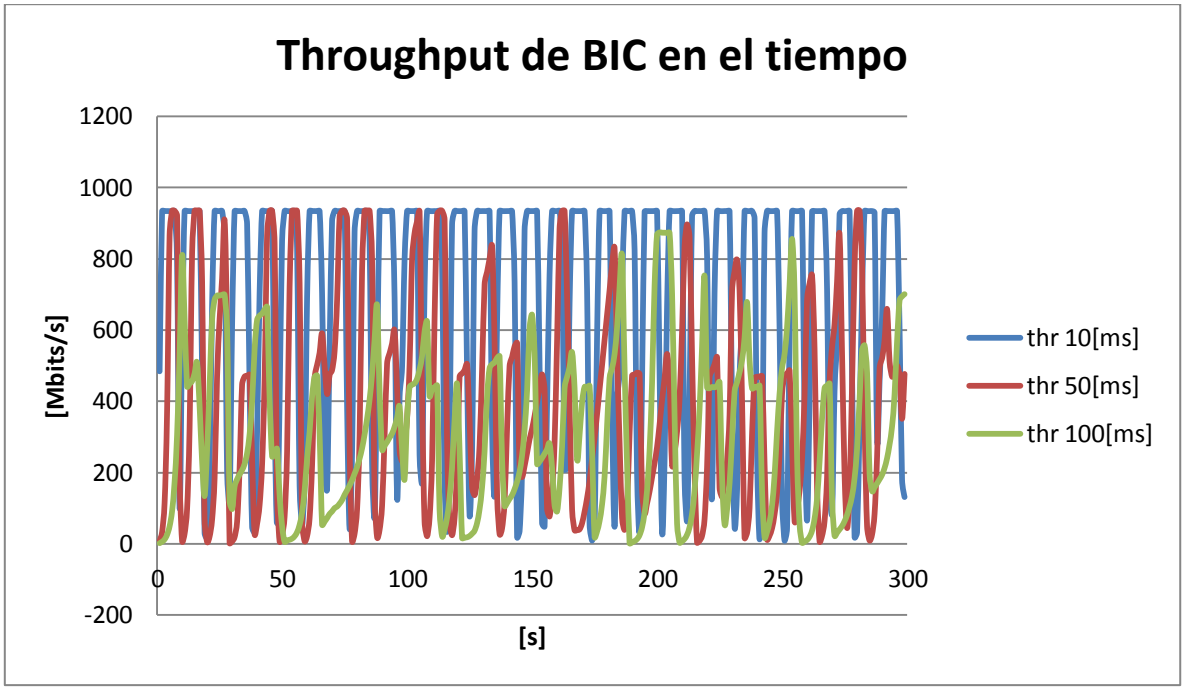


Figura 31. Gráfico Throughput de BIC en el tiempo

En los gráficos se observa un comportamiento agresivo cuando el RTT es bajo. En teoría, BIC posee un algoritmo de búsqueda binaria, donde la ventana de congestión crece al valor medio entre la última ventana de congestión donde ha ocurrido una pérdida de paquetes y la última ventana de congestión que no ha tenido una pérdida en el periodo de RTT. De esta manera prueba el ancho de banda, alcanzando altos niveles de Throughput, sin embargo no se comporta de manera justa frente a otros protocolos. El comportamiento se observa parcialmente en los gráficos de throughput. Por último se observa un rendimiento similar con respecto a la variación de RTT.

4.2.2. COMPARACIÓN DE PROTOCOLOS

Anteriormente se han presentado las curvas en el tiempo de los distintos protocolos a distintos niveles de RTT. Todas las curvas obtenidas pueden apreciarse en la sección Anexos.

Cabe destacar que independiente de la forma en que la ventana de congestión es calculada, el resultado final del protocolo, tiene relación con la cantidad de datos que han sido entregados satisfactoriamente en el tiempo. Por lo tanto, la cantidad de datos entregados corresponde al producto final que, representado a través del flujo de datos muestra el desempeño general del protocolo en un intervalo de tiempo.

Debido a lo anterior se calcula el promedio total del throughput en el tiempo para cada RTT y se representa en un gráfico. Los resultados obtenidos muestran las siguientes curvas características.

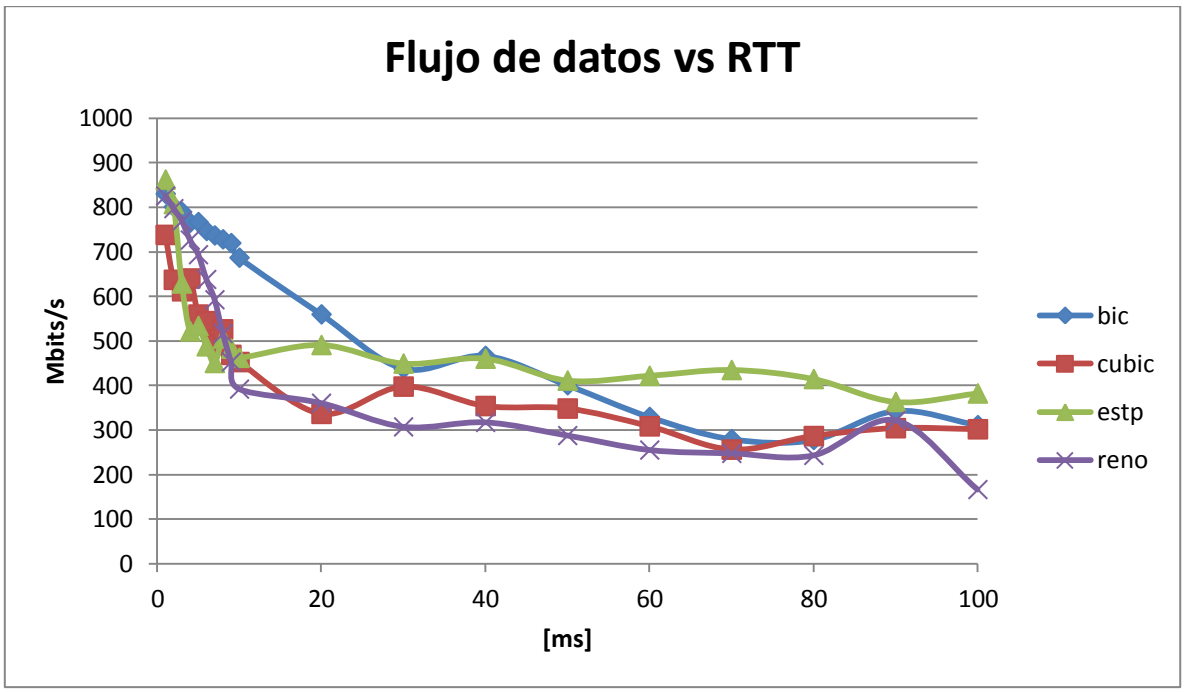


Figura 32. Gráfico Flujo de datos vs RTT

Se observa que ESTP no presenta un gran desempeño cuando el RTT es bajo. Entre los 40 y 50 [ms] de RTT, presenta un comportamiento similar al de cubic. Luego entre los 50 y 100 [ms] de RTT, muestra un throughput más alto que el resto de los protocolos de evitación de tráfico.

En los siguientes gráficos, se presenta la diferencia porcentual del flujo de datos de BIC, CUBIC y RENO respecto al throughput de ESTP a variaciones de RTT.

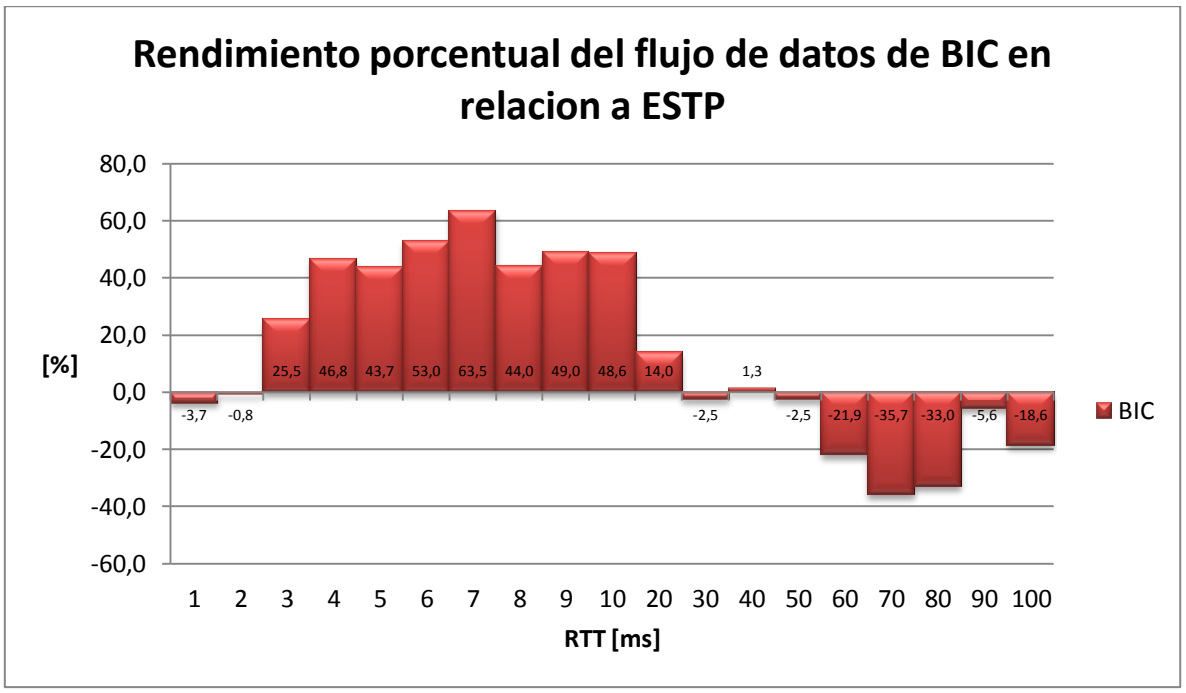


Figura 33. Gráfico Rendimiento porcentual del flujo de datos de BIC en relacion a ESTP

Se observa que a RTTs bajo los 10[ms], BIC alcanza un máximo de 63,5% sobre el flujo de datos de ESTP. En este intervalo se observa la superioridad de BIC sobre ESTP a bajos niveles de delay. A partir de un RTT de 60[ms] ESTP supera a BIC, se observa una máxima diferencia a un RTT de 70[ms] de 37,5%.

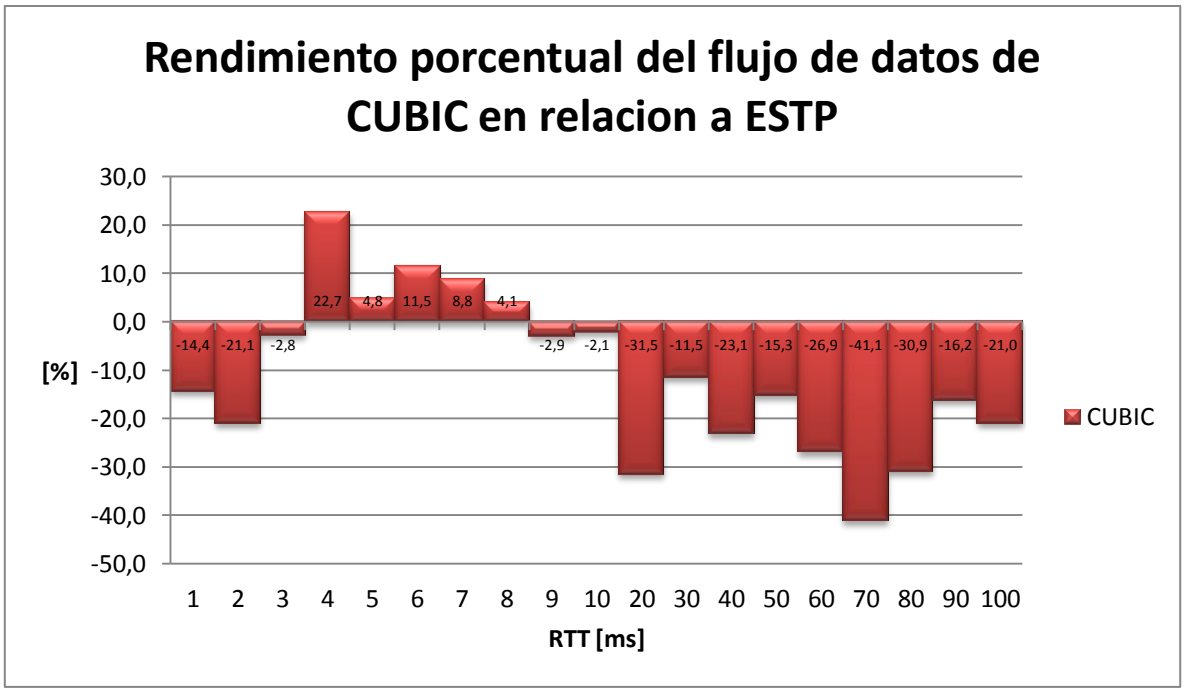


Figura 34. Gráfico Rendimiento porcentual del flujo de datos de CUBIC en relacion a ESTP

Se observa que CUBIC, presenta solo un flujo de datos superior en el intervalo de RTT entre 4 y 8[ms]. En la medida que el RTT aumenta se observa claramente que ESTP se comporta mejor que CUBIC. A un RTT de 70[ms] se ve una diferencia máxima, en la que CUBIC es inferior en 41,1%.

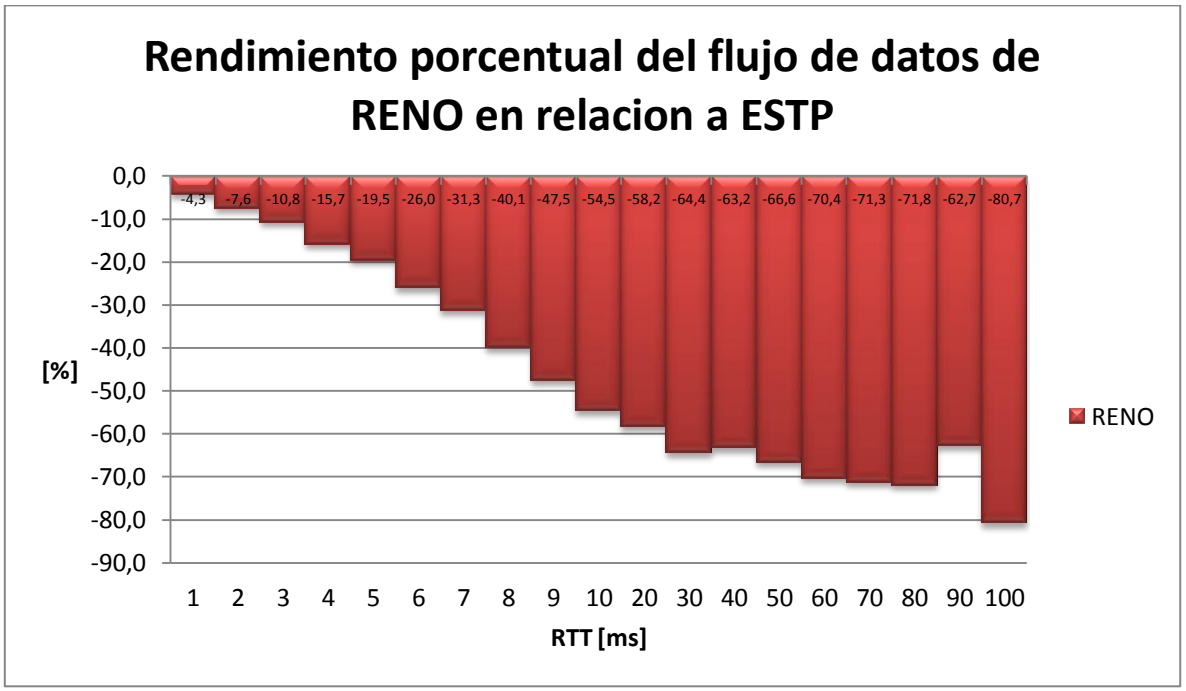


Figura 35. Gráfico Rendimiento porcentual del flujo de datos de RENO en relacion a ESTP

En el gráfico anterior se observa claramente una superioridad del comportamiento de ESTP sobre Reno. En la medida que aumenta RTT, la diferencia porcentual aumenta. La máxima diferencia porcentual se observa a un RTT de 100[ms] donde RENO es inferior en un 80,7%.

De los datos antes analizados, se concluye que en la medida que el RTT aumenta, ESTP presenta un mejor comportamiento que BIC, CUBIC y RENO.

CAPÍTULO 5: CONCLUSIONES

El desarrollo de esta memoria, ha logrado dar respuesta al problema de investigación planteado inicialmente: ¿La implementación de ESTP valida el modelo teórico presentando un mejor comportamiento que los métodos de congestión tradicionales? El problema se desarrollo, a través de la codificación de ESTP, lo cual permitió su implementación y posteriores pruebas. Para dar respuesta a la pregunta inicial se generaron dos hipótesis de investigación, obteniendo los resultados explicados a continuación.

Para la primera de las hipótesis: "Los datos experimentales de ESTP validan el modelo teórico del protocolo", se concluye de los datos experimentales que la ventana de congestión es efectivamente reducida de acuerdo a la función $map(\alpha)$, el flujo de datos tiende a un valor cercano a la tasa garantizada CIR y que mientras más alto es el valor de RTT, el flujo de datos de ESTP presenta un comportamiento que tiende a un valor constante cercano a CIR. Se concluye que el valor del flujo de datos fue inferior a CIR, debido a variaciones de la obtención del valor de RTT respecto a lo esperado teóricamente. Se propone como solución alternativa mejorar la calidad en que la estimación de RTT es realizada dentro del protocolo o bien aumentar el valor de CIR en un 15% para cumplir con las condiciones de Carrier-Ethernet.

Para la segunda hipótesis: "Al aumentar el Round Trip Time, ESTP tiene un mejor comportamiento que BIC, CUBIC y RENO", se concluye de los datos experimentales que, el rendimiento de ESTP supera a todos los demás protocolos a partir de un RTT de 60[ms]. Se observa que BIC presenta el segundo mejor comportamiento a valores de RTT altos, sin embargo este no es un protocolo que se comporte de manera justa (TCP-Friendly), como en teoría se comporta ESTP. Así, se observa que a un valor de RTT de 70[ms], en un ambiente de pérdida variable, el throughput de ESTP es superior a BIC en un 35,7% , a CUBIC en un 41,1% y a RENO en un 71,8%.

En consecuencia, el trabajo realizado permite corroborar el modelo teórico de ESTP y su buen desempeño, en cuanto a velocidad de transmisión cuando los valores de RTT aumentan, en comparación a los protocolos BIC, CUBIC y RENO.

Como investigaciones futuras se sugiere probar el comportamiento de ESTP cuando compite por el uso de ancho de banda de transmisión, con el fin de probar que el protocolo se comporta de manera justa frente a otros protocolos de evitación de tráfico. Además, se sugiere mejorar la estimación de RTT del protocolo para observar directamente el cumplimiento de la tasa mínima de transmisión CIR. Por otro lado sería interesante probar el protocolo en redes congestionadas reales de alto rendimiento, donde RTT esté dado por la separación espacial entre cliente y servidor, esquema que se encontró fuera de alcance, para la realización de pruebas en este trabajo.

CAPÍTULO 6: REFERENCIAS BIBLIOGRÁFICAS

[1] B. W. Kernighan y D. M. Ritchie, El lenguaje de Programación C.Segunda Edición, Prentice-Hall, 1991.

[2] Estevez, Angulo, Abujatum, Ellinas, Ceng Liu, Chang. *A Carrier-Ethernet Oriented Transport Protocol with a Novel Congestion Control and QoS Integration: Analytical, Simulated and Experimental Validation*, ICC 2012.

[3] Estevez, Angulo, Ehijo, Ellinas, Chang. *Ethernet-Services Transport Protocol Design oriented to Carrier Ethernet Networks*.GLOBECOM 2012.

[4] Estevez, Angulo, Ellinas, Chang. *Ethernet-Services Transport Protocol forCarrier Ethernet Networks*, ICCCN 2012.

[5] Hernandez Sampieri, Fernandez Collado y Baptista Lucio, *Metodología de la Investigación*, McGraw-Hill Interamericana, 2006.

[6] http://linux.die.net/man/7/socket [Último acceso: 04 08 2013].

[7] http://linux.die.net/man/7/TCP [Último acceso: 04 08 2013].

[8] http://www.ijcaonline.org/volume16/number1/pxc3872647.pd [Último acceso: 04 08 2013].

[9] http://www.linuxcertif.com/man/7/TCP/es/ [Último acceso: 04 08 2013].

[10] Information Sciences Institute, University of Southern California, «Transmission Control Protocol,» Information Processing Techniques Office, Defense Advanced Reserach Projects Agency, Marina del Rey, California, 1981.

[11] Internet Society, [En línea]. Available: http://www.internetsociety.org/internet/internet51/history-internet/brief-history-internet#Origins. [Último acceso: 27 05 2013].

[12] J. Corbet, A. Rubini y G. Kroah-Hartman, Linux Device Drivers, Third Edition, O'Reilly Media, Inc., 2005.

[13] J. F. Kurose y R. K. W., Redes de Computadoras: Un enfoque Descendente, Pearson Educación S.A., 2010.

[14] P. Ateline y J. Dordoigne, TCP/IP y Protocolos de Internet, ENI, 2007.

[15] UCLA Engineering Computer Science, 6 04 2011. [En línea]. Available: http://www.lk.cs.ucla.edu/internet_first_words.html. [Último acceso: 27 05 2013].

ANEXOS

En este apartado se explicitaran todos aquellos datos indicados en el documento como existentes en anexos.

ANEXO A: CÓDIGO PROTOCOLO ESTP

```
//C:\Users\Sergio\Desktop\PENDRIVE\Testbed\Common\Protocol\
TCP_ESTP_test6.c
#include <linux/module.h>
#include <net/TCP.h>
#include <linux/time.h>
int cir=500;
int promedio_flag=0;
int debug=0;
int MSS=1460;
module_param(cir,int,S_IRUGO);
module_param(promedio_flag,int,S_IRUGO);
module_param(debug,int,S_IRUGO);
module_param(MSS,int,S_IRUGO);
int id_test;

static struct ESTP_estructura_datos{
        u32 valor;
}
rtts_ultimos_valores[]={{0},{0},{0},{0},{0},{0},{0},{0},{0},{0}
},
segmentacion_alphas[]={{0},{0},{0},{0},{0},{0},{0},{0},{0},{0}
,{0},{0},{0},{0},{0},{0},{0},{0},{0},{0},{0},{0},{0},{0},{0},{0
},{0},{0},{0},{0},{0},{0},{0}};

//Declaracion de funciones
char Obtener_indice_AIMD(u32 a, struct ESTP_estructura_datos
dat[33]);
void Obtener_segmentacion_alphas(struct ESTP_estructura_datos
m[33],int cir);
u32 Obtener_seq_number(struct sock*, int*);

static const struct ESTP_aimd_val {
        unsigned int md;
}AIMD_ESTP[]={
{128,/*     (1) 128/(2^8) = 0.500 */},
{124,/*     (2) 124/(2^8) = 0.484 */},
{120,/*     (3) 120/(2^8) = 0.469 */},
{116,/*     (4) 116/(2^8) = 0.453 */},
{112,/*     (5) 112/(2^8) = 0.438 */},
{108,/*     (6) 108/(2^8) = 0.422 */},
{104,/*     (7) 104/(2^8) = 0.406 */},
{100,/*     (8) 100/(2^8) = 0.391 */},
{96,/*      (9) 96/(2^8) = 0.375 */},
{92,/*      (10) 92/(2^8) = 0.359 */},
{88,/*      (11) 88/(2^8) = 0.344 */},
{84,/*      (12) 84/(2^8) = 0.328 */},
{80,/*      (13) 80/(2^8) = 0.313 */},
{76,/*      (14) 76/(2^8) = 0.297 */},
{72,/*      (15) 72/(2^8) = 0.281 */},
{68,/*      (16) 68/(2^8) = 0.266 */},
{64,/*      (17) 64/(2^8) = 0.250 */},
{60,/*      (18) 60/(2^8) = 0.234 */},
{56,/*      (19) 56/(2^8) = 0.219 */},
{52,/*      (20) 52/(2^8) = 0.203 */},
{48,/*      (21) 48/(2^8) = 0.188 */},
{44,/*      (22) 44/(2^8) = 0.172 */},
{40,/*      (23) 40/(2^8) = 0.156 */},
{36,/*      (24) 36/(2^8) = 0.141 */},
{32,/*      (25) 32/(2^8) = 0.125 */},
{28,/*      (26) 28/(2^8) = 0.109 */},
{24,/*      (27) 24/(2^8) = 0.094 */},
{20,/*      (28) 20/(2^8) = 0.078 */},
{16,/*      (29) 16/(2^8) = 0.063 */},
{12,/*      (30) 12/(2^8) = 0.047 */},
{8,/*       (31) 8/(2^8) = 0.031 */},
{4,/*       (32) 4/(2^8) = 0.016 */},
{0,/*       (33) 0/(2^8) = 0.000 */},
};

#define ESTP_AIMD_MAXARRAY_SIZE(AIMD_ESTP)

static struct ESTP {
        u32 ai;
        int id_test;
        int contador_rtts;
        u32 seq_number_anterior;
        u32 ultimo_alpha;
        int estructura_rtts_llena;
        unsigned int rtts_anterior;
        int no_hay_seq_number_flag;
} datos;

static void ESTP_init(struct sock *sk)
{
        //extern struct segmentacion_alphas;
        struct TCP_sock *tp = TCP_sk(sk);
        struct ESTP *ca = inet_csk_ca(sk);
        //struct prom_vals = inet_csk_ca(sk);
        //int i=0:
        ca->ai = 0;
        datos.contador_rtts=0;
        datos.estructura_rtts_llena=0;
        datos.no_hay_seq_number_flag=0;
        datos.seq_number_anterior=tp->snd_nxt;
        datos.rtts_anterior=0;
        datos.ultimo_alpha=0;
        Obtener_segmentacion_alphas(segmentacion_alphas,cir
);
        if(debug!=0)printk(KERN_INFO "Entrando en funcion
ESTP_init\n");
        /* Ensure the MD arithmetic works.  This is somewhat
pedantic,
         * since I don't think we will see a cwnd this large. :) */
        if(debug!=0) printk(KERN_INFO " snd_cwnd_clamp
:%u\n",tp->snd_cwnd_clamp);
        tp->snd_cwnd_clamp = min_t(u32, tp-
>snd_cwnd_clamp, 0xffffffff/128); //Elige el mÃnimo entre
snd_cwnd_clamp y un

                                        // numero de 32 bits con solo
unos
```

```
	id_test+=1;
	datos.id_test=id_test;

	printk(KERN_INFO "****************NUEVA
PRUEBA %d*********************\n",datos.id_test);
	printk(KERN_INFO "*\n*\tSe definio cir=%d\n",cir);
	printk(KERN_INFO "*\n*\tSe definio
promedio_flag=%d\n",promedio_flag);
	printk(KERN_INFO "*\tSize u32 : %d
[bits]\n",sizeof(u32)*8);
	printk(KERN_INFO "*\tEl equipo funciona con HZ =
%d", HZ);
	printk(KERN_INFO
"**************************************************")
;
	if(debug!=0)printk(KERN_INFO "Saliendo de funcion
ESTP_init\n");
}

static void ESTP_cong_avoid(struct sock *sk, u32 ack, u32
in_flight)
{
	struct TCP_sock *tp = TCP_sk(sk);
	if(debug!=0)printk(KERN_INFO "Entrando en funcion
ESTP_cong_avoid\n");
	if (!TCP_is_cwnd_limited(sk, in_flight)){
		if(debug!=0)printk(KERN_INFO "Saliendo
de funcion ESTP_cong_avoid porque !TCP_is_cwnd_limited(sk,
in_flight)\n");
		return;
	}
	if (tp->snd_cwnd <= tp->snd_ssthresh){
		TCP_slow_start(tp);
	}else{
		/* Do additive increase */
		if (tp->snd_cwnd < tp->snd_cwnd_clamp) {
			/* cwnd = cwnd + a(w) / cwnd */
			//_________________Claudio
Estevez_____________v
			// tp->snd_cwnd_cnt += ca->ai +
1;
			tp->snd_cwnd_cnt += 1;

	//_____________________________________________
^
			if (tp->snd_cwnd_cnt >= tp-
>snd_cwnd) {
				tp->snd_cwnd_cnt -=
tp->snd_cwnd;
				tp->snd_cwnd++;
			}
		}
	}
	if(debug!=0)printk(KERN_INFO "Saliendo de funcion
ESTP_cong_avoid\n");
}

static u32 ESTP_ssthresh(struct sock *sk)
{
	int i;
	unsigned char indice_aimd_ESTP;
	const struct TCP_sock *tp = TCP_sk(sk);
	//extern struct rtts_ultimos_valores;
	//extern struct segmentacion_alphas;
	u32 promedio_rtts;
	u32 seq_number_actual,alpha,rtt,cwnd_min,rtts;
	printk(KERN_INFO "\n");
	if(debug!=0){
		for(i=0;i<32;i++){
		printk(KERN_INFO "el valor de
segmentacion_alphas[%d]=%d",i,segmentacion_alphas[i].valor);
		}
		for(i=0;i<10;i++){
			printk(KERN_INFO "el valor de
rtts_ultimos_valores[%d]=%d",i,rtts_ultimos_valores[i].valor);
			}
	}
	promedio_rtts=0;
	//rtts =(tp->srtt)*500;
	rtts = ((tp->srtt)>>3 ) *(1000000/ HZ); //us
	/*srtt es un multiplo de 8 por definicion,
  RTT measurement
	u32	srtt;	// smoothed round trip time << 3
	u32	mdev;	// medium deviation
	u32	mdev_max;	// maximal mdev for the last rtt period
	u32	rttvar;	// smoothed mdev_max
	u32	rtt_seq;	// sequence number to update rttvar

	Por este motivo se divide por 8, dando una cantidad de
segundos asumiendo que un ciclo es un segundo,
	* Luego multiplicamos por 1000000 para tener el rtt en
[us]
	* La division por HZ, es debido a que el valor de srtt
depende de la granularidad
	*/
	if(promedio_flag!=0){
		datos.rtts_anterior=datos.rtts_anterior+rtts-
(int)rtts_ultimos_valores[datos.contador_rtts].valor;
			if(debug!=0) printk(KERN_INFO
"%d)estructura_rtts_llena:
%u\n",datos.id_test,datos.estructura_rtts_llena);
		if(!datos.estructura_rtts_llena){

	promedio_rtts=datos.rtts_anterior/(datos.contador_rtts+1
);
			if(debug!=0) printk(KERN_INFO
"%d)promedio_rtts: %u\n",datos.id_test,promedio_rtts);
		}else{

	promedio_rtts=datos.rtts_anterior/10;
		}

	rtts_ultimos_valores[datos.contador_rtts].valor=rtts;
		datos.contador_rtts++;
		if(datos.contador_rtts==10){
			datos.contador_rtts=0;
			datos.estructura_rtts_llena=1;
		}
	}
	if (debug!=0){
		printk(KERN_INFO "promedio_rtts :
%u\n",promedio_rtts);
		printk(KERN_INFO "tp->srtt = %u\n",tp-
>srtt);
		printk(KERN_INFO "RTT(rtt): %u\n",(tp-
>rcv_rtt_est).rtt);
		printk(KERN_INFO "RTT(time): %u\n",
jiffies_to_usecs(tp->rcv_rtt_est.rtt)>>3);
		// from TCP.c jiffies_to_usecs(tp-
>rcv_rtt_est.rtt)>>3
		printk(KERN_INFO
"RTT(rtt_indice_aimd_ESTPero): %u\n",tp->rcv_rtt_est.rtt);
		printk(KERN_INFO "RTT(tp-
>rcv_rtt_est.time): %u\n",tp->rcv_rtt_est.time);
		printk(KERN_INFO
"%d)seq_number_anterior:
%u\n",datos.id_test,datos.seq_number_anterior);
		printk(KERN_INFO "%d)datos.noseq :
%d",datos.id_test,datos.no_hay_seq_number_flag);
		printk(KERN_INFO "copied_seq :
%u\n",tp->copied_seq);
		printk(KERN_INFO "snd_nxt :   %u\n",tp-
>snd_nxt);
		printk(KERN_INFO "rcv_nxt :    %u\n",tp-
>rcv_nxt);
```

```
                        printk(KERN_INFO "rcv_wup :    %u\n",tp->rcv_wup);
                        printk(KERN_INFO "snd_wl1 :    %u\n",tp->snd_wl1);
                        printk(KERN_INFO "rtt_seq :    %u\n",tp->rtt_seq);
                        printk(KERN_INFO "write_seq :  %u\n",tp->write_seq);
                        printk(KERN_INFO "pushed_seq: %u\n",tp->pushed_seq);
                        printk(KERN_INFO "high_seq :   %u\n",tp->high_seq);
                        printk(KERN_INFO "urg_seq  :   %u\n",tp->urg_seq);
                        printk(KERN_INFO "rcv_tstamp: %u\n",tp->rcv_tstamp);
                        printk(KERN_INFO "snd_una: %u\n" , tp->snd_una);
                        //if(debug!=0) printk(KERN_INFO "rcv_tstamp:   %lu\n",tp->lrcvtime);
                }

                seq_number_actual = Obtener_seq_number(sk,&datos.no_hay_seq_number_flag);
                if(debug!=0) printk(KERN_INFO "%d)seq_number_actual: %u\n",datos.id_test,seq_number_actual);
                if((seq_number_actual>=datos.seq_number_anterior) && !datos.no_hay_seq_number_flag){//quizas deberia ser ">"
                        alpha = (seq_number_actual - datos.seq_number_anterior)/MSS;

                datos.seq_number_anterior=seq_number_actual;
                        datos.ultimo_alpha=alpha;
                        if(debug!=0) printk(KERN_INFO "alpha: %u\n",alpha);
                        if(debug!=0) printk(KERN_INFO "seq_number_actual>=datos.seq_number_anterior && !datos.no_hay_seq_number_flag");
                }else if(!datos.no_hay_seq_number_flag){
                        alpha = ((0xffffffff-datos.seq_number_anterior)+seq_number_actual)/MSS;

                datos.seq_number_anterior=seq_number_actual;
                        datos.ultimo_alpha=alpha;
                        if(debug!=0) printk(KERN_INFO "alpha: %u\n",alpha);
                        if(debug!=0) printk(KERN_INFO "seq_number_actual<datos.seq_number_anterior");
                        if(debug!=0) printk(KERN_INFO "%d)SEQ NUMBER SOBREPASADO\n",datos.id_test);
                }else{
                        alpha=datos.ultimo_alpha;
                        datos.no_hay_seq_number_flag=0;
                        if(debug!=0) printk(KERN_INFO "estoy en else");
                        if(debug!=0) printk(KERN_INFO "%d)NO HAY SEQ NUMBER ESTRUCTURA VACIA\n",datos.id_test);
                }

                if(debug!=0) printk(KERN_INFO "alpha: %u\n",alpha);

                /* Do multiplicative decrease */
                //_____________________Claudio Estevez_____________v
                if(promedio_flag!=0){
                        rtt = promedio_rtts; // unidades de milisegundos
                        if(debug!=0) printk(KERN_INFO "%d)rtt=promedio_rtts= %u\n",datos.id_test,rtt);
                }else{
                        rtt = rtts;
                        if(debug!=0) printk(KERN_INFO "%d)rtt=rtts= %u\n",datos.id_test,rtt);
                }
                //cwnd_min = 42808*rtt/1000000;
                cwnd_min = cir*rtt/(8*MSS); //CIR[Mbytes], rtt[us],MSS[bytes] => [Packets * s]
           if(debug!=0) printk(KERN_INFO "%d)cwnd_min: %u",datos.id_test,cwnd_min);
                //NO BORRAR: cwnd_min=(CIR[bytes/s]/MSS[bits])*(rtt[us]/1000000)
                //1000000 es para dejar RTT en segundos
                indice_aimd_ESTP=Obtener_indice_AIMD(alpha,segmentacion_alphas);
                if(debug!=0) printk(KERN_INFO "%d)indice_aimd_ESTP= %u",datos.id_test,indice_aimd_ESTP);

                if(tp->snd_cwnd > cwnd_min){
                        u32 delta_cwnd=(tp->snd_cwnd - cwnd_min);
                        //256 Por que si AIMD_ESTP= 128 => (2â¸-2â·)/2â¸=1-1/2=0.5 . Si AIMD_ESTP=0 (2â¸-2â°)/2â¸ = 1
                        return max(delta_cwnd * (( 256 - AIMD_ESTP[indice_aimd_ESTP].md) >> 8) + cwnd_min , cwnd_min);//a>>b => a/(2^b)
                }else{
                        return cwnd_min;
                }

//              cwnd_min = 4281*rtt/1000;

}


static struct TCP_congestion_ops TCP_ESTP = {
        .init           = ESTP_init,
        .ssthresh    = ESTP_ssthresh,
        .cong_avoid     = ESTP_cong_avoid,
        .min_cwnd       = TCP_RENO_min_cwnd,

        .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);


char Obtener_indice_AIMD(u32 alpha, struct ESTP_estructura_datos segmentacion_alphas[33]){
        int i=0;
        char punt=32;//cualquier valor mayor a 33
        for (i=0;i<33;i++){
                if(debug!=0) printk(KERN_INFO "\talpha: %u\n",alpha);
                if(debug!=0) printk(KERN_INFO "\tsegmentacion_alphas[i]: %u\n",segmentacion_alphas[i].valor);
                if (alpha < segmentacion_alphas[i].valor){
                        punt=i;
                        i=33;
                }
        }
        return punt;
```

```
}

u32 Obtener_seq_number(struct sock *sk,int *flag){
	//if((sk->sk_send_head)==NULL){
		// printk(KERN_INFO "\n El problema es sk->sk_send_head es igual a NULL\n");
	if((TCP_sk(sk)->snd_una)==0){
		*flag=1;
		return 0;
	}else{
		return TCP_sk(sk)->snd_una; //Funciona mejor
		//return TCP_sk(sk)->high_seq;
	}
//return TCP_SKB_CB(sk->sk_send_head)->seq;
//	return TCP_sk(sk)->high_seq;
}

MODULE_AUTHOR("Sergio Angulo");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("ESTP");


//codigo funcion creada externamente

void Obtener_segmentacion_alphas(struct ESTP_estructura_datos m[33],int cir){
	switch (cir){
		case 10:
			m[0].valor=1;
			m[1].valor=450;
			m[2].valor=914;
			m[3].valor=1393;
			m[4].valor=1887;
			m[5].valor=2400;
			m[6].valor=2931;
			m[7].valor=3482;
			m[8].valor=4054;
			m[9].valor=4650;
			m[10].valor=5272;
			m[11].valor=5921;
			m[12].valor=6600;
			m[13].valor=7312;
			m[14].valor=8061;
			m[15].valor=8851;
			m[16].valor=9685;
			m[17].valor=10570;
			m[18].valor=11512;
			m[19].valor=12520;
			m[20].valor=13602;
			m[21].valor=14770;
			m[22].valor=16041;
			m[23].valor=17432;
			m[24].valor=18971;
			m[25].valor=20690;
			m[26].valor=22640;
			m[27].valor=24890;
			m[28].valor=27552;
			m[29].valor=30810;
			m[30].valor=35010;
			m[31].valor=40930;
			m[32].valor=51050;
		break;
		case 20:
			m[0].valor=1;
			m[1].valor=900;
			m[2].valor=1827;
			m[3].valor=2784;
			m[4].valor=3774;
			m[5].valor=4799;
			m[6].valor=5861;
			m[7].valor=6963;
			m[8].valor=8108;
			m[9].valor=9300;
			m[10].valor=10543;
			m[11].valor=11841;
			m[12].valor=13199;
			m[13].valor=14624;
			m[14].valor=16121;
			m[15].valor=17700;
			m[16].valor=19369;
			m[17].valor=21139;
			m[18].valor=23024;
			m[19].valor=25039;
			m[20].valor=27202;
			m[21].valor=29540;
			m[22].valor=32080;
			m[23].valor=34864;
			m[24].valor=37940;
			m[25].valor=41379;
			m[26].valor=45278;
			m[27].valor=49780;
			m[28].valor=55103;
			m[29].valor=61619;
			m[30].valor=70020;
			m[31].valor=81859;
			m[32].valor=102099;
		break;
		case 30:
			m[0].valor=1;
			m[1].valor=1349;
			m[2].valor=2739;
			m[3].valor=4176;
			m[4].valor=5660;
			m[5].valor=7197;
			m[6].valor=8790;
			m[7].valor=10443;
			m[8].valor=12161;
			m[9].valor=13949;
			m[10].valor=15813;
			m[11].valor=17760;
			m[12].valor=19798;
			m[13].valor=21935;
			m[14].valor=24182;
			m[15].valor=26550;
			m[16].valor=29053;
			m[17].valor=31709;
			m[18].valor=34535;
			m[19].valor=37557;
			m[20].valor=40803;
			m[21].valor=44309;
			m[22].valor=48120;
			m[23].valor=52295;
			m[24].valor=56910;
			m[25].valor=62068;
			m[26].valor=67917;
			m[27].valor=74669;
			m[28].valor=82655;
			m[29].valor=92428;
			m[30].valor=105029;
			m[31].valor=122788;
			m[32].valor=153148;
		break;
		case 40:
			m[0].valor=1;
			m[1].valor=1798;
			m[2].valor=3652;
			m[3].valor=5567;
			m[4].valor=7547;
			m[5].valor=9596;
			m[6].valor=11720;
			m[7].valor=13924;
			m[8].valor=16215;
			m[9].valor=18599;
			m[10].valor=21084;
			m[11].valor=23680;
```

```
            m[12].valor=26397;
            m[13].valor=29246;
            m[14].valor=32242;
            m[15].valor=35399;
            m[16].valor=38737;
            m[17].valor=42278;
            m[18].valor=46047;
            m[19].valor=50076;
            m[20].valor=54404;
            m[21].valor=59078;
            m[22].valor=64160;
            m[23].valor=69726;
            m[24].valor=75879;
            m[25].valor=82758;
            m[26].valor=90556;
            m[27].valor=99558;
            m[28].valor=110206;
            m[29].valor=123237;
            m[30].valor=140038;
            m[31].valor=163717;
            m[32].valor=204197;
        break;
        case 50:
            m[0].valor=1;
            m[1].valor=2247;
            m[2].valor=4565;
            m[3].valor=6959;
            m[4].valor=9433;
            m[5].valor=11995;
            m[6].valor=14650;
            m[7].valor=17405;
            m[8].valor=20268;
            m[9].valor=23248;
            m[10].valor=26355;
            m[11].valor=29600;
            m[12].valor=32996;
            m[13].valor=36558;
            m[14].valor=40302;
            m[15].valor=44249;
            m[16].valor=48421;
            m[17].valor=52847;
            m[18].valor=57558;
            m[19].valor=62595;
            m[20].valor=68005;
            m[21].valor=73848;
            m[22].valor=80200;
            m[23].valor=87157;
            m[24].valor=94849;
            m[25].valor=103447;
            m[26].valor=113195;
            m[27].valor=124448;
            m[28].valor=137757;
            m[29].valor=154047;
            m[30].valor=175047;
            m[31].valor=204646;
            m[32].valor=255246;
        break;
        case 60:
            m[0].valor=1;
            m[1].valor=2697;
            m[2].valor=5478;
            m[3].valor=8350;
            m[4].valor=11320;
            m[5].valor=14394;
            m[6].valor=17580;
            m[7].valor=20886;
            m[8].valor=24322;
            m[9].valor=27898;
            m[10].valor=31626;
            m[11].valor=35520;
            m[12].valor=39595;
            m[13].valor=43869;
            m[14].valor=48362;
            m[15].valor=53098;
            m[16].valor=58106;
            m[17].valor=63416;
            m[18].valor=69070;
            m[19].valor=75114;
            m[20].valor=81605;
            m[21].valor=88617;
            m[22].valor=96239;
            m[23].valor=104589;
            m[24].valor=113818;
            m[25].valor=124136;
            m[26].valor=135833;
            m[27].valor=149337;
            m[28].valor=165308;
            m[29].valor=184856;
            m[30].valor=210057;
            m[31].valor=245575;
            m[32].valor=306295;
        break;
        case 70:
            m[0].valor=1;
            m[1].valor=3146;
            m[2].valor=6391;
            m[3].valor=9742;
            m[4].valor=13206;
            m[5].valor=16793;
            m[6].valor=20510;
            m[7].valor=24367;
            m[8].valor=28375;
            m[9].valor=32547;
            m[10].valor=36897;
            m[11].valor=41440;
            m[12].valor=46194;
            m[13].valor=51180;
            m[14].valor=56422;
            m[15].valor=61948;
            m[16].valor=67790;
            m[17].valor=73986;
            m[18].valor=80581;
            m[19].valor=87632;
            m[20].valor=95206;
            m[21].valor=103387;
            m[22].valor=112279;
            m[23].valor=122020;
            m[24].valor=132788;
            m[25].valor=144825;
            m[26].valor=158472;
            m[27].valor=174226;
            m[28].valor=192860;
            m[29].valor=215665;
            m[30].valor=245066;
            m[31].valor=286504;
            m[32].valor=357344;
        break;
        case 80:
            m[0].valor=1;
            m[1].valor=3595;
            m[2].valor=7303;
            m[3].valor=11133;
            m[4].valor=15093;
            m[5].valor=19192;
            m[6].valor=23439;
            m[7].valor=27847;
            m[8].valor=32428;
            m[9].valor=37196;
            m[10].valor=42167;
            m[11].valor=47359;
            m[12].valor=52793;
            m[13].valor=58492;
            m[14].valor=64483;
            m[15].valor=70798;
            m[16].valor=77474;
            m[17].valor=84555;
```

```
            m[18].valor=92093;
            m[19].valor=100151;
            m[20].valor=108807;
            m[21].valor=118156;
            m[22].valor=128319;
            m[23].valor=139451;
            m[24].valor=151757;
            m[25].valor=165514;
            m[26].valor=181111;
            m[27].valor=199116;
            m[28].valor=220411;
            m[29].valor=246474;
            m[30].valor=280075;
            m[31].valor=327433;
            m[32].valor=408393;
        break;
        case 90:
            m[0].valor=1;
            m[1].valor=4044;
            m[2].valor=8216;
            m[3].valor=12525;
            m[4].valor=16979;
            m[5].valor=21590;
            m[6].valor=26369;
            m[7].valor=31328;
            m[8].valor=36482;
            m[9].valor=41846;
            m[10].valor=47438;
            m[11].valor=53279;
            m[12].valor=59392;
            m[13].valor=65803;
            m[14].valor=72543;
            m[15].valor=79647;
            m[16].valor=87158;
            m[17].valor=95124;
            m[18].valor=103604;
            m[19].valor=112670;
            m[20].valor=122408;
            m[21].valor=132925;
            m[22].valor=144359;
            m[23].valor=156882;
            m[24].valor=170727;
            m[25].valor=186203;
            m[26].valor=203749;
            m[27].valor=224005;
            m[28].valor=247962;
            m[29].valor=277283;
            m[30].valor=315084;
            m[31].valor=368363;
            m[32].valor=459442;
        break;
        case 100:
            m[0].valor=1;
            m[1].valor=4494;
            m[2].valor=9129;
            m[3].valor=13916;
            m[4].valor=18866;
            m[5].valor=23989;
            m[6].valor=29299;
            m[7].valor=34809;
            m[8].valor=40535;
            m[9].valor=46495;
            m[10].valor=52709;
            m[11].valor=59199;
            m[12].valor=65991;
            m[13].valor=73114;
            m[14].valor=80603;
            m[15].valor=88497;
            m[16].valor=96842;
            m[17].valor=105693;
            m[18].valor=115116;
            m[19].valor=125189;
            m[20].valor=136008;
            m[21].valor=147695;
            m[22].valor=160398;
            m[23].valor=174314;
            m[24].valor=189696;
            m[25].valor=206893;
            m[26].valor=226388;
            m[27].valor=248894;
            m[28].valor=275513;
            m[29].valor=308092;
            m[30].valor=350094;
            m[31].valor=409292;
            m[32].valor=510491;
        break;
        case 110:
            m[0].valor=1;
            m[1].valor=4943;
            m[2].valor=10042;
            m[3].valor=15308;
            m[4].valor=20752;
            m[5].valor=26388;
            m[6].valor=32229;
            m[7].valor=38290;
            m[8].valor=44589;
            m[9].valor=51145;
            m[10].valor=57980;
            m[11].valor=65119;
            m[12].valor=72590;
            m[13].valor=80426;
            m[14].valor=88663;
            m[15].valor=97346;
            m[16].valor=106526;
            m[17].valor=116262;
            m[18].valor=126627;
            m[19].valor=137708;
            m[20].valor=149609;
            m[21].valor=162464;
            m[22].valor=176438;
            m[23].valor=191745;
            m[24].valor=208666;
            m[25].valor=227582;
            m[26].valor=249027;
            m[27].valor=273784;
            m[28].valor=303064;
            m[29].valor=338901;
            m[30].valor=385103;
            m[31].valor=450221;
            m[32].valor=561540;
        break;
        case 120:
            m[0].valor=1;
            m[1].valor=5392;
            m[2].valor=10955;
            m[3].valor=16699;
            m[4].valor=22639;
            m[5].valor=28787;
            m[6].valor=35159;
            m[7].valor=41771;
            m[8].valor=48642;
            m[9].valor=55794;
            m[10].valor=63251;
            m[11].valor=71038;
            m[12].valor=79189;
            m[13].valor=87737;
            m[14].valor=96723;
            m[15].valor=106196;
            m[16].valor=116210;
            m[17].valor=126832;
            m[18].valor=138139;
            m[19].valor=150226;
            m[20].valor=163210;
            m[21].valor=177233;
            m[22].valor=192478;
            m[23].valor=209176;
```

```
            m[24].valor=227635;
            m[25].valor=248271;
            m[26].valor=271666;
            m[27].valor=298673;
            m[28].valor=330616;
            m[29].valor=369710;
            m[30].valor=420112;
            m[31].valor=491150;
            m[32].valor=612589;
        break;
        case 130:
            m[0].valor=1;
            m[1].valor=5841;
            m[2].valor=11867;
            m[3].valor=18091;
            m[4].valor=24525;
            m[5].valor=31186;
            m[6].valor=38088;
            m[7].valor=45251;
            m[8].valor=52696;
            m[9].valor=60444;
            m[10].valor=68521;
            m[11].valor=76958;
            m[12].valor=85788;
            m[13].valor=95048;
            m[14].valor=104784;
            m[15].valor=115046;
            m[16].valor=125894;
            m[17].valor=137401;
            m[18].valor=149650;
            m[19].valor=162745;
            m[20].valor=176811;
            m[21].valor=192003;
            m[22].valor=208518;
            m[23].valor=226607;
            m[24].valor=246605;
            m[25].valor=268960;
            m[26].valor=294304;
            m[27].valor=323562;
            m[28].valor=358167;
            m[29].valor=400519;
            m[30].valor=455122;
            m[31].valor=532079;
            m[32].valor=663638;
        break;
        case 140:
            m[0].valor=1;
            m[1].valor=6291;
            m[2].valor=12780;
            m[3].valor=19482;
            m[4].valor=26412;
            m[5].valor=33585;
            m[6].valor=41018;
            m[7].valor=48732;
            m[8].valor=56749;
            m[9].valor=65093;
            m[10].valor=73792;
            m[11].valor=82878;
            m[12].valor=92387;
            m[13].valor=102359;
            m[14].valor=112844;
            m[15].valor=123895;
            m[16].valor=135578;
            m[17].valor=147970;
            m[18].valor=161162;
            m[19].valor=175264;
            m[20].valor=190411;
            m[21].valor=206772;
            m[22].valor=224557;
            m[23].valor=244039;
            m[24].valor=265574;
            m[25].valor=289649;
            m[26].valor=316943;
            m[27].valor=348452;
            m[28].valor=385718;
            m[29].valor=431329;
            m[30].valor=490131;
            m[31].valor=573008;
            m[32].valor=714687;
        break;
        case 150:
            m[0].valor=1;
            m[1].valor=6740;
            m[2].valor=13693;
            m[3].valor=20874;
            m[4].valor=28298;
            m[5].valor=35983;
            m[6].valor=43948;
            m[7].valor=52213;
            m[8].valor=60802;
            m[9].valor=69742;
            m[10].valor=79063;
            m[11].valor=88798;
            m[12].valor=98986;
            m[13].valor=109671;
            m[14].valor=120904;
            m[15].valor=132745;
            m[16].valor=145262;
            m[17].valor=158539;
            m[18].valor=172673;
            m[19].valor=187783;
            m[20].valor=204012;
            m[21].valor=221542;
            m[22].valor=240597;
            m[23].valor=261470;
            m[24].valor=284544;
            m[25].valor=310338;
            m[26].valor=339582;
            m[27].valor=373341;
            m[28].valor=413269;
            m[29].valor=462138;
            m[30].valor=525140;
            m[31].valor=613937;
            m[32].valor=765736;
        break;
        case 160:
            m[0].valor=1;
            m[1].valor=7189;
            m[2].valor=14606;
            m[3].valor=22265;
            m[4].valor=30185;
            m[5].valor=38382;
            m[6].valor=46878;
            m[7].valor=55694;
            m[8].valor=64856;
            m[9].valor=74392;
            m[10].valor=84334;
            m[11].valor=94718;
            m[12].valor=105585;
            m[13].valor=116982;
            m[14].valor=128964;
            m[15].valor=141594;
            m[16].valor=154947;
            m[17].valor=169108;
            m[18].valor=184185;
            m[19].valor=200301;
            m[20].valor=217613;
            m[21].valor=236311;
            m[22].valor=256637;
            m[23].valor=278901;
            m[24].valor=303514;
            m[25].valor=331028;
            m[26].valor=362221;
            m[27].valor=398230;
            m[28].valor=440820;
            m[29].valor=492947;
```

```
            m[30].valor=560149;
            m[31].valor=654866;
            m[32].valor=816785;
        break;
        case 170:
            m[0].valor=1;
            m[1].valor=7639;
            m[2].valor=15519;
            m[3].valor=23657;
            m[4].valor=32071;
            m[5].valor=40781;
            m[6].valor=49807;
            m[7].valor=59175;
            m[8].valor=68909;
            m[9].valor=79041;
            m[10].valor=89605;
            m[11].valor=100637;
            m[12].valor=112184;
            m[13].valor=124293;
            m[14].valor=137024;
            m[15].valor=150444;
            m[16].valor=164631;
            m[17].valor=179678;
            m[18].valor=195696;
            m[19].valor=212820;
            m[20].valor=231214;
            m[21].valor=251080;
            m[22].valor=272677;
            m[23].valor=296333;
            m[24].valor=322483;
            m[25].valor=351717;
            m[26].valor=384859;
            m[27].valor=423119;
            m[28].valor=468372;
            m[29].valor=523756;
            m[30].valor=595159;
            m[31].valor=695795;
            m[32].valor=867834;
        break;
        case 180:
            m[0].valor=1;
            m[1].valor=8088;
            m[2].valor=16431;
            m[3].valor=25049;
            m[4].valor=33958;
            m[5].valor=43180;
            m[6].valor=52737;
            m[7].valor=62655;
            m[8].valor=72963;
            m[9].valor=83691;
            m[10].valor=94875;
            m[11].valor=106557;
            m[12].valor=118783;
            m[13].valor=131605;
            m[14].valor=145085;
            m[15].valor=159293;
            m[16].valor=174315;
            m[17].valor=190247;
            m[18].valor=207208;
            m[19].valor=225339;
            m[20].valor=244814;
            m[21].valor=265850;
            m[22].valor=288716;
            m[23].valor=313764;
            m[24].valor=341453;
            m[25].valor=372406;
            m[26].valor=407498;
            m[27].valor=448009;
            m[28].valor=495923;
            m[29].valor=554565;
            m[30].valor=630168;
            m[31].valor=736724;
            m[32].valor=918883;
        break;
        case 190:
            m[0].valor=1;
            m[1].valor=8537;
            m[2].valor=17344;
            m[3].valor=26440;
            m[4].valor=35844;
            m[5].valor=45579;
            m[6].valor=55667;
            m[7].valor=66136;
            m[8].valor=77016;
            m[9].valor=88340;
            m[10].valor=100146;
            m[11].valor=112477;
            m[12].valor=125382;
            m[13].valor=138916;
            m[14].valor=153145;
            m[15].valor=168143;
            m[16].valor=183999;
            m[17].valor=200816;
            m[18].valor=218719;
            m[19].valor=237858;
            m[20].valor=258415;
            m[21].valor=280619;
            m[22].valor=304756;
            m[23].valor=331195;
            m[24].valor=360422;
            m[25].valor=393095;
            m[26].valor=430137;
            m[27].valor=472898;
            m[28].valor=523474;
            m[29].valor=585374;
            m[30].valor=665177;
            m[31].valor=777653;
            m[32].valor=969932;
        break;
        case 200:
            m[0].valor=1;
            m[1].valor=8986;
            m[2].valor=18257;
            m[3].valor=27832;
            m[4].valor=37731;
            m[5].valor=47977;
            m[6].valor=58597;
            m[7].valor=69617;
            m[8].valor=81069;
            m[9].valor=92989;
            m[10].valor=105417;
            m[11].valor=118397;
            m[12].valor=131981;
            m[13].valor=146227;
            m[14].valor=161205;
            m[15].valor=176993;
            m[16].valor=193683;
            m[17].valor=211385;
            m[18].valor=230231;
            m[19].valor=250376;
            m[20].valor=272016;
            m[21].valor=295388;
            m[22].valor=320796;
            m[23].valor=348626;
            m[24].valor=379392;
            m[25].valor=413784;
            m[26].valor=452775;
            m[27].valor=497787;
            m[28].valor=551025;
            m[29].valor=616183;
            m[30].valor=700186;
            m[31].valor=818582;
            m[32].valor=1020981;
        break;
        case 210:
            m[0].valor=1;
```

```
            m[1].valor=9436;
            m[2].valor=19170;
            m[3].valor=29223;
            m[4].valor=39617;
            m[5].valor=50376;
            m[6].valor=61527;
            m[7].valor=73098;
            m[8].valor=85123;
            m[9].valor=97639;
            m[10].valor=110688;
            m[11].valor=124317;
            m[12].valor=138580;
            m[13].valor=153539;
            m[14].valor=169265;
            m[15].valor=185842;
            m[16].valor=203367;
            m[17].valor=221955;
            m[18].valor=241742;
            m[19].valor=262895;
            m[20].valor=285617;
            m[21].valor=310158;
            m[22].valor=336836;
            m[23].valor=366058;
            m[24].valor=398361;
            m[25].valor=434473;
            m[26].valor=475414;
            m[27].valor=522677;
            m[28].valor=578577;
            m[29].valor=646992;
            m[30].valor=735196;
            m[31].valor=859511;
            m[32].valor=1072030;
        break;
        case 220:
            m[0].valor=1;
            m[1].valor=9885;
            m[2].valor=20083;
            m[3].valor=30615;
            m[4].valor=41504;
            m[5].valor=52775;
            m[6].valor=64456;
            m[7].valor=76579;
            m[8].valor=89176;
            m[9].valor=102288;
            m[10].valor=115958;
            m[11].valor=130236;
            m[12].valor=145179;
            m[13].valor=160850;
            m[14].valor=177325;
            m[15].valor=194692;
            m[16].valor=213051;
            m[17].valor=232524;
            m[18].valor=253254;
            m[19].valor=275414;
            m[20].valor=299217;
            m[21].valor=324927;
            m[22].valor=352875;
            m[23].valor=383489;
            m[24].valor=417331;
            m[25].valor=455163;
            m[26].valor=498053;
            m[27].valor=547566;
            m[28].valor=606128;
            m[29].valor=677801;
            m[30].valor=770205;
            m[31].valor=900440;
            m[32].valor=1123079;
        break;
        case 230:
            m[0].valor=1;
            m[1].valor=10334;
            m[2].valor=20995;
            m[3].valor=32006;
            m[4].valor=43390;
            m[5].valor=55174;
            m[6].valor=67386;
            m[7].valor=80059;
            m[8].valor=93230;
            m[9].valor=106938;
            m[10].valor=121229;
            m[11].valor=136156;
            m[12].valor=151778;
            m[13].valor=168161;
            m[14].valor=185386;
            m[15].valor=203541;
            m[16].valor=222735;
            m[17].valor=243093;
            m[18].valor=264765;
            m[19].valor=287933;
            m[20].valor=312818;
            m[21].valor=339697;
            m[22].valor=368915;
            m[23].valor=400920;
            m[24].valor=436300;
            m[25].valor=475852;
            m[26].valor=520692;
            m[27].valor=572455;
            m[28].valor=633679;
            m[29].valor=708611;
            m[30].valor=805214;
            m[31].valor=941369;
            m[32].valor=1174128;
        break;
        case 240:
            m[0].valor=1;
            m[1].valor=10783;
            m[2].valor=21908;
            m[3].valor=33398;
            m[4].valor=45277;
            m[5].valor=57573;
            m[6].valor=70316;
            m[7].valor=83540;
            m[8].valor=97283;
            m[9].valor=111587;
            m[10].valor=126500;
            m[11].valor=142076;
            m[12].valor=158377;
            m[13].valor=175473;
            m[14].valor=193446;
            m[15].valor=212391;
            m[16].valor=232419;
            m[17].valor=253662;
            m[18].valor=276276;
            m[19].valor=300452;
            m[20].valor=326419;
            m[21].valor=354466;
            m[22].valor=384955;
            m[23].valor=418351;
            m[24].valor=455270;
            m[25].valor=496541;
            m[26].valor=543330;
            m[27].valor=597345;
            m[28].valor=661230;
            m[29].valor=739420;
            m[30].valor=840224;
            m[31].valor=982298;
            m[32].valor=1225177;
        break;
        case 250:
            m[0].valor=1;
            m[1].valor=11233;
            m[2].valor=22821;
            m[3].valor=34789;
            m[4].valor=47163;
            m[5].valor=59972;
            m[6].valor=73246;
```

```
            m[7].valor=87021;
            m[8].valor=101337;
            m[9].valor=116237;
            m[10].valor=131771;
            m[11].valor=147996;
            m[12].valor=164976;
            m[13].valor=182784;
            m[14].valor=201506;
            m[15].valor=221241;
            m[16].valor=242103;
            m[17].valor=264231;
            m[18].valor=287788;
            m[19].valor=312970;
            m[20].valor=340020;
            m[21].valor=369235;
            m[22].valor=400994;
            m[23].valor=435783;
            m[24].valor=474239;
            m[25].valor=517230;
            m[26].valor=565969;
            m[27].valor=622234;
            m[28].valor=688781;
            m[29].valor=770229;
            m[30].valor=875233;
            m[31].valor=1023228;
            m[32].valor=1276226;
        break;
        case 260:
            m[0].valor=1;
            m[1].valor=11682;
            m[2].valor=23734;
            m[3].valor=36181;
            m[4].valor=49050;
            m[5].valor=62370;
            m[6].valor=76176;
            m[7].valor=90502;
            m[8].valor=105390;
            m[9].valor=120886;
            m[10].valor=137042;
            m[11].valor=153916;
            m[12].valor=171575;
            m[13].valor=190095;
            m[14].valor=209566;
            m[15].valor=230090;
            m[16].valor=251787;
            m[17].valor=274801;
            m[18].valor=299299;
            m[19].valor=325489;
            m[20].valor=353620;
            m[21].valor=384005;
            m[22].valor=417034;
            m[23].valor=453214;
            m[24].valor=493209;
            m[25].valor=537919;
            m[26].valor=588608;
            m[27].valor=647123;
            m[28].valor=716333;
            m[29].valor=801038;
            m[30].valor=910242;
            m[31].valor=1064157;
            m[32].valor=1327275;
        break;
        case 270:
            m[0].valor=1;
            m[1].valor=12131;
            m[2].valor=24647;
            m[3].valor=37572;
            m[4].valor=50936;
            m[5].valor=64769;
            m[6].valor=79105;
            m[7].valor=93983;
            m[8].valor=109443;
            m[9].valor=125535;
            m[10].valor=142312;
            m[11].valor=159835;
            m[12].valor=178174;
            m[13].valor=197407;
            m[14].valor=217626;
            m[15].valor=238940;
            m[16].valor=261472;
            m[17].valor=285370;
            m[18].valor=310811;
            m[19].valor=338008;
            m[20].valor=367221;
            m[21].valor=398774;
            m[22].valor=433074;
            m[23].valor=470645;
            m[24].valor=512178;
            m[25].valor=558608;
            m[26].valor=611246;
            m[27].valor=672013;
            m[28].valor=743884;
            m[29].valor=831847;
            m[30].valor=945251;
            m[31].valor=1105086;
            m[32].valor=1378324;
        break;
        case 280:
            m[0].valor=1;
            m[1].valor=12580;
            m[2].valor=25559;
            m[3].valor=38964;
            m[4].valor=52823;
            m[5].valor=67168;
            m[6].valor=82035;
            m[7].valor=97463;
            m[8].valor=113497;
            m[9].valor=130185;
            m[10].valor=147583;
            m[11].valor=165755;
            m[12].valor=184773;
            m[13].valor=204718;
            m[14].valor=225687;
            m[15].valor=247789;
            m[16].valor=271156;
            m[17].valor=295939;
            m[18].valor=322322;
            m[19].valor=350527;
            m[20].valor=380822;
            m[21].valor=413543;
            m[22].valor=449114;
            m[23].valor=488077;
            m[24].valor=531148;
            m[25].valor=579298;
            m[26].valor=633885;
            m[27].valor=696902;
            m[28].valor=771435;
            m[29].valor=862656;
            m[30].valor=980261;
            m[31].valor=1146015;
            m[32].valor=1429373;
        break;
        case 290:
            m[0].valor=1;
            m[1].valor=13030;
            m[2].valor=26472;
            m[3].valor=40355;
            m[4].valor=54709;
            m[5].valor=69567;
            m[6].valor=84965;
            m[7].valor=100944;
            m[8].valor=117550;
            m[9].valor=134834;
            m[10].valor=152854;
            m[11].valor=171675;
            m[12].valor=191372;
```

```
            m[13].valor=212029;
            m[14].valor=233747;
            m[15].valor=256639;
            m[16].valor=280840;
            m[17].valor=306508;
            m[18].valor=333834;
            m[19].valor=363045;
            m[20].valor=394423;
            m[21].valor=428313;
            m[22].valor=465153;
            m[23].valor=505508;
            m[24].valor=550117;
            m[25].valor=599987;
            m[26].valor=656524;
            m[27].valor=721791;
            m[28].valor=798986;
            m[29].valor=893465;
            m[30].valor=1015270;
            m[31].valor=1186944;
            m[32].valor=1480422;
        break;
        case 300:
            m[0].valor=1;
            m[1].valor=13479;
            m[2].valor=27385;
            m[3].valor=41747;
            m[4].valor=56596;
            m[5].valor=71966;
            m[6].valor=87895;
            m[7].valor=104425;
            m[8].valor=121604;
            m[9].valor=139484;
            m[10].valor=158125;
            m[11].valor=177595;
            m[12].valor=197970;
            m[13].valor=219341;
            m[14].valor=241807;
            m[15].valor=265488;
            m[16].valor=290524;
            m[17].valor=317077;
            m[18].valor=345345;
            m[19].valor=375564;
            m[20].valor=408023;
            m[21].valor=443082;
            m[22].valor=481193;
            m[23].valor=522939;
            m[24].valor=569087;
            m[25].valor=620676;
            m[26].valor=679163;
            m[27].valor=746681;
            m[28].valor=826538;
            m[29].valor=924274;
            m[30].valor=1050279;
            m[31].valor=1227873;
            m[32].valor=1531471;
        break;
        case 310:
            m[0].valor=1;
            m[1].valor=13928;
            m[2].valor=28298;
            m[3].valor=43138;
            m[4].valor=58482;
            m[5].valor=74365;
            m[6].valor=90825;
            m[7].valor=107906;
            m[8].valor=125657;
            m[9].valor=144133;
            m[10].valor=163396;
            m[11].valor=183515;
            m[12].valor=204569;
            m[13].valor=226652;
            m[14].valor=249867;
            m[15].valor=274338;
            m[16].valor=300208;
            m[17].valor=327647;
            m[18].valor=356857;
            m[19].valor=388083;
            m[20].valor=421624;
            m[21].valor=457852;
            m[22].valor=497233;
            m[23].valor=540370;
            m[24].valor=588056;
            m[25].valor=641365;
            m[26].valor=701801;
            m[27].valor=771570;
            m[28].valor=854089;
            m[29].valor=955083;
            m[30].valor=1085288;
            m[31].valor=1268802;
            m[32].valor=1582520;
        break;
        case 320:
            m[0].valor=1;
            m[1].valor=14378;
            m[2].valor=29211;
            m[3].valor=44530;
            m[4].valor=60369;
            m[5].valor=76763;
            m[6].valor=93754;
            m[7].valor=111387;
            m[8].valor=129711;
            m[9].valor=148783;
            m[10].valor=168666;
            m[11].valor=189434;
            m[12].valor=211168;
            m[13].valor=233963;
            m[14].valor=257927;
            m[15].valor=283188;
            m[16].valor=309892;
            m[17].valor=338216;
            m[18].valor=368368;
            m[19].valor=400602;
            m[20].valor=435225;
            m[21].valor=472621;
            m[22].valor=513273;
            m[23].valor=557802;
            m[24].valor=607026;
            m[25].valor=662054;
            m[26].valor=724440;
            m[27].valor=796459;
            m[28].valor=881640;
            m[29].valor=985893;
            m[30].valor=1120298;
            m[31].valor=1309731;
            m[32].valor=1633569;
        break;
        case 330:
            m[0].valor=1;
            m[1].valor=14827;
            m[2].valor=30123;
            m[3].valor=45921;
            m[4].valor=62255;
            m[5].valor=79162;
            m[6].valor=96684;
            m[7].valor=114867;
            m[8].valor=133764;
            m[9].valor=153432;
            m[10].valor=173937;
            m[11].valor=195354;
            m[12].valor=217767;
            m[13].valor=241275;
            m[14].valor=265988;
            m[15].valor=292037;
            m[16].valor=319576;
            m[17].valor=348785;
            m[18].valor=379880;
```

```
            m[19].valor=413121;
            m[20].valor=448826;
            m[21].valor=487390;
            m[22].valor=529312;
            m[23].valor=575233;
            m[24].valor=625996;
            m[25].valor=682743;
            m[26].valor=747079;
            m[27].valor=821349;
            m[28].valor=909191;
            m[29].valor=1016702;
            m[30].valor=1155307;
            m[31].valor=1350660;
            m[32].valor=1684618;
        break;
        case 340:
            m[0].valor=1;
            m[1].valor=15276;
            m[2].valor=31036;
            m[3].valor=47313;
            m[4].valor=64142;
            m[5].valor=81561;
            m[6].valor=99614;
            m[7].valor=118348;
            m[8].valor=137817;
            m[9].valor=158081;
            m[10].valor=179208;
            m[11].valor=201274;
            m[12].valor=224366;
            m[13].valor=248586;
            m[14].valor=274048;
            m[15].valor=300887;
            m[16].valor=329260;
            m[17].valor=359354;
            m[18].valor=391391;
            m[19].valor=425639;
            m[20].valor=462426;
            m[21].valor=502160;
            m[22].valor=545352;
            m[23].valor=592664;
            m[24].valor=644965;
            m[25].valor=703433;
            m[26].valor=769718;
            m[27].valor=846238;
            m[28].valor=936742;
            m[29].valor=1047511;
            m[30].valor=1190316;
            m[31].valor=1391589;
            m[32].valor=1735667;
        break;
        case 350:
            m[0].valor=1;
            m[1].valor=15725;
            m[2].valor=31949;
            m[3].valor=48705;
            m[4].valor=66028;
            m[5].valor=83960;
            m[6].valor=102544;
            m[7].valor=121829;
            m[8].valor=141871;
            m[9].valor=162731;
            m[10].valor=184479;
            m[11].valor=207194;
            m[12].valor=230965;
            m[13].valor=255897;
            m[14].valor=282108;
            m[15].valor=309736;
            m[16].valor=338944;
            m[17].valor=369924;
            m[18].valor=402903;
            m[19].valor=438158;
            m[20].valor=476027;
            m[21].valor=516929;
            m[22].valor=561392;
            m[23].valor=610095;
            m[24].valor=663935;
            m[25].valor=724122;
            m[26].valor=792356;
            m[27].valor=871127;
            m[28].valor=964294;
            m[29].valor=1078320;
            m[30].valor=1225325;
            m[31].valor=1432518;
            m[32].valor=1786716;
        break;
        case 360:
            m[0].valor=1;
            m[1].valor=16175;
            m[2].valor=32862;
            m[3].valor=50096;
            m[4].valor=67915;
            m[5].valor=86359;
            m[6].valor=105474;
            m[7].valor=125310;
            m[8].valor=145924;
            m[9].valor=167380;
            m[10].valor=189750;
            m[11].valor=213113;
            m[12].valor=237564;
            m[13].valor=263208;
            m[14].valor=290168;
            m[15].valor=318586;
            m[16].valor=348628;
            m[17].valor=380493;
            m[18].valor=414414;
            m[19].valor=450677;
            m[20].valor=489628;
            m[21].valor=531698;
            m[22].valor=577432;
            m[23].valor=627527;
            m[24].valor=682904;
            m[25].valor=744811;
            m[26].valor=814995;
            m[27].valor=896017;
            m[28].valor=991845;
            m[29].valor=1109129;
            m[30].valor=1260335;
            m[31].valor=1473447;
            m[32].valor=1837765;
        break;
        case 370:
            m[0].valor=1;
            m[1].valor=16624;
            m[2].valor=33774;
            m[3].valor=51488;
            m[4].valor=69801;
            m[5].valor=88758;
            m[6].valor=108403;
            m[7].valor=128791;
            m[8].valor=149978;
            m[9].valor=172030;
            m[10].valor=195020;
            m[11].valor=219033;
            m[12].valor=244163;
            m[13].valor=270520;
            m[14].valor=298228;
            m[15].valor=327436;
            m[16].valor=358313;
            m[17].valor=391062;
            m[18].valor=425926;
            m[19].valor=463196;
            m[20].valor=503229;
            m[21].valor=546468;
            m[22].valor=593471;
            m[23].valor=644958;
            m[24].valor=701874;
```

```
            m[25].valor=765500;
            m[26].valor=837634;
            m[27].valor=920906;
            m[28].valor=1019396;
            m[29].valor=1139938;
            m[30].valor=1295344;
            m[31].valor=1514376;
            m[32].valor=1888814;
    break;
    case 380:
            m[0].valor=1;
            m[1].valor=17073;
            m[2].valor=34687;
            m[3].valor=52879;
            m[4].valor=71688;
            m[5].valor=91156;
            m[6].valor=111333;
            m[7].valor=132271;
            m[8].valor=154031;
            m[9].valor=176679;
            m[10].valor=200291;
            m[11].valor=224953;
            m[12].valor=250762;
            m[13].valor=277831;
            m[14].valor=306289;
            m[15].valor=336285;
            m[16].valor=367997;
            m[17].valor=401631;
            m[18].valor=437437;
            m[19].valor=475714;
            m[20].valor=516829;
            m[21].valor=561237;
            m[22].valor=609511;
            m[23].valor=662389;
            m[24].valor=720843;
            m[25].valor=786189;
            m[26].valor=860272;
            m[27].valor=945795;
            m[28].valor=1046947;
            m[29].valor=1170747;
            m[30].valor=1330353;
            m[31].valor=1555305;
            m[32].valor=1939863;
    break;
    case 390:
            m[0].valor=1;
            m[1].valor=17522;
            m[2].valor=35600;
            m[3].valor=54271;
            m[4].valor=73574;
            m[5].valor=93555;
            m[6].valor=114263;
            m[7].valor=135752;
            m[8].valor=158085;
            m[9].valor=181329;
            m[10].valor=205562;
            m[11].valor=230873;
            m[12].valor=257361;
            m[13].valor=285142;
            m[14].valor=314349;
            m[15].valor=345135;
            m[16].valor=377681;
            m[17].valor=412200;
            m[18].valor=448949;
            m[19].valor=488233;
            m[20].valor=530430;
            m[21].valor=576007;
            m[22].valor=625551;
            m[23].valor=679820;
            m[24].valor=739813;
            m[25].valor=806878;
            m[26].valor=882911;
            m[27].valor=970685;
            m[28].valor=1074498;
            m[29].valor=1201556;
            m[30].valor=1365363;
            m[31].valor=1596234;
            m[32].valor=1990912;
    break;
    case 400:
            m[0].valor=1;
            m[1].valor=17972;
            m[2].valor=36513;
            m[3].valor=55662;
            m[4].valor=75461;
            m[5].valor=95954;
            m[6].valor=117193;
            m[7].valor=139233;
            m[8].valor=162138;
            m[9].valor=185978;
            m[10].valor=210833;
            m[11].valor=236793;
            m[12].valor=263960;
            m[13].valor=292454;
            m[14].valor=322409;
            m[15].valor=353984;
            m[16].valor=387365;
            m[17].valor=422770;
            m[18].valor=460460;
            m[19].valor=500752;
            m[20].valor=544031;
            m[21].valor=590776;
            m[22].valor=641591;
            m[23].valor=697252;
            m[24].valor=758782;
            m[25].valor=827568;
            m[26].valor=905550;
            m[27].valor=995574;
            m[28].valor=1102050;
            m[29].valor=1232366;
            m[30].valor=1400372;
            m[31].valor=1637163;
            m[32].valor=2041961;
    break;
    case 410:
            m[0].valor=1;
            m[1].valor=18421;
            m[2].valor=37426;
            m[3].valor=57054;
            m[4].valor=77347;
            m[5].valor=98353;
            m[6].valor=120122;
            m[7].valor=142714;
            m[8].valor=166191;
            m[9].valor=190627;
            m[10].valor=216104;
            m[11].valor=242712;
            m[12].valor=270559;
            m[13].valor=299765;
            m[14].valor=330469;
            m[15].valor=362834;
            m[16].valor=397049;
            m[17].valor=433339;
            m[18].valor=471972;
            m[19].valor=513271;
            m[20].valor=557632;
            m[21].valor=605545;
            m[22].valor=657630;
            m[23].valor=714683;
            m[24].valor=777752;
            m[25].valor=848257;
            m[26].valor=928189;
            m[27].valor=1020463;
            m[28].valor=1129601;
            m[29].valor=1263175;
            m[30].valor=1435381;
```

```
            m[31].valor=1678093;
            m[32].valor=2093010;
    break;
    case 420:
            m[0].valor=1;
            m[1].valor=18870;
            m[2].valor=38338;
            m[3].valor=58445;
            m[4].valor=79234;
            m[5].valor=100752;
            m[6].valor=123052;
            m[7].valor=146195;
            m[8].valor=170245;
            m[9].valor=195277;
            m[10].valor=221374;
            m[11].valor=248632;
            m[12].valor=277158;
            m[13].valor=307076;
            m[14].valor=338529;
            m[15].valor=371683;
            m[16].valor=406733;
            m[17].valor=443908;
            m[18].valor=483483;
            m[19].valor=525789;
            m[20].valor=571232;
            m[21].valor=620315;
            m[22].valor=673670;
            m[23].valor=732114;
            m[24].valor=796721;
            m[25].valor=868946;
            m[26].valor=950827;
            m[27].valor=1045353;
            m[28].valor=1157152;
            m[29].valor=1293984;
            m[30].valor=1470390;
            m[31].valor=1719022;
            m[32].valor=2144059;
    break;
    case 430:
            m[0].valor=1;
            m[1].valor=19319;
            m[2].valor=39251;
            m[3].valor=59837;
            m[4].valor=81120;
            m[5].valor=103150;
            m[6].valor=125982;
            m[7].valor=149675;
            m[8].valor=174298;
            m[9].valor=199926;
            m[10].valor=226645;
            m[11].valor=254552;
            m[12].valor=283757;
            m[13].valor=314388;
            m[14].valor=346590;
            m[15].valor=380533;
            m[16].valor=416417;
            m[17].valor=454477;
            m[18].valor=494995;
            m[19].valor=538308;
            m[20].valor=584833;
            m[21].valor=635084;
            m[22].valor=689710;
            m[23].valor=749546;
            m[24].valor=815691;
            m[25].valor=889635;
            m[26].valor=973466;
            m[27].valor=1070242;
            m[28].valor=1184703;
            m[29].valor=1324793;
            m[30].valor=1505400;
            m[31].valor=1759951;
            m[32].valor=2195108;
    break;
    case 440:
            m[0].valor=1;
            m[1].valor=19769;
            m[2].valor=40164;
            m[3].valor=61228;
            m[4].valor=83007;
            m[5].valor=105549;
            m[6].valor=128912;
            m[7].valor=153156;
            m[8].valor=178352;
            m[9].valor=204576;
            m[10].valor=231916;
            m[11].valor=260472;
            m[12].valor=290356;
            m[13].valor=321699;
            m[14].valor=354650;
            m[15].valor=389383;
            m[16].valor=426101;
            m[17].valor=465046;
            m[18].valor=506506;
            m[19].valor=550827;
            m[20].valor=598434;
            m[21].valor=649853;
            m[22].valor=705750;
            m[23].valor=766977;
            m[24].valor=834660;
            m[25].valor=910324;
            m[26].valor=996105;
            m[27].valor=1095131;
            m[28].valor=1212255;
            m[29].valor=1355602;
            m[30].valor=1540409;
            m[31].valor=1800880;
            m[32].valor=2246157;
    break;
    case 450:
            m[0].valor=1;
            m[1].valor=20218;
            m[2].valor=41077;
            m[3].valor=62620;
            m[4].valor=84893;
            m[5].valor=107948;
            m[6].valor=131842;
            m[7].valor=156637;
            m[8].valor=182405;
            m[9].valor=209225;
            m[10].valor=237187;
            m[11].valor=266392;
            m[12].valor=296955;
            m[13].valor=329010;
            m[14].valor=362710;
            m[15].valor=398232;
            m[16].valor=435785;
            m[17].valor=475616;
            m[18].valor=518017;
            m[19].valor=563346;
            m[20].valor=612035;
            m[21].valor=664623;
            m[22].valor=721789;
            m[23].valor=784408;
            m[24].valor=853630;
            m[25].valor=931013;
            m[26].valor=1018743;
            m[27].valor=1120020;
            m[28].valor=1239806;
            m[29].valor=1386411;
            m[30].valor=1575418;
            m[31].valor=1841809;
            m[32].valor=2297206;
    break;
    case 460:
            m[0].valor=1;
            m[1].valor=20667;
```

```
            m[2].valor=41990;
            m[3].valor=64011;
            m[4].valor=86780;
            m[5].valor=110347;
            m[6].valor=134771;
            m[7].valor=160118;
            m[8].valor=186458;
            m[9].valor=213875;
            m[10].valor=242458;
            m[11].valor=272311;
            m[12].valor=303554;
            m[13].valor=336322;
            m[14].valor=370770;
            m[15].valor=407082;
            m[16].valor=445469;
            m[17].valor=486185;
            m[18].valor=529529;
            m[19].valor=575865;
            m[20].valor=625635;
            m[21].valor=679392;
            m[22].valor=737829;
            m[23].valor=801839;
            m[24].valor=872599;
            m[25].valor=951703;
            m[26].valor=1041382;
            m[27].valor=1144910;
            m[28].valor=1267357;
            m[29].valor=1417220;
            m[30].valor=1610427;
            m[31].valor=1882738;
            m[32].valor=2348255;
        break;
        case 470:
            m[0].valor=1;
            m[1].valor=21117;
            m[2].valor=42902;
            m[3].valor=65403;
            m[4].valor=88666;
            m[5].valor=112746;
            m[6].valor=137701;
            m[7].valor=163599;
            m[8].valor=190512;
            m[9].valor=218524;
            m[10].valor=247728;
            m[11].valor=278231;
            m[12].valor=310153;
            m[13].valor=343633;
            m[14].valor=378830;
            m[15].valor=415931;
            m[16].valor=455153;
            m[17].valor=496754;
            m[18].valor=541040;
            m[19].valor=588383;
            m[20].valor=639236;
            m[21].valor=694162;
            m[22].valor=753869;
            m[23].valor=819271;
            m[24].valor=891569;
            m[25].valor=972392;
            m[26].valor=1064021;
            m[27].valor=1169799;
            m[28].valor=1294908;
            m[29].valor=1448029;
            m[30].valor=1645437;
            m[31].valor=1923667;
            m[32].valor=2399304;
        break;
        case 480:
            m[0].valor=1;
            m[1].valor=21566;
            m[2].valor=43815;
            m[3].valor=66794;
            m[4].valor=90553;
            m[5].valor=115145;
            m[6].valor=140631;
            m[7].valor=167079;
            m[8].valor=194565;
            m[9].valor=223173;
            m[10].valor=252999;
            m[11].valor=284151;
            m[12].valor=316752;
            m[13].valor=350944;
            m[14].valor=386891;
            m[15].valor=424781;
            m[16].valor=464838;
            m[17].valor=507323;
            m[18].valor=552552;
            m[19].valor=600902;
            m[20].valor=652837;
            m[21].valor=708931;
            m[22].valor=769908;
            m[23].valor=836702;
            m[24].valor=910539;
            m[25].valor=993081;
            m[26].valor=1086660;
            m[27].valor=1194688;
            m[28].valor=1322459;
            m[29].valor=1478838;
            m[30].valor=1680446;
            m[31].valor=1964596;
            m[32].valor=2450353;
        break;
        case 490:
            m[0].valor=1;
            m[1].valor=22015;
            m[2].valor=44728;
            m[3].valor=68186;
            m[4].valor=92439;
            m[5].valor=117543;
            m[6].valor=143561;
            m[7].valor=170560;
            m[8].valor=198619;
            m[9].valor=227823;
            m[10].valor=258270;
            m[11].valor=290071;
            m[12].valor=323351;
            m[13].valor=358256;
            m[14].valor=394951;
            m[15].valor=433631;
            m[16].valor=474522;
            m[17].valor=517893;
            m[18].valor=564063;
            m[19].valor=613421;
            m[20].valor=666438;
            m[21].valor=723700;
            m[22].valor=785948;
            m[23].valor=854133;
            m[24].valor=929508;
            m[25].valor=1013770;
            m[26].valor=1109298;
            m[27].valor=1219578;
            m[28].valor=1350011;
            m[29].valor=1509648;
            m[30].valor=1715455;
            m[31].valor=2005525;
            m[32].valor=2501403;
        break;
        case 500:
            m[0].valor=1;
            m[1].valor=22464;
            m[2].valor=45641;
            m[3].valor=69577;
            m[4].valor=94326;
            m[5].valor=119942;
            m[6].valor=146491;
            m[7].valor=174041;
```

```
            m[8].valor=202672;
            m[9].valor=232472;
            m[10].valor=263541;
            m[11].valor=295991;
            m[12].valor=329950;
            m[13].valor=365567;
            m[14].valor=403011;
            m[15].valor=442480;
            m[16].valor=484206;
            m[17].valor=528462;
            m[18].valor=575575;
            m[19].valor=625940;
            m[20].valor=680038;
            m[21].valor=738470;
            m[22].valor=801988;
            m[23].valor=871564;
            m[24].valor=948478;
            m[25].valor=1034459;
            m[26].valor=1131937;
            m[27].valor=1244467;
            m[28].valor=1377562;
            m[29].valor=1540457;
            m[30].valor=1750465;
            m[31].valor=2046454;
            m[32].valor=2552452;
        break;
        case 510:
            m[0].valor=1;
            m[1].valor=22914;
            m[2].valor=46554;
            m[3].valor=70969;
            m[4].valor=96212;
            m[5].valor=122341;
            m[6].valor=149420;
            m[7].valor=177522;
            m[8].valor=206726;
            m[9].valor=237122;
            m[10].valor=268812;
            m[11].valor=301910;
            m[12].valor=336549;
            m[13].valor=372878;
            m[14].valor=411071;
            m[15].valor=451330;
            m[16].valor=493890;
            m[17].valor=539031;
            m[18].valor=587086;
            m[19].valor=638458;
            m[20].valor=693639;
            m[21].valor=753239;
            m[22].valor=818028;
            m[23].valor=888996;
            m[24].valor=967447;
            m[25].valor=1055148;
            m[26].valor=1154576;
            m[27].valor=1269356;
            m[28].valor=1405113;
            m[29].valor=1571266;
            m[30].valor=1785474;
            m[31].valor=2087383;
            m[32].valor=2603501;
        break;
        case 520:
            m[0].valor=1;
            m[1].valor=23363;
            m[2].valor=47466;
            m[3].valor=72360;
            m[4].valor=98099;
            m[5].valor=124740;
            m[6].valor=152350;
            m[7].valor=181003;
            m[8].valor=210779;
            m[9].valor=241771;
            m[10].valor=274082;
            m[11].valor=307830;
            m[12].valor=343148;
            m[13].valor=380190;
            m[14].valor=419131;
            m[15].valor=460179;
            m[16].valor=503574;
            m[17].valor=549600;
            m[18].valor=598598;
            m[19].valor=650977;
            m[20].valor=707240;
            m[21].valor=768008;
            m[22].valor=834067;
            m[23].valor=906427;
            m[24].valor=986417;
            m[25].valor=1075838;
            m[26].valor=1177215;
            m[27].valor=1294246;
            m[28].valor=1432664;
            m[29].valor=1602075;
            m[30].valor=1820483;
            m[31].valor=2128312;
            m[32].valor=2654550;
        break;
        case 530:
            m[0].valor=1;
            m[1].valor=23812;
            m[2].valor=48379;
            m[3].valor=73752;
            m[4].valor=99985;
            m[5].valor=127139;
            m[6].valor=155280;
            m[7].valor=184483;
            m[8].valor=214832;
            m[9].valor=246420;
            m[10].valor=279353;
            m[11].valor=313750;
            m[12].valor=349747;
            m[13].valor=387501;
            m[14].valor=427192;
            m[15].valor=469029;
            m[16].valor=513258;
            m[17].valor=560169;
            m[18].valor=610109;
            m[19].valor=663496;
            m[20].valor=720841;
            m[21].valor=782778;
            m[22].valor=850107;
            m[23].valor=923858;
            m[24].valor=1005386;
            m[25].valor=1096527;
            m[26].valor=1199853;
            m[27].valor=1319135;
            m[28].valor=1460215;
            m[29].valor=1632884;
            m[30].valor=1855492;
            m[31].valor=2169241;
            m[32].valor=2705599;
        break;
        case 540:
            m[0].valor=1;
            m[1].valor=24261;
            m[2].valor=49292;
            m[3].valor=75144;
            m[4].valor=101872;
            m[5].valor=129538;
            m[6].valor=158210;
            m[7].valor=187964;
            m[8].valor=218886;
            m[9].valor=251070;
            m[10].valor=284624;
            m[11].valor=319670;
            m[12].valor=356346;
            m[13].valor=394812;
```

```
            m[14].valor=435252;
            m[15].valor=477878;
            m[16].valor=522942;
            m[17].valor=570739;
            m[18].valor=621621;
            m[19].valor=676015;
            m[20].valor=734441;
            m[21].valor=797547;
            m[22].valor=866147;
            m[23].valor=941289;
            m[24].valor=1024356;
            m[25].valor=1117216;
            m[26].valor=1222492;
            m[27].valor=1344024;
            m[28].valor=1487767;
            m[29].valor=1663693;
            m[30].valor=1890502;
            m[31].valor=2210170;
            m[32].valor=2756648;
        break;
        case 550:
            m[0].valor=1;
            m[1].valor=24711;
            m[2].valor=50205;
            m[3].valor=76535;
            m[4].valor=103758;
            m[5].valor=131936;
            m[6].valor=161140;
            m[7].valor=191445;
            m[8].valor=222939;
            m[9].valor=255719;
            m[10].valor=289895;
            m[11].valor=325589;
            m[12].valor=362945;
            m[13].valor=402124;
            m[14].valor=443312;
            m[15].valor=486728;
            m[16].valor=532626;
            m[17].valor=581308;
            m[18].valor=633132;
            m[19].valor=688534;
            m[20].valor=748042;
            m[21].valor=812317;
            m[22].valor=882187;
            m[23].valor=958721;
            m[24].valor=1043325;
            m[25].valor=1137905;
            m[26].valor=1245131;
            m[27].valor=1368914;
            m[28].valor=1515318;
            m[29].valor=1694502;
            m[30].valor=1925511;
            m[31].valor=2251099;
            m[32].valor=2807697;
        break;
        case 560:
            m[0].valor=1;
            m[1].valor=25160;
            m[2].valor=51118;
            m[3].valor=77927;
            m[4].valor=105645;
            m[5].valor=134335;
            m[6].valor=164069;
            m[7].valor=194926;
            m[8].valor=226993;
            m[9].valor=260369;
            m[10].valor=295166;
            m[11].valor=331509;
            m[12].valor=369544;
            m[13].valor=409435;
            m[14].valor=451372;
            m[15].valor=495578;
            m[16].valor=542310;
            m[17].valor=591877;
            m[18].valor=644644;
            m[19].valor=701052;
            m[20].valor=761643;
            m[21].valor=827086;
            m[22].valor=898226;
            m[23].valor=976152;
            m[24].valor=1062295;
            m[25].valor=1158594;
            m[26].valor=1267769;
            m[27].valor=1393803;
            m[28].valor=1542869;
            m[29].valor=1725311;
            m[30].valor=1960520;
            m[31].valor=2292028;
            m[32].valor=2858746;
        break;
        case 570:
            m[0].valor=1;
            m[1].valor=25609;
            m[2].valor=52030;
            m[3].valor=79318;
            m[4].valor=107531;
            m[5].valor=136734;
            m[6].valor=166999;
            m[7].valor=198407;
            m[8].valor=231046;
            m[9].valor=265018;
            m[10].valor=300436;
            m[11].valor=337429;
            m[12].valor=376143;
            m[13].valor=416746;
            m[14].valor=459432;
            m[15].valor=504427;
            m[16].valor=551994;
            m[17].valor=602446;
            m[18].valor=656155;
            m[19].valor=713571;
            m[20].valor=775244;
            m[21].valor=841855;
            m[22].valor=914266;
            m[23].valor=993583;
            m[24].valor=1081264;
            m[25].valor=1179283;
            m[26].valor=1290408;
            m[27].valor=1418692;
            m[28].valor=1570420;
            m[29].valor=1756120;
            m[30].valor=1995529;
            m[31].valor=2332958;
            m[32].valor=2909795;
        break;
        case 580:
            m[0].valor=1;
            m[1].valor=26058;
            m[2].valor=52943;
            m[3].valor=80710;
            m[4].valor=109417;
            m[5].valor=139133;
            m[6].valor=169929;
            m[7].valor=201887;
            m[8].valor=235100;
            m[9].valor=269668;
            m[10].valor=305707;
            m[11].valor=343349;
            m[12].valor=382742;
            m[13].valor=424058;
            m[14].valor=467493;
            m[15].valor=513277;
            m[16].valor=561679;
            m[17].valor=613015;
            m[18].valor=667667;
            m[19].valor=726090;
```

```
            m[20].valor=788844;
            m[21].valor=856625;
            m[22].valor=930306;
            m[23].valor=1011015;
            m[24].valor=1100234;
            m[25].valor=1199973;
            m[26].valor=1313047;
            m[27].valor=1443582;
            m[28].valor=1597972;
            m[29].valor=1786930;
            m[30].valor=2030539;
            m[31].valor=2373887;
            m[32].valor=2960844;
        break;
        case 590:
            m[0].valor=1;
            m[1].valor=26508;
            m[2].valor=53856;
            m[3].valor=82101;
            m[4].valor=111304;
            m[5].valor=141532;
            m[6].valor=172859;
            m[7].valor=205368;
            m[8].valor=239153;
            m[9].valor=274317;
            m[10].valor=310978;
            m[11].valor=349269;
            m[12].valor=389341;
            m[13].valor=431369;
            m[14].valor=475553;
            m[15].valor=522126;
            m[16].valor=571363;
            m[17].valor=623585;
            m[18].valor=679178;
            m[19].valor=738609;
            m[20].valor=802445;
            m[21].valor=871394;
            m[22].valor=946346;
            m[23].valor=1028446;
            m[24].valor=1119203;
            m[25].valor=1220662;
            m[26].valor=1335686;
            m[27].valor=1468471;
            m[28].valor=1625523;
            m[29].valor=1817739;
            m[30].valor=2065548;
            m[31].valor=2414816;
            m[32].valor=3011893;
        break;
        case 600:
            m[0].valor=1;
            m[1].valor=26957;
            m[2].valor=54769;
            m[3].valor=83493;
            m[4].valor=113190;
            m[5].valor=143930;
            m[6].valor=175789;
            m[7].valor=208849;
            m[8].valor=243206;
            m[9].valor=278966;
            m[10].valor=316249;
            m[11].valor=355188;
            m[12].valor=395940;
            m[13].valor=438680;
            m[14].valor=483613;
            m[15].valor=530976;
            m[16].valor=581047;
            m[17].valor=634154;
            m[18].valor=690690;
            m[19].valor=751127;
            m[20].valor=816046;
            m[21].valor=886163;
            m[22].valor=962385;
            m[23].valor=1045877;
            m[24].valor=1138173;
            m[25].valor=1241351;
            m[26].valor=1358324;
            m[27].valor=1493360;
            m[28].valor=1653074;
            m[29].valor=1848548;
            m[30].valor=2100557;
            m[31].valor=2455745;
            m[32].valor=3062942;
        break;
        case 610:
            m[0].valor=1;
            m[1].valor=27406;
            m[2].valor=55682;
            m[3].valor=84884;
            m[4].valor=115077;
            m[5].valor=146329;
            m[6].valor=178718;
            m[7].valor=212330;
            m[8].valor=247260;
            m[9].valor=283616;
            m[10].valor=321519;
            m[11].valor=361108;
            m[12].valor=402539;
            m[13].valor=445991;
            m[14].valor=491673;
            m[15].valor=539826;
            m[16].valor=590731;
            m[17].valor=644723;
            m[18].valor=702201;
            m[19].valor=763646;
            m[20].valor=829647;
            m[21].valor=900933;
            m[22].valor=978425;
            m[23].valor=1063308;
            m[24].valor=1157142;
            m[25].valor=1262040;
            m[26].valor=1380963;
            m[27].valor=1518250;
            m[28].valor=1680625;
            m[29].valor=1879357;
            m[30].valor=2135567;
            m[31].valor=2496674;
            m[32].valor=3113991;
        break;
        case 620:
            m[0].valor=1;
            m[1].valor=27856;
            m[2].valor=56594;
            m[3].valor=86276;
            m[4].valor=116963;
            m[5].valor=148728;
            m[6].valor=181648;
            m[7].valor=215811;
            m[8].valor=251313;
            m[9].valor=288265;
            m[10].valor=326790;
            m[11].valor=367028;
            m[12].valor=409138;
            m[13].valor=453303;
            m[14].valor=499733;
            m[15].valor=548675;
            m[16].valor=600415;
            m[17].valor=655292;
            m[18].valor=713713;
            m[19].valor=776165;
            m[20].valor=843247;
            m[21].valor=915702;
            m[22].valor=994465;
            m[23].valor=1080740;
            m[24].valor=1176112;
            m[25].valor=1282729;
```

```
            m[26].valor=1403602;
            m[27].valor=1543139;
            m[28].valor=1708176;
            m[29].valor=1910166;
            m[30].valor=2170576;
            m[31].valor=2537603;
            m[32].valor=3165040;
        break;
        case 630:
            m[0].valor=1;
            m[1].valor=28305;
            m[2].valor=57507;
            m[3].valor=87667;
            m[4].valor=118850;
            m[5].valor=151127;
            m[6].valor=184578;
            m[7].valor=219291;
            m[8].valor=255367;
            m[9].valor=292915;
            m[10].valor=332061;
            m[11].valor=372948;
            m[12].valor=415737;
            m[13].valor=460614;
            m[14].valor=507794;
            m[15].valor=557525;
            m[16].valor=610099;
            m[17].valor=665862;
            m[18].valor=725224;
            m[19].valor=788684;
            m[20].valor=856848;
            m[21].valor=930472;
            m[22].valor=1010505;
            m[23].valor=1098171;
            m[24].valor=1195081;
            m[25].valor=1303418;
            m[26].valor=1426240;
            m[27].valor=1568028;
            m[28].valor=1735728;
            m[29].valor=1940975;
            m[30].valor=2205585;
            m[31].valor=2578532;
            m[32].valor=3216089;
        break;
        case 640:
            m[0].valor=1;
            m[1].valor=28754;
            m[2].valor=58420;
            m[3].valor=89059;
            m[4].valor=120736;
            m[5].valor=153526;
            m[6].valor=187508;
            m[7].valor=222772;
            m[8].valor=259420;
            m[9].valor=297564;
            m[10].valor=337332;
            m[11].valor=378868;
            m[12].valor=422336;
            m[13].valor=467925;
            m[14].valor=515854;
            m[15].valor=566374;
            m[16].valor=619783;
            m[17].valor=676431;
            m[18].valor=736736;
            m[19].valor=801202;
            m[20].valor=870449;
            m[21].valor=945241;
            m[22].valor=1026544;
            m[23].valor=1115602;
            m[24].valor=1214051;
            m[25].valor=1324107;
            m[26].valor=1448879;
            m[27].valor=1592918;
            m[28].valor=1763279;
            m[29].valor=1971784;
            m[30].valor=2240594;
            m[31].valor=2619461;
            m[32].valor=3267138;
        break;
        case 650:
            m[0].valor=1;
            m[1].valor=29203;
            m[2].valor=59333;
            m[3].valor=90450;
            m[4].valor=122623;
            m[5].valor=155925;
            m[6].valor=190437;
            m[7].valor=226253;
            m[8].valor=263474;
            m[9].valor=302214;
            m[10].valor=342603;
            m[11].valor=384787;
            m[12].valor=428935;
            m[13].valor=475237;
            m[14].valor=523914;
            m[15].valor=575224;
            m[16].valor=629467;
            m[17].valor=687000;
            m[18].valor=748247;
            m[19].valor=813721;
            m[20].valor=884050;
            m[21].valor=960010;
            m[22].valor=1042584;
            m[23].valor=1133033;
            m[24].valor=1233021;
            m[25].valor=1344797;
            m[26].valor=1471518;
            m[27].valor=1617807;
            m[28].valor=1790830;
            m[29].valor=2002593;
            m[30].valor=2275604;
            m[31].valor=2660390;
            m[32].valor=3318187;
        break;
        case 660:
            m[0].valor=1;
            m[1].valor=29653;
            m[2].valor=60246;
            m[3].valor=91842;
            m[4].valor=124509;
            m[5].valor=158323;
            m[6].valor=193367;
            m[7].valor=229734;
            m[8].valor=267527;
            m[9].valor=306863;
            m[10].valor=347873;
            m[11].valor=390707;
            m[12].valor=435534;
            m[13].valor=482548;
            m[14].valor=531974;
            m[15].valor=584073;
            m[16].valor=639151;
            m[17].valor=697569;
            m[18].valor=759759;
            m[19].valor=826240;
            m[20].valor=897650;
            m[21].valor=974780;
            m[22].valor=1058624;
            m[23].valor=1150465;
            m[24].valor=1251990;
            m[25].valor=1365486;
            m[26].valor=1494157;
            m[27].valor=1642696;
            m[28].valor=1818381;
            m[29].valor=2033402;
            m[30].valor=2310613;
            m[31].valor=2701319;
```

```
            m[32].valor=3369236;
break;
case 670:
            m[0].valor=1;
            m[1].valor=30102;
            m[2].valor=61158;
            m[3].valor=93233;
            m[4].valor=126396;
            m[5].valor=160722;
            m[6].valor=196297;
            m[7].valor=233215;
            m[8].valor=271580;
            m[9].valor=311512;
            m[10].valor=353144;
            m[11].valor=396627;
            m[12].valor=442133;
            m[13].valor=489859;
            m[14].valor=540034;
            m[15].valor=592923;
            m[16].valor=648835;
            m[17].valor=708138;
            m[18].valor=771270;
            m[19].valor=838759;
            m[20].valor=911251;
            m[21].valor=989549;
            m[22].valor=1074664;
            m[23].valor=1167896;
            m[24].valor=1270960;
            m[25].valor=1386175;
            m[26].valor=1516795;
            m[27].valor=1667586;
            m[28].valor=1845933;
            m[29].valor=2064212;
            m[30].valor=2345622;
            m[31].valor=2742248;
            m[32].valor=3420285;
break;
case 680:
            m[0].valor=1;
            m[1].valor=30551;
            m[2].valor=62071;
            m[3].valor=94625;
            m[4].valor=128282;
            m[5].valor=163121;
            m[6].valor=199227;
            m[7].valor=236695;
            m[8].valor=275634;
            m[9].valor=316162;
            m[10].valor=358415;
            m[11].valor=402547;
            m[12].valor=448732;
            m[13].valor=497171;
            m[14].valor=548095;
            m[15].valor=601773;
            m[16].valor=658519;
            m[17].valor=718708;
            m[18].valor=782781;
            m[19].valor=851278;
            m[20].valor=924852;
            m[21].valor=1004318;
            m[22].valor=1090703;
            m[23].valor=1185327;
            m[24].valor=1289929;
            m[25].valor=1406864;
            m[26].valor=1539434;
            m[27].valor=1692475;
            m[28].valor=1873484;
            m[29].valor=2095021;
            m[30].valor=2380631;
            m[31].valor=2783177;
            m[32].valor=3471334;
break;
case 690:
            m[0].valor=1;
            m[1].valor=31000;
            m[2].valor=62984;
            m[3].valor=96016;
            m[4].valor=130169;
            m[5].valor=165520;
            m[6].valor=202157;
            m[7].valor=240176;
            m[8].valor=279687;
            m[9].valor=320811;
            m[10].valor=363686;
            m[11].valor=408467;
            m[12].valor=455331;
            m[13].valor=504482;
            m[14].valor=556155;
            m[15].valor=610622;
            m[16].valor=668204;
            m[17].valor=729277;
            m[18].valor=794293;
            m[19].valor=863796;
            m[20].valor=938453;
            m[21].valor=1019088;
            m[22].valor=1106743;
            m[23].valor=1202758;
            m[24].valor=1308899;
            m[25].valor=1427553;
            m[26].valor=1562073;
            m[27].valor=1717364;
            m[28].valor=1901035;
            m[29].valor=2125830;
            m[30].valor=2415641;
            m[31].valor=2824106;
            m[32].valor=3522383;
break;
case 700:
            m[0].valor=1;
            m[1].valor=31450;
            m[2].valor=63897;
            m[3].valor=97408;
            m[4].valor=132055;
            m[5].valor=167919;
            m[6].valor=205086;
            m[7].valor=243657;
            m[8].valor=283741;
            m[9].valor=325461;
            m[10].valor=368957;
            m[11].valor=414386;
            m[12].valor=461930;
            m[13].valor=511793;
            m[14].valor=564215;
            m[15].valor=619472;
            m[16].valor=677888;
            m[17].valor=739846;
            m[18].valor=805804;
            m[19].valor=876315;
            m[20].valor=952053;
            m[21].valor=1033857;
            m[22].valor=1122783;
            m[23].valor=1220190;
            m[24].valor=1327868;
            m[25].valor=1448242;
            m[26].valor=1584712;
            m[27].valor=1742254;
            m[28].valor=1928586;
            m[29].valor=2156639;
            m[30].valor=2450650;
            m[31].valor=2865035;
            m[32].valor=3573432;
break;
case 710:
            m[0].valor=1;
            m[1].valor=31899;
            m[2].valor=64810;
```

```
            m[3].valor=98800;
            m[4].valor=133942;
            m[5].valor=170318;
            m[6].valor=208016;
            m[7].valor=247138;
            m[8].valor=287794;
            m[9].valor=330110;
            m[10].valor=374227;
            m[11].valor=420306;
            m[12].valor=468529;
            m[13].valor=519105;
            m[14].valor=572275;
            m[15].valor=628321;
            m[16].valor=687572;
            m[17].valor=750415;
            m[18].valor=817316;
            m[19].valor=888834;
            m[20].valor=965654;
            m[21].valor=1048627;
            m[22].valor=1138822;
            m[23].valor=1237621;
            m[24].valor=1346838;
            m[25].valor=1468932;
            m[26].valor=1607350;
            m[27].valor=1767143;
            m[28].valor=1956137;
            m[29].valor=2187448;
            m[30].valor=2485659;
            m[31].valor=2905964;
            m[32].valor=3624481;
        break;
        case 720:
            m[0].valor=1;
            m[1].valor=32348;
            m[2].valor=65722;
            m[3].valor=100191;
            m[4].valor=135828;
            m[5].valor=172716;
            m[6].valor=210946;
            m[7].valor=250619;
            m[8].valor=291847;
            m[9].valor=334760;
            m[10].valor=379498;
            m[11].valor=426226;
            m[12].valor=475128;
            m[13].valor=526416;
            m[14].valor=580335;
            m[15].valor=637171;
            m[16].valor=697256;
            m[17].valor=760984;
            m[18].valor=828827;
            m[19].valor=901353;
            m[20].valor=979255;
            m[21].valor=1063396;
            m[22].valor=1154862;
            m[23].valor=1255052;
            m[24].valor=1365807;
            m[25].valor=1489621;
            m[26].valor=1629989;
            m[27].valor=1792032;
            m[28].valor=1983689;
            m[29].valor=2218257;
            m[30].valor=2520669;
            m[31].valor=2946893;
            m[32].valor=3675530;
        break;
        case 730:
            m[0].valor=1;
            m[1].valor=32797;
            m[2].valor=66635;
            m[3].valor=101583;
            m[4].valor=137715;
            m[5].valor=175115;
            m[6].valor=213876;
            m[7].valor=254099;
            m[8].valor=295901;
            m[9].valor=339409;
            m[10].valor=384769;
            m[11].valor=432146;
            m[12].valor=481727;
            m[13].valor=533727;
            m[14].valor=588396;
            m[15].valor=646021;
            m[16].valor=706940;
            m[17].valor=771554;
            m[18].valor=840339;
            m[19].valor=913871;
            m[20].valor=992856;
            m[21].valor=1078165;
            m[22].valor=1170902;
            m[23].valor=1272484;
            m[24].valor=1384777;
            m[25].valor=1510310;
            m[26].valor=1652628;
            m[27].valor=1816922;
            m[28].valor=2011240;
            m[29].valor=2249066;
            m[30].valor=2555678;
            m[31].valor=2987822;
            m[32].valor=3726579;
        break;
        case 740:
            m[0].valor=1;
            m[1].valor=33247;
            m[2].valor=67548;
            m[3].valor=102974;
            m[4].valor=139601;
            m[5].valor=177514;
            m[6].valor=216806;
            m[7].valor=257580;
            m[8].valor=299954;
            m[9].valor=344058;
            m[10].valor=390040;
            m[11].valor=438066;
            m[12].valor=488326;
            m[13].valor=541039;
            m[14].valor=596456;
            m[15].valor=654870;
            m[16].valor=716624;
            m[17].valor=782123;
            m[18].valor=851850;
            m[19].valor=926390;
            m[20].valor=1006456;
            m[21].valor=1092935;
            m[22].valor=1186942;
            m[23].valor=1289915;
            m[24].valor=1403746;
            m[25].valor=1530999;
            m[26].valor=1675266;
            m[27].valor=1841811;
            m[28].valor=2038791;
            m[29].valor=2279875;
            m[30].valor=2590687;
            m[31].valor=3028752;
            m[32].valor=3777628;
        break;
        case 750:
            m[0].valor=1;
            m[1].valor=33696;
            m[2].valor=68461;
            m[3].valor=104366;
            m[4].valor=141488;
            m[5].valor=179913;
            m[6].valor=219735;
            m[7].valor=261061;
            m[8].valor=304008;
```

```
            m[9].valor=348708;
            m[10].valor=395311;
            m[11].valor=443985;
            m[12].valor=494925;
            m[13].valor=548350;
            m[14].valor=604516;
            m[15].valor=663720;
            m[16].valor=726308;
            m[17].valor=792692;
            m[18].valor=863362;
            m[19].valor=938909;
            m[20].valor=1020057;
            m[21].valor=1107704;
            m[22].valor=1202981;
            m[23].valor=1307346;
            m[24].valor=1422716;
            m[25].valor=1551688;
            m[26].valor=1697905;
            m[27].valor=1866700;
            m[28].valor=2066342;
            m[29].valor=2310684;
            m[30].valor=2625696;
            m[31].valor=3069681;
            m[32].valor=3828677;
        break;
        case 760:
            m[0].valor=1;
            m[1].valor=34145;
            m[2].valor=69374;
            m[3].valor=105757;
            m[4].valor=143374;
            m[5].valor=182312;
            m[6].valor=222665;
            m[7].valor=264542;
            m[8].valor=308061;
            m[9].valor=353357;
            m[10].valor=400581;
            m[11].valor=449905;
            m[12].valor=501524;
            m[13].valor=555661;
            m[14].valor=612576;
            m[15].valor=672569;
            m[16].valor=735992;
            m[17].valor=803261;
            m[18].valor=874873;
            m[19].valor=951428;
            m[20].valor=1033658;
            m[21].valor=1122473;
            m[22].valor=1219021;
            m[23].valor=1324777;
            m[24].valor=1441685;
            m[25].valor=1572377;
            m[26].valor=1720544;
            m[27].valor=1891589;
            m[28].valor=2093893;
            m[29].valor=2341494;
            m[30].valor=2660706;
            m[31].valor=3110610;
            m[32].valor=3879726;
        break;
        case 770:
            m[0].valor=1;
            m[1].valor=34594;
            m[2].valor=70286;
            m[3].valor=107149;
            m[4].valor=145261;
            m[5].valor=184710;
            m[6].valor=225595;
            m[7].valor=268023;
            m[8].valor=312115;
            m[9].valor=358007;
            m[10].valor=405852;
            m[11].valor=455825;
            m[12].valor=508123;
            m[13].valor=562973;
            m[14].valor=620637;
            m[15].valor=681419;
            m[16].valor=745676;
            m[17].valor=813831;
            m[18].valor=886385;
            m[19].valor=963947;
            m[20].valor=1047259;
            m[21].valor=1137243;
            m[22].valor=1235061;
            m[23].valor=1342209;
            m[24].valor=1460655;
            m[25].valor=1593067;
            m[26].valor=1743183;
            m[27].valor=1916479;
            m[28].valor=2121445;
            m[29].valor=2372303;
            m[30].valor=2695715;
            m[31].valor=3151539;
            m[32].valor=3930775;
        break;
        case 780:
            m[0].valor=1;
            m[1].valor=35044;
            m[2].valor=71199;
            m[3].valor=108540;
            m[4].valor=147147;
            m[5].valor=187109;
            m[6].valor=228525;
            m[7].valor=271503;
            m[8].valor=316168;
            m[9].valor=362656;
            m[10].valor=411123;
            m[11].valor=461745;
            m[12].valor=514722;
            m[13].valor=570284;
            m[14].valor=628697;
            m[15].valor=690268;
            m[16].valor=755360;
            m[17].valor=824400;
            m[18].valor=897896;
            m[19].valor=976465;
            m[20].valor=1060859;
            m[21].valor=1152012;
            m[22].valor=1251101;
            m[23].valor=1359640;
            m[24].valor=1479624;
            m[25].valor=1613756;
            m[26].valor=1765821;
            m[27].valor=1941368;
            m[28].valor=2148996;
            m[29].valor=2403112;
            m[30].valor=2730724;
            m[31].valor=3192468;
            m[32].valor=3981824;
        break;
        case 790:
            m[0].valor=1;
            m[1].valor=35493;
            m[2].valor=72112;
            m[3].valor=109932;
            m[4].valor=149034;
            m[5].valor=189508;
            m[6].valor=231455;
            m[7].valor=274984;
            m[8].valor=320221;
            m[9].valor=367306;
            m[10].valor=416394;
            m[11].valor=467664;
            m[12].valor=521321;
            m[13].valor=577595;
            m[14].valor=636757;
```

```
            m[15].valor=699118;
            m[16].valor=765045;
            m[17].valor=834969;
            m[18].valor=909408;
            m[19].valor=988984;
            m[20].valor=1074460;
            m[21].valor=1166781;
            m[22].valor=1267140;
            m[23].valor=1377071;
            m[24].valor=1498594;
            m[25].valor=1634445;
            m[26].valor=1788460;
            m[27].valor=1966257;
            m[28].valor=2176547;
            m[29].valor=2433921;
            m[30].valor=2765733;
            m[31].valor=3233397;
            m[32].valor=4032873;
        break;
        case 800:
            m[0].valor=1;
            m[1].valor=35942;
            m[2].valor=73025;
            m[3].valor=111323;
            m[4].valor=150920;
            m[5].valor=191907;
            m[6].valor=234384;
            m[7].valor=278465;
            m[8].valor=324275;
            m[9].valor=371955;
            m[10].valor=421665;
            m[11].valor=473584;
            m[12].valor=527920;
            m[13].valor=584907;
            m[14].valor=644817;
            m[15].valor=707968;
            m[16].valor=774729;
            m[17].valor=845538;
            m[18].valor=920919;
            m[19].valor=1001503;
            m[20].valor=1088061;
            m[21].valor=1181551;
            m[22].valor=1283180;
            m[23].valor=1394502;
            m[24].valor=1517564;
            m[25].valor=1655134;
            m[26].valor=1811099;
            m[27].valor=1991147;
            m[28].valor=2204098;
            m[29].valor=2464730;
            m[30].valor=2800743;
            m[31].valor=3274326;
            m[32].valor=4083922;
        break;
        case 810:
            m[0].valor=1;
            m[1].valor=36392;
            m[2].valor=73938;
            m[3].valor=112715;
            m[4].valor=152807;
            m[5].valor=194306;
            m[6].valor=237314;
            m[7].valor=281946;
            m[8].valor=328328;
            m[9].valor=376604;
            m[10].valor=426935;
            m[11].valor=479504;
            m[12].valor=534519;
            m[13].valor=592218;
            m[14].valor=652877;
            m[15].valor=716817;
            m[16].valor=784413;
            m[17].valor=856107;
            m[18].valor=932431;
            m[19].valor=1014022;
            m[20].valor=1101662;
            m[21].valor=1196320;
            m[22].valor=1299220;
            m[23].valor=1411934;
            m[24].valor=1536533;
            m[25].valor=1675823;
            m[26].valor=1833737;
            m[27].valor=2016036;
            m[28].valor=2231650;
            m[29].valor=2495539;
            m[30].valor=2835752;
            m[31].valor=3315255;
            m[32].valor=4134971;
        break;
        case 820:
            m[0].valor=1;
            m[1].valor=36841;
            m[2].valor=74850;
            m[3].valor=114106;
            m[4].valor=154693;
            m[5].valor=196705;
            m[6].valor=240244;
            m[7].valor=285427;
            m[8].valor=332382;
            m[9].valor=381254;
            m[10].valor=432206;
            m[11].valor=485424;
            m[12].valor=541118;
            m[13].valor=599529;
            m[14].valor=660938;
            m[15].valor=725667;
            m[16].valor=794097;
            m[17].valor=866677;
            m[18].valor=943942;
            m[19].valor=1026540;
            m[20].valor=1115262;
            m[21].valor=1211090;
            m[22].valor=1315260;
            m[23].valor=1429365;
            m[24].valor=1555503;
            m[25].valor=1696512;
            m[26].valor=1856376;
            m[27].valor=2040925;
            m[28].valor=2259201;
            m[29].valor=2526348;
            m[30].valor=2870761;
            m[31].valor=3356184;
            m[32].valor=4186020;
        break;
        case 830:
            m[0].valor=1;
            m[1].valor=37290;
            m[2].valor=75763;
            m[3].valor=115498;
            m[4].valor=156580;
            m[5].valor=199103;
            m[6].valor=243174;
            m[7].valor=288907;
            m[8].valor=336435;
            m[9].valor=385903;
            m[10].valor=437477;
            m[11].valor=491344;
            m[12].valor=547717;
            m[13].valor=606840;
            m[14].valor=668998;
            m[15].valor=734516;
            m[16].valor=803781;
            m[17].valor=877246;
            m[18].valor=955454;
            m[19].valor=1039059;
            m[20].valor=1128863;
```

```
            m[21].valor=1225859;
            m[22].valor=1331299;
            m[23].valor=1446796;
            m[24].valor=1574472;
            m[25].valor=1717202;
            m[26].valor=1879015;
            m[27].valor=2065815;
            m[28].valor=2286752;
            m[29].valor=2557157;
            m[30].valor=2905770;
            m[31].valor=3397113;
            m[32].valor=4237069;
        break;
        case 840:
            m[0].valor=1;
            m[1].valor=37739;
            m[2].valor=76676;
            m[3].valor=116889;
            m[4].valor=158466;
            m[5].valor=201502;
            m[6].valor=246104;
            m[7].valor=292388;
            m[8].valor=340489;
            m[9].valor=390553;
            m[10].valor=442748;
            m[11].valor=497263;
            m[12].valor=554316;
            m[13].valor=614152;
            m[14].valor=677058;
            m[15].valor=743366;
            m[16].valor=813465;
            m[17].valor=887815;
            m[18].valor=966965;
            m[19].valor=1051578;
            m[20].valor=1142464;
            m[21].valor=1240628;
            m[22].valor=1347339;
            m[23].valor=1464228;
            m[24].valor=1593442;
            m[25].valor=1737891;
            m[26].valor=1901654;
            m[27].valor=2090704;
            m[28].valor=2314303;
            m[29].valor=2587966;
            m[30].valor=2940780;
            m[31].valor=3438042;
            m[32].valor=4288118;
        break;
        case 850:
            m[0].valor=1;
            m[1].valor=38189;
            m[2].valor=77589;
            m[3].valor=118281;
            m[4].valor=160353;
            m[5].valor=203901;
            m[6].valor=249033;
            m[7].valor=295869;
            m[8].valor=344542;
            m[9].valor=395202;
            m[10].valor=448019;
            m[11].valor=503183;
            m[12].valor=560915;
            m[13].valor=621463;
            m[14].valor=685118;
            m[15].valor=752216;
            m[16].valor=823149;
            m[17].valor=898384;
            m[18].valor=978477;
            m[19].valor=1064097;
            m[20].valor=1156065;
            m[21].valor=1255398;
            m[22].valor=1363379;
            m[23].valor=1481659;
            m[24].valor=1612411;
            m[25].valor=1758580;
            m[26].valor=1924292;
            m[27].valor=2115593;
            m[28].valor=2341854;
            m[29].valor=2618776;
            m[30].valor=2975789;
            m[31].valor=3478971;
            m[32].valor=4339167;
        break;
        case 860:
            m[0].valor=1;
            m[1].valor=38638;
            m[2].valor=78502;
            m[3].valor=119672;
            m[4].valor=162239;
            m[5].valor=206300;
            m[6].valor=251963;
            m[7].valor=299350;
            m[8].valor=348595;
            m[9].valor=399852;
            m[10].valor=453289;
            m[11].valor=509103;
            m[12].valor=567514;
            m[13].valor=628774;
            m[14].valor=693178;
            m[15].valor=761065;
            m[16].valor=832833;
            m[17].valor=908953;
            m[18].valor=989988;
            m[19].valor=1076616;
            m[20].valor=1169665;
            m[21].valor=1270167;
            m[22].valor=1379419;
            m[23].valor=1499090;
            m[24].valor=1631381;
            m[25].valor=1779269;
            m[26].valor=1946931;
            m[27].valor=2140483;
            m[28].valor=2369406;
            m[29].valor=2649585;
            m[30].valor=3010798;
            m[31].valor=3519900;
            m[32].valor=4390216;
        break;
        case 870:
            m[0].valor=1;
            m[1].valor=39087;
            m[2].valor=79414;
            m[3].valor=121064;
            m[4].valor=164126;
            m[5].valor=208699;
            m[6].valor=254893;
            m[7].valor=302831;
            m[8].valor=352649;
            m[9].valor=404501;
            m[10].valor=458560;
            m[11].valor=515023;
            m[12].valor=574113;
            m[13].valor=636086;
            m[14].valor=701239;
            m[15].valor=769915;
            m[16].valor=842517;
            m[17].valor=919523;
            m[18].valor=1001500;
            m[19].valor=1089134;
            m[20].valor=1183266;
            m[21].valor=1284936;
            m[22].valor=1395458;
            m[23].valor=1516521;
            m[24].valor=1650350;
            m[25].valor=1799958;
            m[26].valor=1969570;
```

```
            m[27].valor=2165372;
            m[28].valor=2396957;
            m[29].valor=2680394;
            m[30].valor=3045808;
            m[31].valor=3560829;
            m[32].valor=4441265;
        break;
        case 880:
            m[0].valor=1;
            m[1].valor=39536;
            m[2].valor=80327;
            m[3].valor=122456;
            m[4].valor=166012;
            m[5].valor=211098;
            m[6].valor=257823;
            m[7].valor=306311;
            m[8].valor=356702;
            m[9].valor=409150;
            m[10].valor=463831;
            m[11].valor=520943;
            m[12].valor=580711;
            m[13].valor=643397;
            m[14].valor=709299;
            m[15].valor=778764;
            m[16].valor=852201;
            m[17].valor=930092;
            m[18].valor=1013011;
            m[19].valor=1101653;
            m[20].valor=1196867;
            m[21].valor=1299706;
            m[22].valor=1411498;
            m[23].valor=1533953;
            m[24].valor=1669320;
            m[25].valor=1820647;
            m[26].valor=1992209;
            m[27].valor=2190261;
            m[28].valor=2424508;
            m[29].valor=2711203;
            m[30].valor=3080817;
            m[31].valor=3601758;
            m[32].valor=4492314;
        break;
        case 890:
            m[0].valor=1;
            m[1].valor=39986;
            m[2].valor=81240;
            m[3].valor=123847;
            m[4].valor=167899;
            m[5].valor=213496;
            m[6].valor=260753;
            m[7].valor=309792;
            m[8].valor=360756;
            m[9].valor=413800;
            m[10].valor=469102;
            m[11].valor=526862;
            m[12].valor=587310;
            m[13].valor=650708;
            m[14].valor=717359;
            m[15].valor=787614;
            m[16].valor=861886;
            m[17].valor=940661;
            m[18].valor=1024522;
            m[19].valor=1114172;
            m[20].valor=1210468;
            m[21].valor=1314475;
            m[22].valor=1427538;
            m[23].valor=1551384;
            m[24].valor=1688289;
            m[25].valor=1841337;
            m[26].valor=2014847;
            m[27].valor=2215151;
            m[28].valor=2452059;
            m[29].valor=2742012;
            m[30].valor=3115826;
            m[31].valor=3642687;
            m[32].valor=4543363;
        break;
        case 900:
            m[0].valor=1;
            m[1].valor=40435;
            m[2].valor=82153;
            m[3].valor=125239;
            m[4].valor=169785;
            m[5].valor=215895;
            m[6].valor=263682;
            m[7].valor=313273;
            m[8].valor=364809;
            m[9].valor=418449;
            m[10].valor=474373;
            m[11].valor=532782;
            m[12].valor=593909;
            m[13].valor=658020;
            m[14].valor=725419;
            m[15].valor=796463;
            m[16].valor=871570;
            m[17].valor=951230;
            m[18].valor=1036034;
            m[19].valor=1126691;
            m[20].valor=1224068;
            m[21].valor=1329245;
            m[22].valor=1443578;
            m[23].valor=1568815;
            m[24].valor=1707259;
            m[25].valor=1862026;
            m[26].valor=2037486;
            m[27].valor=2240040;
            m[28].valor=2479611;
            m[29].valor=2772821;
            m[30].valor=3150835;
            m[31].valor=3683617;
            m[32].valor=4594412;
        break;
        case 910:
            m[0].valor=1;
            m[1].valor=40884;
            m[2].valor=83066;
            m[3].valor=126630;
            m[4].valor=171672;
            m[5].valor=218294;
            m[6].valor=266612;
            m[7].valor=316754;
            m[8].valor=368863;
            m[9].valor=423099;
            m[10].valor=479643;
            m[11].valor=538702;
            m[12].valor=600508;
            m[13].valor=665331;
            m[14].valor=733479;
            m[15].valor=805313;
            m[16].valor=881254;
            m[17].valor=961800;
            m[18].valor=1047545;
            m[19].valor=1139209;
            m[20].valor=1237669;
            m[21].valor=1344014;
            m[22].valor=1459617;
            m[23].valor=1586246;
            m[24].valor=1726228;
            m[25].valor=1882715;
            m[26].valor=2060125;
            m[27].valor=2264929;
            m[28].valor=2507162;
            m[29].valor=2803630;
            m[30].valor=3185845;
            m[31].valor=3724546;
            m[32].valor=4645461;
```

```
break;
case 920:
        m[0].valor=1;
        m[1].valor=41333;
        m[2].valor=83978;
        m[3].valor=128022;
        m[4].valor=173558;
        m[5].valor=220693;
        m[6].valor=269542;
        m[7].valor=320235;
        m[8].valor=372916;
        m[9].valor=427748;
        m[10].valor=484914;
        m[11].valor=544622;
        m[12].valor=607107;
        m[13].valor=672642;
        m[14].valor=741540;
        m[15].valor=814163;
        m[16].valor=890938;
        m[17].valor=972369;
        m[18].valor=1059057;
        m[19].valor=1151728;
        m[20].valor=1251270;
        m[21].valor=1358783;
        m[22].valor=1475657;
        m[23].valor=1603678;
        m[24].valor=1745198;
        m[25].valor=1903404;
        m[26].valor=2082763;
        m[27].valor=2289819;
        m[28].valor=2534713;
        m[29].valor=2834439;
        m[30].valor=3220854;
        m[31].valor=3765475;
        m[32].valor=4696510;
break;
case 930:
        m[0].valor=1;
        m[1].valor=41783;
        m[2].valor=84891;
        m[3].valor=129413;
        m[4].valor=175445;
        m[5].valor=223092;
        m[6].valor=272472;
        m[7].valor=323715;
        m[8].valor=376969;
        m[9].valor=432397;
        m[10].valor=490185;
        m[11].valor=550542;
        m[12].valor=613706;
        m[13].valor=679954;
        m[14].valor=749600;
        m[15].valor=823012;
        m[16].valor=900622;
        m[17].valor=982938;
        m[18].valor=1070568;
        m[19].valor=1164247;
        m[20].valor=1264871;
        m[21].valor=1373553;
        m[22].valor=1491697;
        m[23].valor=1621109;
        m[24].valor=1764167;
        m[25].valor=1924093;
        m[26].valor=2105402;
        m[27].valor=2314708;
        m[28].valor=2562264;
        m[29].valor=2865248;
        m[30].valor=3255863;
        m[31].valor=3806404;
        m[32].valor=4747559;
break;
case 940:
        m[0].valor=1;
        m[1].valor=42232;
        m[2].valor=85804;
        m[3].valor=130805;
        m[4].valor=177331;
        m[5].valor=225491;
        m[6].valor=275401;
        m[7].valor=327196;
        m[8].valor=381023;
        m[9].valor=437047;
        m[10].valor=495456;
        m[11].valor=556461;
        m[12].valor=620305;
        m[13].valor=687265;
        m[14].valor=757660;
        m[15].valor=831862;
        m[16].valor=910306;
        m[17].valor=993507;
        m[18].valor=1082080;
        m[19].valor=1176766;
        m[20].valor=1278471;
        m[21].valor=1388322;
        m[22].valor=1507737;
        m[23].valor=1638540;
        m[24].valor=1783137;
        m[25].valor=1944782;
        m[26].valor=2128041;
        m[27].valor=2339597;
        m[28].valor=2589815;
        m[29].valor=2896058;
        m[30].valor=3290872;
        m[31].valor=3847333;
        m[32].valor=4798608;
break;
case 950:
        m[0].valor=1;
        m[1].valor=42681;
        m[2].valor=86717;
        m[3].valor=132196;
        m[4].valor=179218;
        m[5].valor=227889;
        m[6].valor=278331;
        m[7].valor=330677;
        m[8].valor=385076;
        m[9].valor=441696;
        m[10].valor=500727;
        m[11].valor=562381;
        m[12].valor=626904;
        m[13].valor=694576;
        m[14].valor=765720;
        m[15].valor=840711;
        m[16].valor=919990;
        m[17].valor=1004076;
        m[18].valor=1093591;
        m[19].valor=1189284;
        m[20].valor=1292072;
        m[21].valor=1403091;
        m[22].valor=1523776;
        m[23].valor=1655971;
        m[24].valor=1802106;
        m[25].valor=1965472;
        m[26].valor=2150680;
        m[27].valor=2364487;
        m[28].valor=2617367;
        m[29].valor=2926867;
        m[30].valor=3325882;
        m[31].valor=3888262;
        m[32].valor=4849657;
break;
case 960:
        m[0].valor=1;
        m[1].valor=43131;
        m[2].valor=87630;
        m[3].valor=133588;
```

```
            m[4].valor=181104;
            m[5].valor=230288;
            m[6].valor=281261;
            m[7].valor=334158;
            m[8].valor=389130;
            m[9].valor=446346;
            m[10].valor=505997;
            m[11].valor=568301;
            m[12].valor=633503;
            m[13].valor=701888;
            m[14].valor=773780;
            m[15].valor=849561;
            m[16].valor=929674;
            m[17].valor=1014646;
            m[18].valor=1105103;
            m[19].valor=1201803;
            m[20].valor=1305673;
            m[21].valor=1417861;
            m[22].valor=1539816;
            m[23].valor=1673403;
            m[24].valor=1821076;
            m[25].valor=1986161;
            m[26].valor=2173318;
            m[27].valor=2389376;
            m[28].valor=2644918;
            m[29].valor=2957676;
            m[30].valor=3360891;
            m[31].valor=3929191;
            m[32].valor=4900706;
        break;
        case 970:
            m[0].valor=1;
            m[1].valor=43580;
            m[2].valor=88542;
            m[3].valor=134979;
            m[4].valor=182991;
            m[5].valor=232687;
            m[6].valor=284191;
            m[7].valor=337639;
            m[8].valor=393183;
            m[9].valor=450995;
            m[10].valor=511268;
            m[11].valor=574221;
            m[12].valor=640102;
            m[13].valor=709199;
            m[14].valor=781841;
            m[15].valor=858411;
            m[16].valor=939358;
            m[17].valor=1025215;
            m[18].valor=1116614;
            m[19].valor=1214322;
            m[20].valor=1319274;
            m[21].valor=1432630;
            m[22].valor=1555856;
            m[23].valor=1690834;
            m[24].valor=1840046;
            m[25].valor=2006850;
            m[26].valor=2195957;
            m[27].valor=2414265;
            m[28].valor=2672469;
            m[29].valor=2988485;
            m[30].valor=3395900;
            m[31].valor=3970120;
            m[32].valor=4951755;
        break;
        case 980:
            m[0].valor=1;
            m[1].valor=44029;
            m[2].valor=89455;
            m[3].valor=136371;
            m[4].valor=184877;
            m[5].valor=235086;
            m[6].valor=287121;
            m[7].valor=341119;
            m[8].valor=397236;
            m[9].valor=455645;
            m[10].valor=516539;
            m[11].valor=580140;
            m[12].valor=646701;
            m[13].valor=716510;
            m[14].valor=789901;
            m[15].valor=867260;
            m[16].valor=949042;
            m[17].valor=1035784;
            m[18].valor=1128126;
            m[19].valor=1226841;
            m[20].valor=1332874;
            m[21].valor=1447400;
            m[22].valor=1571895;
            m[23].valor=1708265;
            m[24].valor=1859015;
            m[25].valor=2027539;
            m[26].valor=2218596;
            m[27].valor=2439155;
            m[28].valor=2700020;
            m[29].valor=3019294;
            m[30].valor=3430910;
            m[31].valor=4011049;
            m[32].valor=5002804;
        break;
        case 990:
            m[0].valor=1;
            m[1].valor=44478;
            m[2].valor=90368;
            m[3].valor=137762;
            m[4].valor=186764;
            m[5].valor=237485;
            m[6].valor=290050;
            m[7].valor=344600;
            m[8].valor=401290;
            m[9].valor=460294;
            m[10].valor=521810;
            m[11].valor=586060;
            m[12].valor=653300;
            m[13].valor=723822;
            m[14].valor=797961;
            m[15].valor=876110;
            m[16].valor=958726;
            m[17].valor=1046353;
            m[18].valor=1139637;
            m[19].valor=1239360;
            m[20].valor=1346475;
            m[21].valor=1462169;
            m[22].valor=1587935;
            m[23].valor=1725697;
            m[24].valor=1877985;
            m[25].valor=2048228;
            m[26].valor=2241234;
            m[27].valor=2464044;
            m[28].valor=2727571;
            m[29].valor=3050103;
            m[30].valor=3465919;
            m[31].valor=4051978;
            m[32].valor=5053853;
        break;
        case 1000:
            m[0].valor=1;
            m[1].valor=44928;
            m[2].valor=91281;
            m[3].valor=139154;
            m[4].valor=188650;
            m[5].valor=239883;
            m[6].valor=292980;
            m[7].valor=348081;
            m[8].valor=405343;
            m[9].valor=464943;
```

```
                m[10].valor=527080;
                m[11].valor=591980;
                m[12].valor=659899;
                m[13].valor=731133;
                m[14].valor=806021;
                m[15].valor=884959;
                m[16].valor=968411;
                m[17].valor=1056923;
                m[18].valor=1151149;
                m[19].valor=1251878;
                m[20].valor=1360076;
                m[21].valor=1476938;
                m[22].valor=1603975;
                m[23].valor=1743128;
                m[24].valor=1896954;
                m[25].valor=2068917;
                m[26].valor=2263873;
                m[27].valor=2488933;
                m[28].valor=2755123;
                m[29].valor=3080912;
                m[30].valor=3500928;
                m[31].valor=4092907;
                m[32].valor=5104902;
                        break;
                }
        }
```

ANEXO B: CÓDIGO PARA ANALIZAR DATOS

DataAnalyser

```
#include <iostream>
#include <fstream>
#include <string>
//#include <regex>
#include <stdlib.h>
/*Compile with:
 * g++ data_Reader.cpp -o data_Reader -std=c++11
 *
 */
using namespace std;

enum { TIME, SENDER_PORT, RECEIVER_PORT, PACK_BYTES, SND_NXT, SND_UNA, CWND, SSTHRES, SND_WIN };

int main(int argc, char *argv[]){

        if (argc<3){
                //cout << "\nIntente con mas argumentos: \"data_Reader Dir_input col1 col2 col3 ... coln \"\n\n";
        }else{
                string comando = "ls -1 ";
                string dir = argv[1];
                string end = "*.info > lista.tmp";
                string full;
                full = comando + dir + end;
                system(full.c_str()); //Se crea lista de directorios donde buscar
                ifstream tmp("lista.tmp", std::ifstream::in);
                while (tmp.good()){
                        string file;
                        getline (tmp,file);
                        string line;
                        int counter=0;
                        string instancia(argv[3]);
                        ifstream data(file.c_str(), std::ifstream::in);
                        string out_file = file+".ana";
                        ofstream outfile(out_file.c_str(), std::ofstream::out);
                        //("test.txt", std::ofstream::out)
                        // DEFINICION DE ARCHIVOS DE SALIDA

                                string p_cwnd = file+".p.cwnd";
                                ofstream pcwnd(p_cwnd.c_str(), std::ofstream::out);
                                double prom_cwnd_full=0;
                                long unsigned int cnt_cwnd_full=0;

                                string p_hscwnd = file+".phs.cwnd";
                                ofstream phscwnd(p_hscwnd.c_str(), std::ofstream::out);
                                double prom_cwnd_hs=0.0;
                                long unsigned int cnt_cwnd_hs=0;
```

```
		double cwnd_hs=0.5;

		string p_1cwnd = file+".p1s.cwnd";
		ofstream p1cwnd(p_1cwnd.c_str(), std::ofstream::out);
		double prom_cwnd_1s=0;
		long unsigned int cnt_cwnd_1s=0;
		double cwnd_1s=1;

		string p_10cwnd = file+".p10s.cwnd";
		ofstream p10cwnd(p_10cwnd.c_str(), std::ofstream::out);
		double prom_cwnd_10s=0;
		long unsigned int cnt_cwnd_10s=0;
		double cwnd_10s=10;

		string p_hsthr = file+".phs.thr";
		ofstream phsthr(p_hsthr.c_str(), std::ofstream::out);
		double prom_thr_hs=0.0;
		long unsigned int ref_thr_hs=0;
		double thr_hs=1/2;
		long unsigned int last_UNA=0;

		string p_thr = file+".p.thr";
		ofstream pthr(p_thr.c_str(), std::ofstream::out);
		double prom_thr_full=0;
		long unsigned int ref_thr_full=0;

		string p1s_thr = file+".p1s.thr";
		ofstream p1sthr(p1s_thr.c_str(), std::ofstream::out);
		double prom_thr_1s=0;
		double ref_thr_1s=0;
		double thr_1s=1;

		string p10s_thr = file+".p10s.thr";
		ofstream p10sthr(p10s_thr.c_str(), std::ofstream::out);
		double prom_thr_10s=0;
		double ref_thr_10s=0;
		double thr_10s=10;

		string p_sst = file+".p.sst";
		ofstream psst(p_sst.c_str(), std::ofstream::out);
		double prom_sst_full=0;
		long unsigned int cnt_sst_full=0;

		string p_hssst = file+".phs.sst";
		ofstream phssst(p_hssst.c_str(), std::ofstream::out);
		double prom_sst_hs=0.0;
		long unsigned int cnt_sst_hs=0;
		double sst_hs=0.5;

		string p1s_sst = file+".p1s.sst";
		ofstream p1ssst(p1s_sst.c_str(), std::ofstream::out);
		double prom_sst_1s=0;
		long unsigned int cnt_sst_1s=0;
		double sst_1s=1;

		string p10s_sst = file+".p10s.sst";
		ofstream p10ssst(p10s_sst.c_str(), std::ofstream::out);
		double prom_sst_10s=0;
		long unsigned int cnt_sst_10s=0;
		double sst_10s=10;

//FIN DEFINICION ARCHIVOS DE SALIDA
int index_space;
string datos[20];
while (data.good()){
		getline (data,line);
		////cout << line;

		int columna=0;
		do{
			index_space=line.find(" ");
			//cout << "index_space: " << index_space << "\n";
			if(index_space<0){
```

```
                                        datos[columna]=line;
                                        //cout << datos[columna] << "\n\n\n\n";
                                }else{
                                        datos[columna++]=line.substr(0,index_space);
                                        line=line.substr(index_space+1,line.length());
                                        //cout << line << "\n";
                                }
                        }while(index_space>0);

                        // EN ESTE PUNTO SE TIENEN TODOS LOS DATOS DE LA LINEA, FALTA ESCRIBIR
FUNCION QUE HAGA ANALISIS
                        //cout << "argc: " << argc <<"\n";
                        //for (int i=0;i<columna+1;i++){
                                //cout << datos[i] << "\t";
                        //}
                        //for(int i=2;i<argc-1;i++){
                                ////cout << "atoi(argv[" << i << "]) : " << atoi(argv[i]) << " ";
                                ////cout << "datos[atoi(argv" << i << " )] = " <<datos[atoi(argv[i])]<<"\n";
                                ////cout << datos[TIME] << "\n";
                                ////cout << datos[SND_NXT] << " = "<< strtoul(datos[SND_NXT].c_str(),NULL,0) <<
"\t";
                                ////cout << datos[SND_UNA] << " = "<< strtoul(datos[SND_UNA].c_str(),NULL,0) <<
"\n";
                                //outfile << datos[atoi(argv[i])-1] << "\t";
                        //}
                        ////cout << "siguiente linea\n\n";
                        //outfile << datos[atoi(argv[argc-1])-1] << "\n";

        prom_cwnd_full=((prom_cwnd_full*cnt_cwnd_full)+strtoul(datos[CWND].c_str(),NULL,0))/(cnt_cwnd_full+1);
                        cnt_cwnd_full++;


        prom_cwnd_hs=((prom_cwnd_hs*(cnt_cwnd_hs*1.0))+strtoul(datos[CWND].c_str(),NULL,0))/(cnt_cwnd_hs+1);
                        cnt_cwnd_hs++;
                        if((strtoul(datos[TIME].c_str(),NULL,0)/(cwnd_hs*1.0))>=1){
                                phscwnd << cwnd_hs << "\t" << (int)prom_cwnd_hs << "\n";
                                cwnd_hs+=0.5;
                                prom_cwnd_hs=0;
                                cnt_cwnd_hs=0;
                        }

        prom_cwnd_1s=((prom_cwnd_1s*cnt_cwnd_1s)+strtoul(datos[CWND].c_str(),NULL,0))/(cnt_cwnd_1s+1);
                        cnt_cwnd_1s++;
                        if((strtoul(datos[TIME].c_str(),NULL,0)/cwnd_1s)>=1){
                                p1cwnd << cwnd_1s << "\t" << (int)prom_cwnd_1s << "\n";
                                cwnd_1s+=1;
                                prom_cwnd_1s=0;
                                cnt_cwnd_1s=0;
                        }

        prom_cwnd_10s=((prom_cwnd_10s*cnt_cwnd_10s)+strtoul(datos[CWND].c_str(),NULL,0))/(cnt_cwnd_10s+1);
                        cnt_cwnd_10s++;
                        if((strtoul(datos[TIME].c_str(),NULL,0)/cwnd_10s)>=1){
                                p10cwnd << cwnd_10s << "\t" << (int)prom_cwnd_10s << "\n";
                                cwnd_10s+=10;
                                prom_cwnd_10s=0;
                                cnt_cwnd_10s=0;
                        }

                        if(ref_thr_full==0){
                                ref_thr_full=(strtoul(datos[SND_UNA].c_str(),NULL,0)-1);
                                ref_thr_hs=(strtoul(datos[SND_UNA].c_str(),NULL,0)-1);
                                ref_thr_1s=(strtoul(datos[SND_UNA].c_str(),NULL,0)-1);
                                ref_thr_10s=(strtoul(datos[SND_UNA].c_str(),NULL,0)-1);
                        }

                        if(strtoul(datos[TIME].c_str(),NULL,0)>=thr_hs){
                                if(strtoul(datos[SND_UNA].c_str(),NULL,0)>=ref_thr_hs){
                                        prom_thr_hs=16*(strtoul(datos[SND_UNA].c_str(),NULL,0)-
ref_thr_hs)/1000000.0; //Mbits
                                }else{
                                        prom_thr_hs=16*(0xffffffff-
ref_thr_hs+strtoul(datos[SND_UNA].c_str(),NULL,0))/1000000.0; //Mbits
```

```
                                        }
                                        phsthr << thr_hs << "\t" << prom_thr_hs << "\n";
                                        ref_thr_hs=(strtoul(datos[SND_UNA].c_str(),NULL,0))-1;
                                        thr_hs+=0.5;
                                }

                                if(strtoul(datos[SND_UNA].c_str(),NULL,0)>0){
                                        last_UNA=strtoul(datos[SND_UNA].c_str(),NULL,0)>0;
                                }

                                if((strtoul(datos[TIME].c_str(),NULL,0)/thr_1s)>=1){
                                        if(strtoul(datos[SND_UNA].c_str(),NULL,0)>=ref_thr_1s){
                                                prom_thr_1s=8*(strtoul(datos[SND_UNA].c_str(),NULL,0)-
ref_thr_1s)/1000000.0; //Mbits
                                        }else{
                                                prom_thr_1s=8*(0xffffffff-
ref_thr_1s+strtoul(datos[SND_UNA].c_str(),NULL,0))/1000000.0; //Mbits
                                        }
                                        p1sthr << thr_1s << "\t" << prom_thr_1s << "\n";
                                        ref_thr_1s=strtoul(datos[SND_UNA].c_str(),NULL,0)-1;
                                        thr_1s+=1;
                                }

                                if((strtoul(datos[TIME].c_str(),NULL,0)/thr_10s)>=1){
                                        if(strtoul(datos[SND_UNA].c_str(),NULL,0)>=ref_thr_10s){
                                                prom_thr_10s=8*(strtoul(datos[SND_UNA].c_str(),NULL,0)-
ref_thr_10s)/10000000.0; //Mbits
                                        }else{
                                                prom_thr_10s=8*(0xffffffff-
ref_thr_10s+strtoul(datos[SND_UNA].c_str(),NULL,0))/10000000.0; //Mbits
                                        }
                                        p10sthr << thr_10s << "\t" << prom_thr_10s << "\n";
                                        ref_thr_10s=strtoul(datos[SND_UNA].c_str(),NULL,0)-1;
                                        thr_10s+=0.5;
                                }


        prom_sst_full=((prom_sst_full*cnt_sst_full)+strtoul(datos[SSTHRES].c_str(),NULL,0))/(cnt_sst_full+1);
                                cnt_sst_full++;



                                prom_sst_hs=((prom_sst_hs*cnt_sst_hs)+strtoul(datos[SSTHRES].c_str(),NULL,0))/(cnt_sst_hs+1);
                                cnt_sst_hs++;
                                if((strtoul(datos[TIME].c_str(),NULL,0)/sst_hs)>=1){
                                        phssst << sst_hs << "\t" << prom_sst_hs << "\n";
                                        sst_hs+=0.5;
                                        prom_sst_hs=0;
                                        cnt_sst_hs=0;
                                }

                                prom_sst_1s=((prom_sst_1s*cnt_sst_1s)+strtoul(datos[SSTHRES].c_str(),NULL,0))/(cnt_sst_1s+1);
                                cnt_sst_1s++;
                                if((strtoul(datos[TIME].c_str(),NULL,0)/sst_1s)>=1){
                                        p1ssst << sst_1s << "\t" << prom_sst_1s << "\n";
                                        sst_1s+=1;
                                        prom_sst_1s=0;
                                        cnt_sst_1s=0;
                                }


        prom_sst_10s=((prom_sst_10s*cnt_sst_10s)+strtoul(datos[SSTHRES].c_str(),NULL,0))/(cnt_sst_10s+1);
                                cnt_sst_10s++;
                                if((strtoul(datos[TIME].c_str(),NULL,0)/sst_10s)>=1){
                                        p10ssst << sst_10s << "\t" << prom_sst_10s << "\n";
                                        sst_10s+=10;
                                        prom_sst_10s=0;
                                        cnt_sst_10s=0;
                                }
                        }


                        psst << strtoul(datos[TIME].c_str(),NULL,0) << "\t" << prom_sst_full << "\n";
```

```
                                        if(strtoul(datos[SND_UNA].c_str(),NULL,0)>=ref_thr_full){
                                                prom_thr_full=8*(last_UNA-ref_thr_full)/(1000000.0*strtoul(datos[TIME].c_str(),NULL,0)); //Mbits
                                        }else{
                                                prom_thr_full=8*(0xffffffff-
ref_thr_full+last_UNA)/(1000000.0*strtoul(datos[TIME].c_str(),NULL,0)); //Mbits
                                        }

                                        pthr << strtoul(datos[TIME].c_str(),NULL,0) << "\t" << prom_thr_full << "\n";
                                        pcwnd << strtoul(datos[TIME].c_str(),NULL,0) << "\t" << prom_cwnd_full << "\n";
                                        //prom_cwnd_full
                                        //prom_sst_full

                                        data.close();
                                        outfile.close();
                                        pcwnd.close();
                                        phscwnd.close();
                                        p1cwnd.close();
                                        p10cwnd.close();
                                        pthr.close();
                                        phsthr.close();
                                        p1sthr.close();
                                        p10sthr.close();
                                        psst.close();
                                        phssst.close();
                                        p1ssst.close();
                                        p10ssst.close();
                                }
                        }
                        return 0;
}
```

PromediaResultados

```
#include <iostream>
#include <fstream>
#include <string>
//#include <regex>
#include <stdlib.h>
/*Compile with:
 * g++ data_Reader.cpp -o data_Reader -std=c++11
 *
 */
using namespace std;

enum { TIME, SENDER_PORT, RECEIVER_PORT, PACK_BYTES, SND_NXT, SND_UNA, CWND, SSTHRES, SND_WIN };
const int numero_de_protocolos=4;
const int numero_de_pruebas=2;
const int numero_de_rtts=19;
const int numero_de_extensiones=3;
long unsigned int n_datos_no_validos=0;

int main(int argc, char *argv[]){
        if (argc<3){
                cout << "\nIntente con mas argumentos: \"PromediaResultados Dir_input Dir_output \"\n\n";
        }else{
                string uBICacion=argv[1];
                string uBICacion_salida=argv[2];
                string protocolo[5];
                protocolo[0]="BIC";
                protocolo[1]="CUBIC";
                protocolo[2]="ESTP";
                protocolo[3]="RENO";
                //protocolo[3]="hTCP";
                //protocolo[4]="RENO";

                int delay[numero_de_rtts];
                for (int i=0;i<10;i++){
                        delay[i]=i+1;
                }
```

```
		int paso=10;
		for (int i=10;i<19;i++){
			delay[i]=10+paso*(i-9);
		}

		string tipo[3];
		tipo[0]="cwnd";
		tipo[1]="sst";
		tipo[2]="thr";

		int test[numero_de_pruebas];
		test[0]=1;
		test[1]=2;
		//test[2]=3;
		//test[3]=4;
		//test[4]=5;
		string d_sep="_d_";
		string t_sep="_t_";
		string first_ext=".info.p1s.";
		string point = ".";
		string tail_log=".log";

		int proto;
		int prueba;
		int ext;
		int rtt;
		string linea[numero_de_pruebas];
		string datos[numero_de_pruebas][2];
		string tiempo[5];
		ifstream archivos[numero_de_pruebas];
		for (proto=0;proto<numero_de_protocolos;proto++){
			for (rtt=0;rtt<numero_de_rtts;rtt++){
				for (ext=0;ext<numero_de_extensiones;ext++){
					//Se crean variable para guardar el promedio total
					double promedio_total=0;
					double n_promedios=0;
					//Se abren los 5 archivos
					ifstream archivos[5];
					for(int i = 0; i < numero_de_pruebas; i++){
						char *buffer1 = new char[200];
						char *buffer2 = new char[200];
						string filename;
						char * delay_str = buffer1;
						char * imas1 = buffer2;
						sprintf(delay_str, "%d", delay[rtt]);
						int aux=i+1;
						sprintf(imas1, "%d", aux);
						filename = uBICacion+protocolo[proto] + d_sep + string(delay_str) + t_sep +
string(imas1) + first_ext + tipo[ext];
						//cout << "archivo: "<< filename <<"\n";
						archivos[i].open(filename.c_str());
						delete [] buffer1;
						delete [] buffer2;
					}
					//Abro el archivo de promedio y log
					char *buffer1 = new char[200];
					char * delay_str = buffer1;
					sprintf(delay_str, "%d", delay[rtt]);
					string nombre_salida = uBICacion_salida + protocolo[proto] + d_sep + string(delay_str) +
point + tipo[ext];
					ofstream archivo_salida(nombre_salida.c_str(), std::ofstream::out);

					string nombre_log = uBICacion_salida + protocolo[proto] + d_sep + string(delay_str) +
point + tipo[ext] + tail_log;
					ofstream archivo_log(nombre_log.c_str(), std::ofstream::out);
					delete [] buffer1;
					//Rescato las 5 lineas si es que archivos son legibles
					while (archivos[0].good() && archivos[1].good() && archivos[2].good() &&
archivos[3].good() && archivos[4].good()){
						for(int prueba=0;prueba<numero_de_pruebas;prueba++){
							getline(archivos[prueba],linea[prueba]);
						}
						//Se separan datos de cada linea
```

```
                                                int columna=0;
                                                int index_space=0;
                                                for(int i=0;i<numero_de_pruebas;i++){
                                                        columna=0;
                                                        do{
                                                                index_space=linea[i].find("\t");
                                                                //cout << "index_space: " << index_space << "\n";
                                                                if(index_space<0){
                                                                        datos[i][columna]=linea[i];
                                                                        //cout << datos[i][columna] << "\n\n";
                                                                }else{

        datos[i][columna++]=linea[i].substr(0,index_space);

        linea[i]=linea[i].substr(index_space+1,linea[i].length());
                                                                        //cout << line << "\n";
                                                                }
                                                        }while(index_space>0);
                                                }
                                                //En este punto se tienen los datos para la iteracion para cada archivo en array
datos[][]
                                                //Analizo los datos, guardo datos validos en un nuevo arreglo
                                                double datos_validos[numero_de_pruebas];
                                                int n_datos_validos=0;
                                                double dato_actual=0;
                                                for(int i=0;i<numero_de_pruebas;i++){
                                                        //cout << "datos[" << i << "][1] = " << datos[i][1] << "\n";
                                                        //dato_actual=strtoul(datos[i][1].c_str(),NULL,10);
                                                        dato_actual=strtod(datos[i][1].c_str(),NULL);
                                                        //printf("Dato Actual %f\n",dato_actual);
                                                        //cout << "dato_actual " << dato_actual << "\n";
                                                        if( (dato_actual>=0) ){
                                                                datos_validos[n_datos_validos++]=dato_actual;
                                                        }else{
                                                                n_datos_no_validos++;
                                                        }
                                                }
                                                //Tengo datos validados para calcular el promedio y escribir en archivo
                                                double promedio_actual=0;
                                                if(n_datos_validos>0){
                                                        for(int i=0;i<n_datos_validos;i++){

        promedio_actual=(promedio_actual*i+datos_validos[i])/(i+1.0);
                                                        }
                                                        //Escribo la entrada
                                                        archivo_salida << datos[0][0] << "\t" << promedio_actual << "\n";
                                                        //Agregar valor al promedio total

        promedio_total=((promedio_total*n_promedios)+promedio_actual)/(n_promedios+1.0);
                                                        n_promedios++;
                                                        if(n_datos_validos<numero_de_pruebas){ //Si hay datos
despreciados generar log
                                                                archivo_log << "Se promediaron solo " <<
n_datos_validos << " en tiempo " << datos[0][0] << "\n";
                                                        }
                                                }else{ // si no hay datos validos, no escribir en archivo, generar log
                                                        archivo_log << "No existen datos validos en tiempo " << datos[0][0]
<< "\n";
                                                }
                                        }//FIN DE LECTURA ARCHIVOS ASOCIADOS A UNA tipo

                                        //Cierro archivo de salida y log
                                        archivo_salida.close();
                                        archivo_log.close();
                                        //Cierro los 5 archivos de lectura
                                        for(int i = 0; i < numero_de_pruebas; i++){
                                                archivos[i].close();
                                        }
                                        //Agrego una linea al final (app) en archivo de dato vs_delay
                                        string nombre_archivo_delays = uBICacion_salida+
protocolo[proto]+"_vs_delay."+tipo[ext];

                                        ofstream archivo_salida_delays(nombre_archivo_delays.c_str(), std::ofstream::app);
                                        archivo_salida_delays << delay[rtt] << "\t" << promedio_total << "\n";
```

```
                                                archivo_salida_delays.close();
                                        }
                                }
                        }
                }
                return 0;
}
```

ANEXO C: CÓDIGO PARA LA REALIZACIÓN DE PRUEBAS EXPERIMENTALES

GENERACIÓN DE DATOS: generic

```
#! /bin/bash
#El primer parametro es el nombre del protocolo de congestion.
#El segundo parametro es el delay correspondiente en ms
#El tercer parametro se refiere al nombre de archivo de salida.
#Comente los protocolos que no usarÃ¡.
#Set Options
echo sleep $LyD_SLEEP_TIME
sleep $LyD_SLEEP_TIME

echo sudo $LyD_HOME_PATH/Testbed/Common/./opciones_buff
sudo $LyD_HOME_PATH/Testbed/Common/./opciones_buff

#ESTP
#----------------- ESTP only-----------------------------------------------
echo sudo /sbin/./insmod $LyD_HOME_PATH/Testbed/Common/Protocol/$LyD_ESTP_KO $LyD_ESTP_CONFIG
sudo /sbin/./insmod $LyD_HOME_PATH/Testbed/Common/Protocol/$LyD_ESTP_KO $LyD_ESTP_CONFIG
#-------------------------------------------------------------------------
echo sudo sysctl -w net.ipv4.TCP_congestion_control=$1
sudo sysctl -w net.ipv4.TCP_congestion_control=$1

echo FECHA=$(date +%F--%T)
FECHA=$(date +%F--%T)
echo $FECHA >> $LyD_HOME_PATH/Testbed/LyD_Testbed/TCP_probe/log/$3.log
echo sudo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./set_rate 10
sudo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./set_rate 10

echo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./testbed_loss $LyD_T_CONG $LyD_T_TOTAL $2 $LyD_LOW_LOSS
$LyD_MEDIUM_LOSS $LyD_HIGH_LOSS
sudo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./testbed_loss $LyD_T_CONG $LyD_T_TOTAL $2 $LyD_LOW_LOSS
$LyD_MEDIUM_LOSS $LyD_HIGH_LOSS

echo sudo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./connect_lyd
sudo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./connect_lyd

echo sudo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./set_rate -3
sudo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./set_rate -3

echo sudo cp $LyD_HOME_PATH/Testbed/LyD_Testbed/Temp/TCPprobe.out $LyD_HOME_PATH/Testbed/LyD_Testbed/TCP_probe/$3.info
sudo cp $LyD_HOME_PATH/Testbed/LyD_Testbed/Temp/TCPprobe.out $LyD_HOME_PATH/Testbed/LyD_Testbed/TCP_probe/$3.info

#close all
echo sudo /sbin/./rmmod TCP_probe
sudo /sbin/./rmmod TCP_probe
#----------ESTP only--------
echo sudo sysctl net.ipv4.TCP_congestion_control=CUBIC
sudo sysctl net.ipv4.TCP_congestion_control=CUBIC

echo sudo /sbin/./rmmod ESTP
sudo /sbin/./rmmod ESTP
#--------------------------
```

GENERACIÓN DE DATOS: Run_test

```
#! /bin/bash
#typeset cong_name[15]
. ../../lyd.env
d="_d_"
t="_t_"
cong_name[0]="ESTP"
cong_name[1]="CUBIC"
cong_name[2]="RENO"
cong_name[3]="BIC"
cong_name[5]="highspeed"
cong_name[6]="hTCP"
cong_name[7]="scalable"
cong_name[8]="westwood"
cong_name[9]="hybla"
cong_name[10]="vegas"
cong_name[11]="veno"
cong_name[12]="lp"
cong_name[13]="yeah"
cong_name[14]="illinois"

for cong in  "${cong_name[@]}"
do
        for (( delay=1; delay<=9; delay++ ))
        do
                for (( test=1; test<=10 ; test++ ))
                do
                        #echo "./generic $cong $delay $cong$d$delay$t$test"
                        #./generic $cong $delay $cong$d$delay$t$test
                        echo sleep $LyD_SLEEP_TIME
                        sleep $LyD_SLEEP_TIME

                        echo sudo $LyD_HOME_PATH/Testbed/Common/./opciones_buff
                        sudo $LyD_HOME_PATH/Testbed/Common/./opciones_buff
                        #ESTP
                        #----------------- ESTP only--------------------------------------------
                        echo sudo /sbin/./insmod $LyD_HOME_PATH/Testbed/Common/Protocol/$LyD_ESTP_KO
$LyD_ESTP_CONFIG
                        sudo /sbin/./insmod $LyD_HOME_PATH/Testbed/Common/Protocol/$LyD_ESTP_KO $LyD_ESTP_CONFIG
                        #-------------------------------------------------------------------------
                        echo sudo sysctl -w net.ipv4.TCP_congestion_control=$cong
                        sudo sysctl -w net.ipv4.TCP_congestion_control=$cong

                        echo FECHA=$(date +%F--%T)
                        FECHA=$(date +%F--%T)
                        echo $FECHA >> $LyD_HOME_PATH/Testbed/LyD_Testbed/TCP_probe/log/$cong$d$delay$t$test.log
                        echo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./set_rate 10
                        $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./set_rate 10

                        echo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./testbed_loss $LyD_T_CONG $LyD_T_TOTAL
$delay $LyD_LOW_LOSS $LyD_MEDIUM_LOSS $LyD_HIGH_LOSS
                        $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./testbed_loss $LyD_T_CONG $LyD_T_TOTAL $delay
$LyD_LOW_LOSS $LyD_MEDIUM_LOSS $LyD_HIGH_LOSS &

                        #echo sudo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./connect_lyd
                        echo sudo /sbin/./insmod $LyD_HOME_PATH/Testbed/Common/Protocol/TCP_probe.ko
                        sudo /sbin/./insmod $LyD_HOME_PATH/Testbed/Common/Protocol/TCP_probe.ko

                        echo sudo chmod 640 /proc/net/TCPprobe
                        sudo chmod 640 /proc/net/TCPprobe

                        echo sudo cat /proc/net/TCPprobe > $LyD_HOME_PATH/Testbed/LyD_Testbed/Temp/TCPprobe.out &
                        sudo cat /proc/net/TCPprobe > $LyD_HOME_PATH/Testbed/LyD_Testbed/Temp/TCPprobe.out &

                        echo TCPCAP=$!
                        TCPCAP=$!

                        echo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./iperf -i $LyD_DISPLAY_TIME -t
$LyD_TEST_TIME -f Kbits -c $LyD_HOST -p $LyD_PUERTO
```

```
                                $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./iperf -i $LyD_DISPLAY_TIME -t $LyD_TEST_TIME -f
Kbits -c $LyD_HOST -p $LyD_PUERTO
                                #sudo iperf -i 1, uno indica cuantas veces escribe el troughput en pantalla.(resolucion testbed_loss)
                                #sudo iperf -i 10 -t 180 -c localhost
                                echo sudo kill $TCPCAP
                                sudo kill $TCPCAP

                                echo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./set_rate -3
                                $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./set_rate -3

                                echo sudo cp $LyD_HOME_PATH/Testbed/LyD_Testbed/Temp/TCPprobe.out
$LyD_HOME_PATH/Testbed/LyD_Testbed/TCP_probe/$cong$d$delay$t$test.info
                                sudo cp $LyD_HOME_PATH/Testbed/LyD_Testbed/Temp/TCPprobe.out
$LyD_HOME_PATH/Testbed/LyD_Testbed/TCP_probe/$cong$d$delay$t$test.info

                                #close all
                                echo sudo /sbin/./rmmod TCP_probe
                                sudo /sbin/./rmmod TCP_probe
                                #----------ESTP only--------
                                echo sudo sysctl net.ipv4.TCP_congestion_control=CUBIC
                                sudo sysctl net.ipv4.TCP_congestion_control=CUBIC

                                echo sudo /sbin/./rmmod TCP_ESTP
                                sudo /sbin/./rmmod TCP_ESTP
                                #---------------------------
                        done
                done

                for (( delay=10; delay<=200; delay=delay+10))
                do
                        for (( test=1; test<=10 ; test++ ))
                        do
                                #echo "./generic $cong $delay $cong$d$delay$t$test"
                                #./generic $cong $delay $cong$d$delay$t$test
                                echo sleep $LyD_SLEEP_TIME
                                sleep $LyD_SLEEP_TIME

                                echo sudo $LyD_HOME_PATH/Testbed/Common/./opciones_buff
                                sudo $LyD_HOME_PATH/Testbed/Common/./opciones_buff

                                #ESTP
                                #----------------- ESTP only-----------------------------------------------
                                echo sudo /sbin/./insmod $LyD_HOME_PATH/Testbed/Common/Protocol/$LyD_ESTP_KO
$LyD_ESTP_CONFIG
                                sudo /sbin/./insmod $LyD_HOME_PATH/Testbed/Common/Protocol/$LyD_ESTP_KO $LyD_ESTP_CONFIG
                                #-------------------------------------------------------------------------
                                echo sudo sysctl -w net.ipv4.TCP_congestion_control=$cong
                                sudo sysctl -w net.ipv4.TCP_congestion_control=$cong

                                echo FECHA=$(date +%F--%T)
                                FECHA=$(date +%F--%T)
                                echo $FECHA >> $LyD_HOME_PATH/Testbed/LyD_Testbed/TCP_probe/log/$cong$d$delay$t$test.log
                                echo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./set_rate 10
                                $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./set_rate 10

                                echo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./testbed_loss $LyD_T_CONG $LyD_T_TOTAL
$delay $LyD_LOW_LOSS $LyD_MEDIUM_LOSS $LyD_HIGH_LOSS
                                $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./testbed_loss $LyD_T_CONG $LyD_T_TOTAL $delay
$LyD_LOW_LOSS $LyD_MEDIUM_LOSS $LyD_HIGH_LOSS &

                                #echo sudo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./connect_lyd
                                echo sudo /sbin/./insmod $LyD_HOME_PATH/Testbed/Common/Protocol/TCP_probe.ko
                                sudo /sbin/./insmod $LyD_HOME_PATH/Testbed/Common/Protocol/TCP_probe.ko

                                echo sudo chmod 640 /proc/net/TCPprobe
                                sudo chmod 640 /proc/net/TCPprobe

                                echo sudo cat /proc/net/TCPprobe > $LyD_HOME_PATH/Testbed/LyD_Testbed/Temp/TCPprobe.out &
                                sudo cat /proc/net/TCPprobe > $LyD_HOME_PATH/Testbed/LyD_Testbed/Temp/TCPprobe.out &

                                echo TCPCAP=$!
                                TCPCAP=$!
```

```
                                    echo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./iperf -i $LyD_DISPLAY_TIME -t
$LyD_TEST_TIME -f Kbits -c $LyD_HOST -p $LyD_PUERTO
                                    $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./iperf -i $LyD_DISPLAY_TIME -t $LyD_TEST_TIME -f
Kbits -c $LyD_HOST -p $LyD_PUERTO
                                    #sudo iperf -i 1, uno indica cuantas veces escribe el troughput en pantalla.(resolucion testbed_loss)
                                    #sudo iperf -i 10 -t 180 -c localhost
                                    echo sudo kill $TCPCAP
                                    sudo kill $TCPCAP

                                    echo $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./set_rate -3
                                    $LyD_HOME_PATH/Testbed/LyD_Testbed/Rutinas/./set_rate -3

                                    echo sudo cp $LyD_HOME_PATH/Testbed/LyD_Testbed/Temp/TCPprobe.out
$LyD_HOME_PATH/Testbed/LyD_Testbed/TCP_probe/$cong$d$delay$t$test.info
                                    sudo cp $LyD_HOME_PATH/Testbed/LyD_Testbed/Temp/TCPprobe.out
$LyD_HOME_PATH/Testbed/LyD_Testbed/TCP_probe/$cong$d$delay$t$test.info

                                    #close all
                                    echo sudo /sbin/./rmmod TCP_probe
                                    sudo /sbin/./rmmod TCP_probe
                                    #----------ESTP only--------
                                    echo sudo sysctl net.ipv4.TCP_congestion_control=CUBIC
                                    sudo sysctl net.ipv4.TCP_congestion_control=CUBIC

                                    echo sudo /sbin/./rmmod TCP_ESTP
                                    sudo /sbin/./rmmod TCP_ESTP
                                    #----------------------------
                        done
            done
done
```

IMPLEMENTACIÓN DE TESTBED: Set_rate

```
#include <stdio.h>
//#include <iostream>
//#include <fstream>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define SHM_SIZE 1024  /* make it a 1K shared memory segment */
#define SIZE 512

/*Esta aplicacion escribe en el segmento de memoria compartida
 *donde escribe su salida el archivo iperf modificado
 */

int main(int argc, char *argv[])
{
            char file_key[100];
            strcpy(file_key,getenv("LyD_HOME_PATH"));
            strcat(file_key,"/key");
            printf("%s\n",file_key);
            key_t key;
   int shmid;
   // char *datac,*rate;
   double   rate;
   void* datac = (void*)&rate;

   //int mode,i;

   /* make the key: */
   if ((key = ftok(file_key, 'R')) == -1) {
      perror("ftok");
      exit(1);
```

```
}

/* connect to (and possibly create) the segment: */
if ((shmid = shmget(key, SHM_SIZE, 0644 | IPC_CREAT)) == -1) {
    perror("shmget");
    exit(1);
}

/* attach to the segment to get a pointer to it: */
datac = shmat(shmid, (void *)0, 0);
if (datac == (char *)(-1)) {
    perror("shmat");
    exit(1);
}

            * (double*) datac = atoi(argv[1]);
            return 0;
            }
```

IMPLEMENTACIÓN DE TESTBED: Testbed_loss_vs_Throughput

```
#include <stdio.h>
//#include <iostream>
//#include <fstream>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHM_SIZE 1024  /* make it a 1K shared memory segment */
#define SIZE 512
//-------------DEFINICION DE CIR y EIR-----------------
//#define CIR atoi(getenv("LyD_CIR"))
//#define EIR 1000
//#define DELAY 10 Se agrega DELAY como variable si no se fija en 10
//---------------------------------------------------------
/*
El programa recibe como
primer argumento: tiempo de congestion
segundo argumento: tiempo total
tercer argumento: delay
------------------------
Si no existe tercer argumento el delay es 10ms
Si no existe segundo argumento el tiempo total es 10s y delay 10ms
Si no existen argumentos el delay es 10ms tiempo total 10s y congestion 2s
*/
int main(int argc, char *argv[])
{
            int CIR=atoi(getenv("LyD_CIR"));
   int EIR=atoi(getenv("LyD_EIR"));
            struct datos{
            char palabra[100];
            } datos1[3];

            int i,k,j,total,delay,timer;
            char *h_loss,*m_loss,*l_loss;
            char T[10];

            char palabra[100],palabra2[20];
            char file_key[100];
            strcpy(file_key,getenv("LyD_HOME_PATH"));
            strcat(file_key,"/key");
            printf("%s\n",file_key);
            char syskey[110];
            sprintf(syskey,"> %s",file_key);
```

```
	//h_loss = argv[4];
	//printf("\n h_loss=%s... ",h_loss);

	printf("\n\t############ INDICACIONES ############\n");
	printf("\tEl programa recibe como:\n");
	printf("\tprimer argumento: tiempo de congestion (2s default) \n");
	printf("\tsegundo argumento: tiempo total (10s default) \n");
	printf("\ttercer argumento: delay (10ms default)\n");
	printf("\tSi recibe mas de 3 argumentos, necesariamente debe recibir 6\n");
	printf("\tcuarto argumento: porcentaje baja perdida (0.0001 default)\n");
	printf("\tquinto argumento: porcentaje mediana perdida (0.01 default)\n");
	printf("\tsexto argumento : porcentaje alta perdida (1 default)\n");
	printf("\t########################################");
	system(syskey);
	printf("\n Una llave se ha creado en %s/ \n",file_key);
	//printf("\n argc = %d",argc);
	if (argc>4){
		i=atoi(argv[1]);
		k=atoi(argv[2])-i;
		total=atoi(argv[2]);
		delay=atoi(argv[3]);
		l_loss=argv[4];
		m_loss=argv[5];
		h_loss=argv[6];
		  }
	else if(argc>3){	//con 3 argumentos
		i=atoi(argv[1]);	//tiempo de congestion
		//printf("argv[1]=%d\n",i);
		k=atoi(argv[2])-i;	//tiempo de no congestion (total - no_cong) tiempo total =10[s]
		//printf("argv[2]-i=%d\n",k);
		total=atoi(argv[2]);
		//printf("total=%d\n",total);
		delay=atoi(argv[3]);
		//printf("delay=%d\n",delay);
		h_loss="1";
		m_loss="0.01";
		l_loss="0.0001";
	}else if(argc>2){	//con 2 argumentos
		i=atoi(argv[1]);	//tiempo de congestion
		k=atoi(argv[2])-i;
		total=atoi(argv[2]);	//tiempo de no congestion		tiempo total =10[s]
		delay=10;
		h_loss="1";
		m_loss="0.01";
		l_loss="0.0001";
	}else if (argc>1){//solo un argumento
		i=atoi(argv[1]);
		total=10;
		k=total-i;
		delay=10;//ms
		h_loss="1";
		m_loss="0.01";
		l_loss="0.0001";
	}else {			//sin argumentos
		i=2;
		total=10;
		k=total-i;
		delay=10;
		h_loss="1";
		m_loss="0.01";
		l_loss="0.0001";
		}
	j=0;

key_t key;
int shmid;
double *data;
int mode;
double rate;

/* make the key: */
if ((key = ftok(file_key, 'R')) == -1) {
```

```
        perror("ftok");
        exit(1);
    }

    /* connect to (and possibly create) the segment: */
    if ((shmid = shmget(key, SHM_SIZE, 0644 | IPC_CREAT)) == -1) {
        perror("shmget");
        exit(1);
    }

    /* attach to the segment to get a pointer to it: */
    data = shmat(shmid, (void *)0, 0);
    if (data == (double *)(-1)) {
        perror("shmat");
        exit(1);
    }
            //rate=600;// ------------SOLO PARA PROBAR...ELIMINAR Y DESCOMENTAR LINEA SIGUIENTE
            rate = -1;//

            timer=0;
    /* read or modify the segment, based on the command line: */

//INIT
sprintf(palabra,"sudo tc qdisc replace dev eth0 root netem delay %dms loss %s%%\n", delay, m_loss);
                        system(palabra);
                        printf("%s \n", palabra);


     while(rate>-2) {
            //if (rate != *(double *) data){
            rate = * (double *) data;
//-----------------------SE DEFINEN LAS PERDIDAS DEPENDIENTES DE THROUGHPUT--------------------------
            /* CASO QUE Throughput < CIR se disminuye la perdida*/


                        if (rate<CIR){
sprintf(palabra,"sudo tc qdisc replace dev eth0 root netem delay %dms loss %s%%\n", delay, l_loss);
                        system(palabra);
                        printf("%s \n", palabra);
                        }
                        else{

                        if(timer==(k*10)){
sprintf(palabra,"sudo tc qdisc replace dev eth0 root netem delay %dms loss %s%%\n", delay, h_loss);
                        system(palabra);
                        printf("%s \n", palabra);
                        //sprintf(T,"sleep %d",k);
                        //system(T);
                        }

                        if(timer==0) {
sprintf(palabra,"sudo tc qdisc replace dev eth0 root netem delay %dms loss %s%%\n", delay, m_loss);
                        system(palabra);
                        printf("%s \n",palabra);
                        //sprintf(T,"sleep %d",i);
                        //system(T);
                        }
                        }

                        system("sleep 0.092"); //No es 0.1 tomando en cuenta el desfaz por operacion
                        timer=timer+1;
                        //printf("timer= %d\n",timer);
                        if (timer>total*10){
                        timer=0;
                        }


//-----------------------FIN DEL CAMBIO-------------------------------------------------------------

            /*printf("segment contains: \"%lf\"\n", *(double *)data);*/
/*          sprintf(palabra,"sudo tc qdisc replace dev eth0 root netem delay 20ms loss 1%%\n");
            printf(palabra);
            system(palabra);  */
```

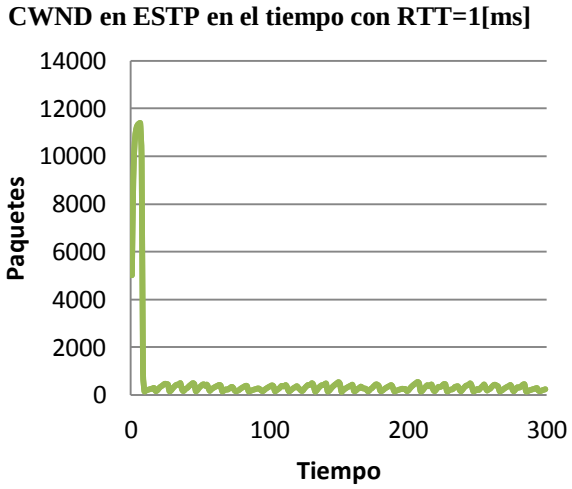
```
                }
//}//end IF

    /* detach from the segment: */
    if (shmdt(data) == -1) {
        perror("shmdt");
        exit(1);
    }

return 0;


}//FIN MAIN
```

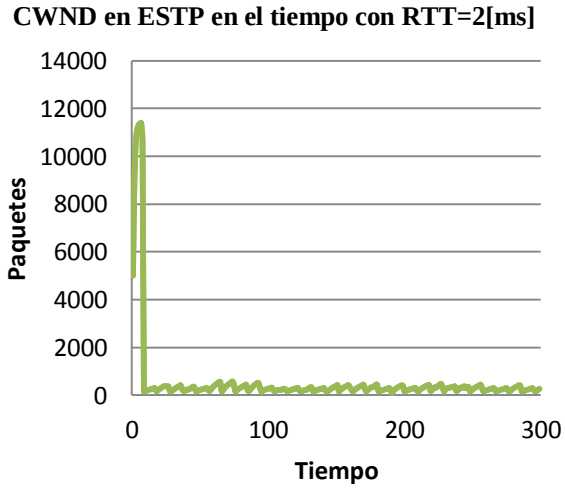
ANEXO D: RESULTADOS OBTENIDOS

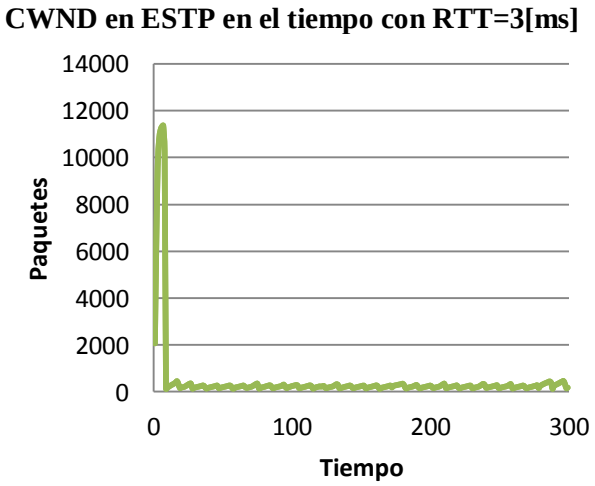
En la siguiente sección se mostrarán los resultados obtenidos en los protocolos ESTP, BIC, CUBIC y RENO en relación a flujo de datos, CWND y SSThreshold. Los datos se mostrarán mediante gráficos, considerando RTT 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90 y 100.

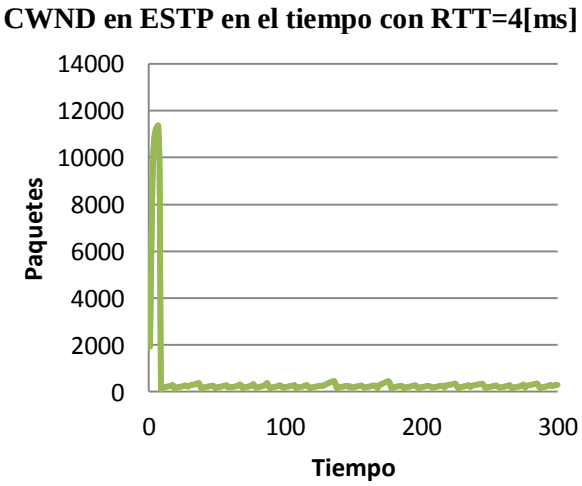
1. CWND EN EL TIEMPO CON DIFERENTES RTT

A. ESTP

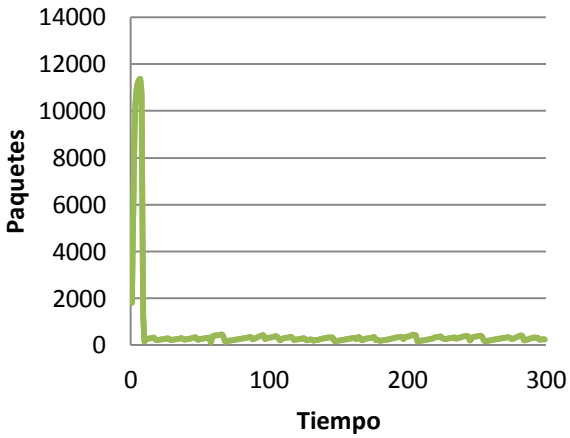




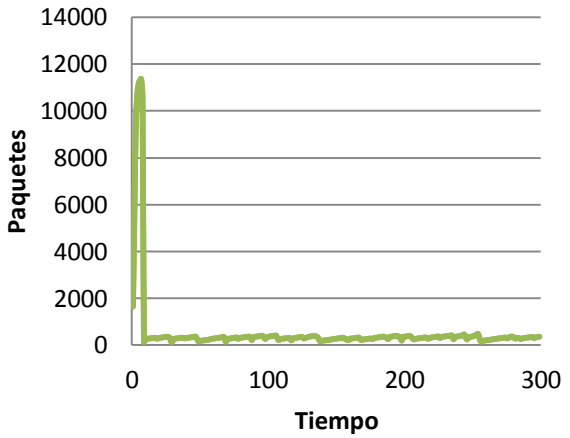




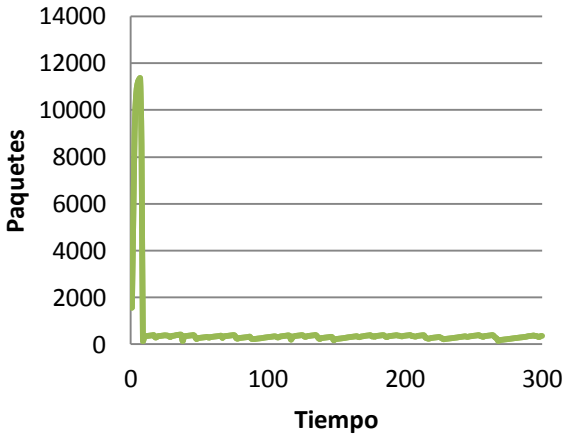
CWND en ESTP en el tiempo con RTT=5[ms]



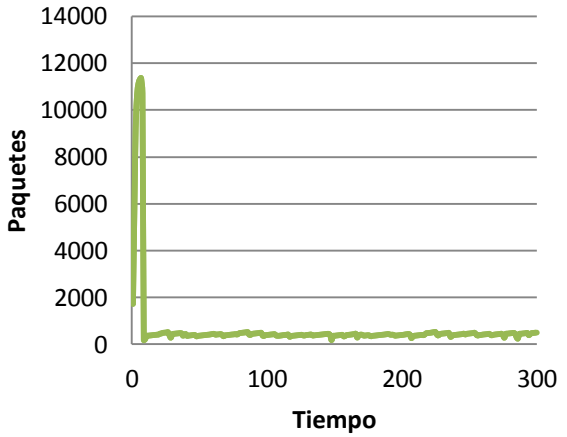
CWND en ESTP en el tiempo con RTT=6[ms]



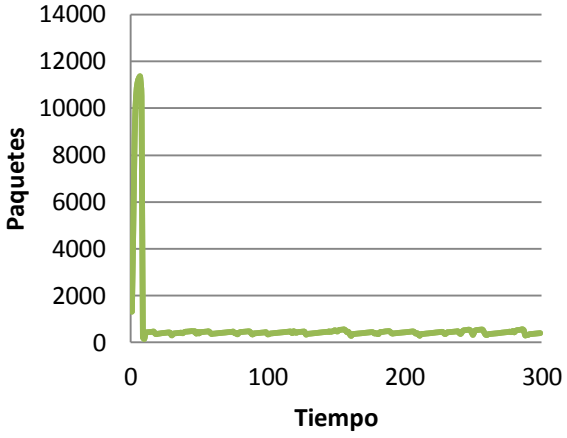
CWND en ESTP en el tiempo con RTT=7[ms]



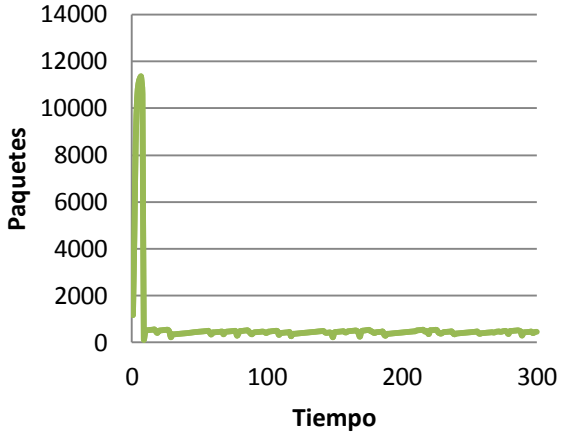
CWND en ESTP en el tiempo con RTT=8[ms]



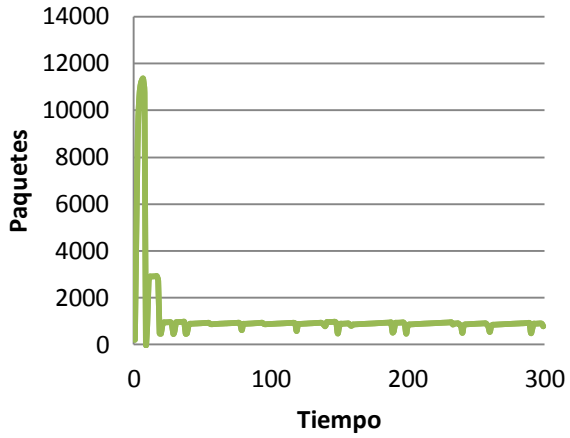
CWND en ESTP en el tiempo con RTT=9[ms]



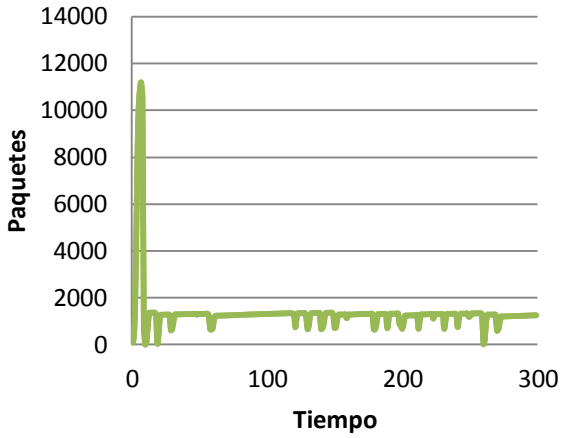
CWND en ESTP en el tiempo con RTT=10[ms]



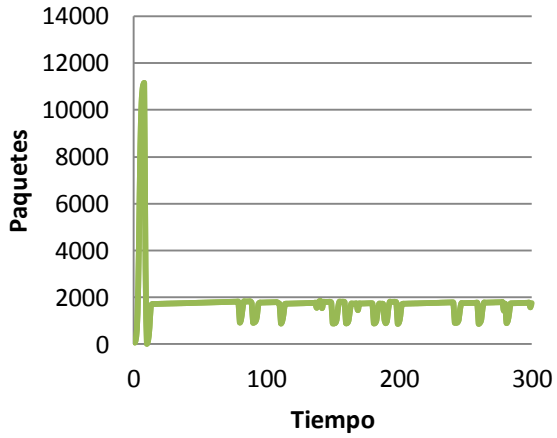
CWND en ESTP en el tiempo con RTT=20[ms]



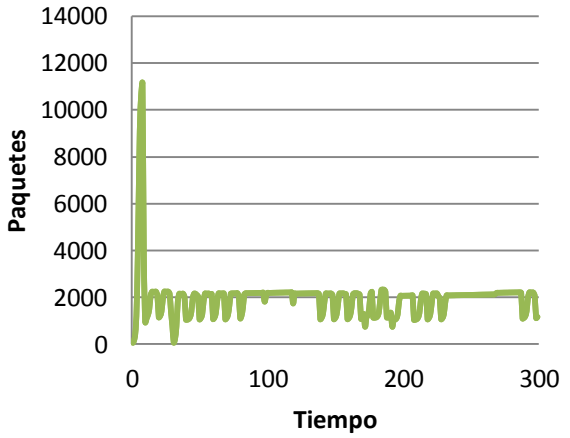
CWND en ESTP en el tiempo con RTT=30[ms]



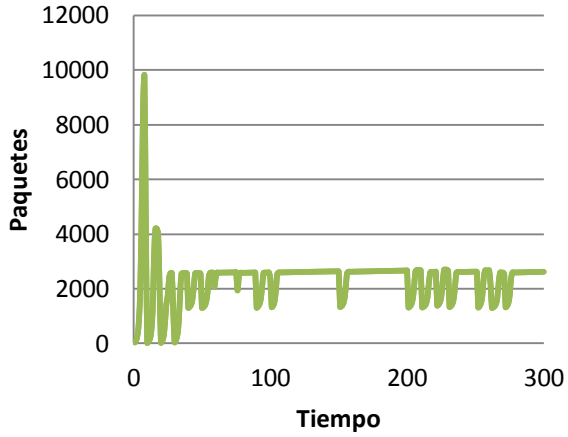
CWND en ESTP en el tiempo con RTT=40[ms]



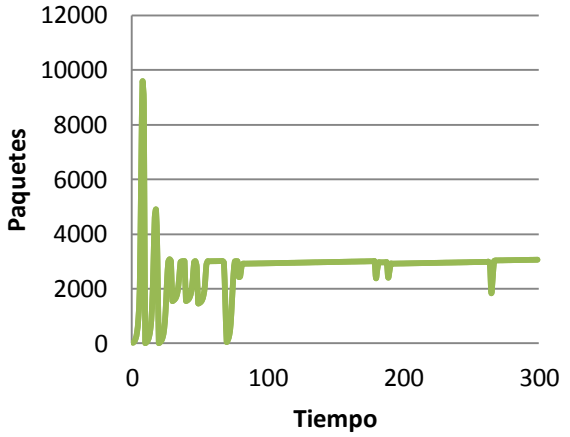
CWND en ESTP en el tiempo con RTT=50[ms]



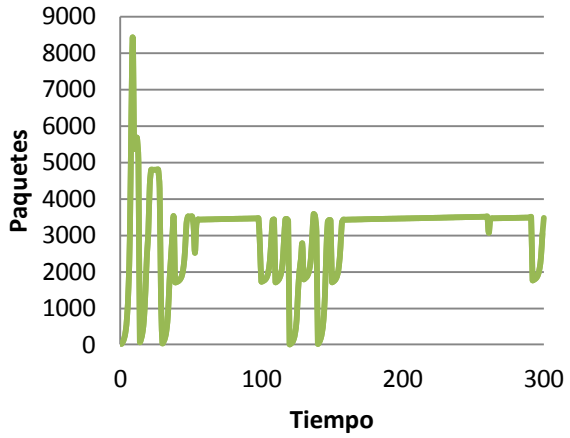
CWND en ESTP en el tiempo con RTT=60[ms]



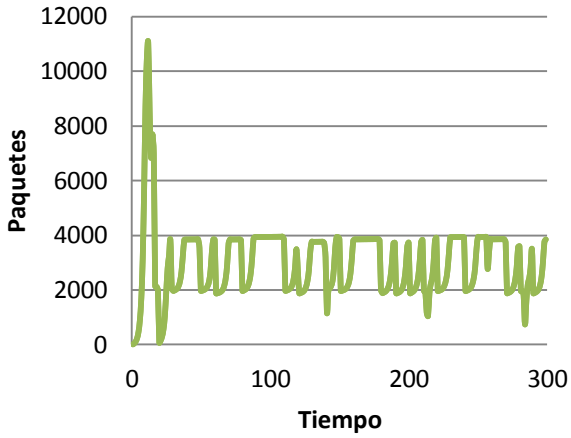
CWND en ESTP en el tiempo con RTT=70[ms]



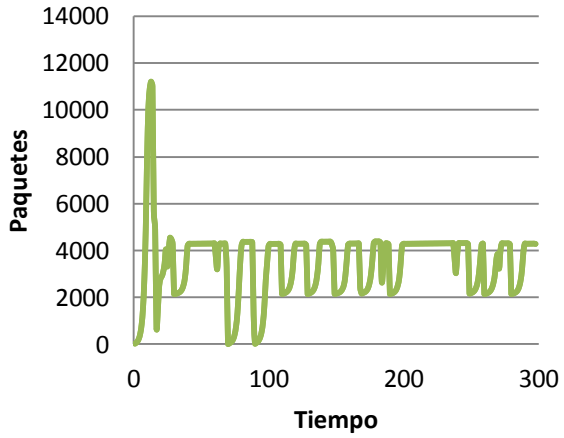
CWND en ESTP en el tiempo con RTT=80[ms]



CWND en ESTP en el tiempo con RTT=90[ms]

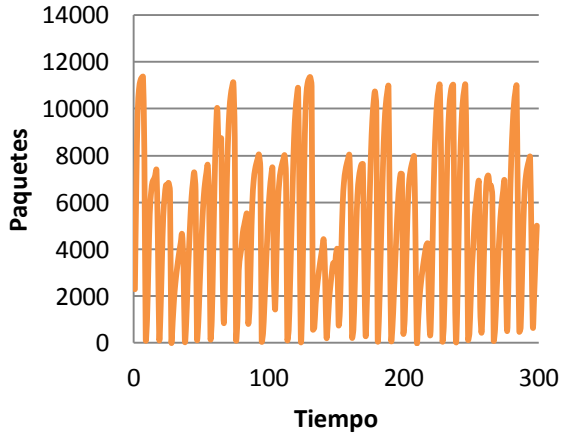


CWND en ESTP en el tiempo con RTT=100[ms]

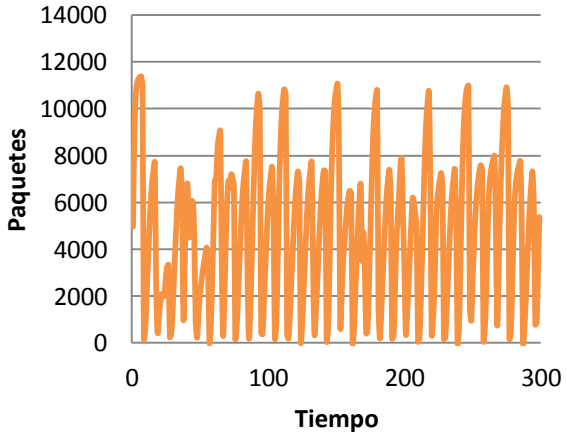


B. BIC

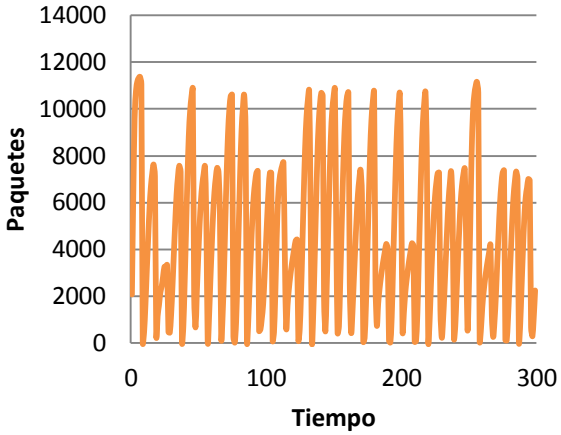
CWND en BIC en el tiempo con RTT=1[ms]



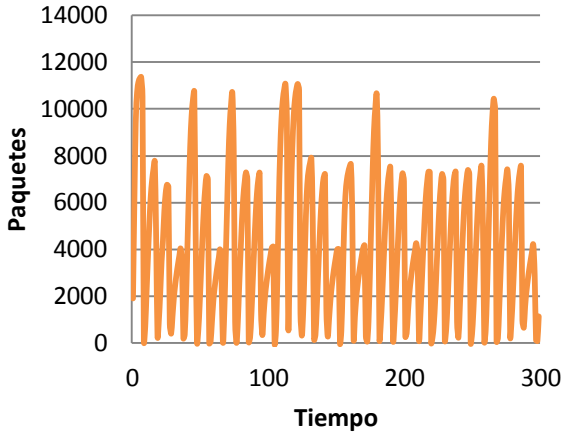
CWND en BIC en el tiempo con RTT=2[ms]



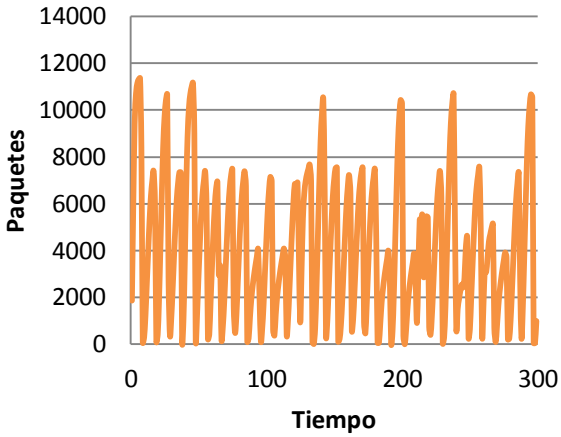
CWND en BIC en el tiempo con RTT=3[ms]



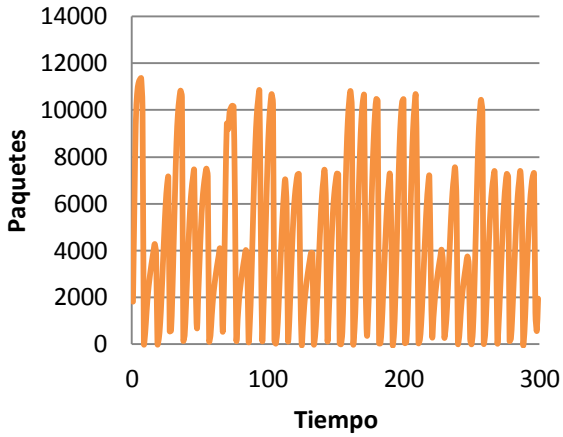
CWND en BIC en el tiempo con RTT=4[ms]



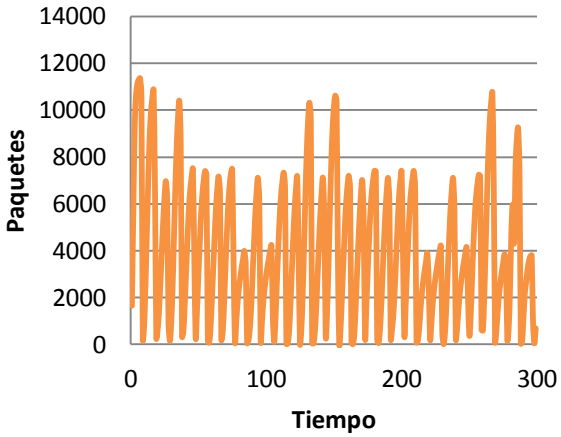
CWND en BIC en el tiempo con RTT=5[ms]



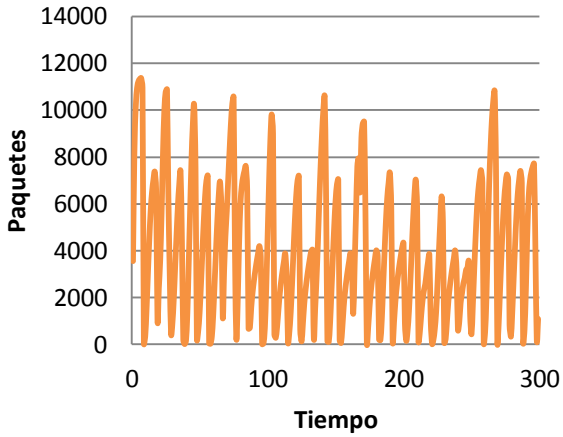
CWND en BIC en el tiempo con RTT=6[ms]



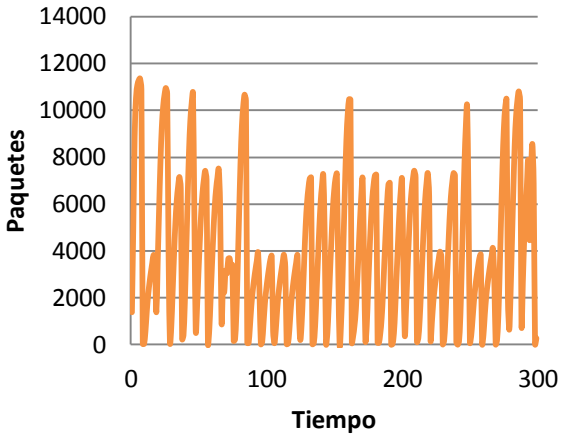
CWND en BIC en el tiempo con RTT=7[ms]



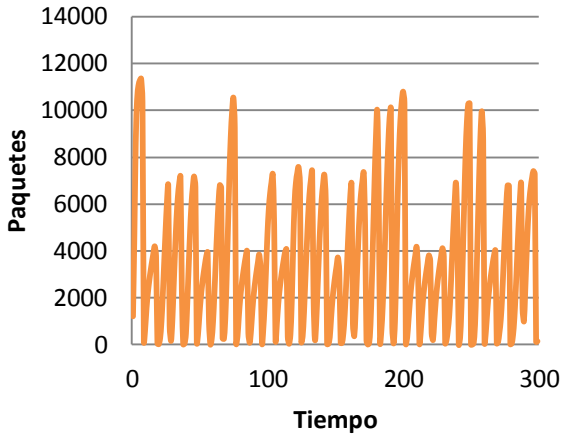
CWND en BIC en el tiempo con RTT=8[ms]



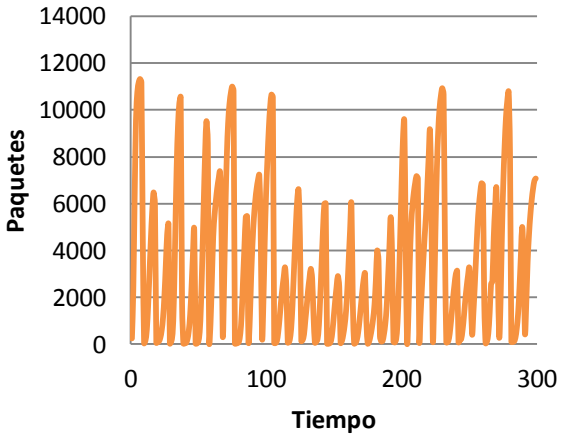
CWND en BIC en el tiempo con RTT=9[ms]



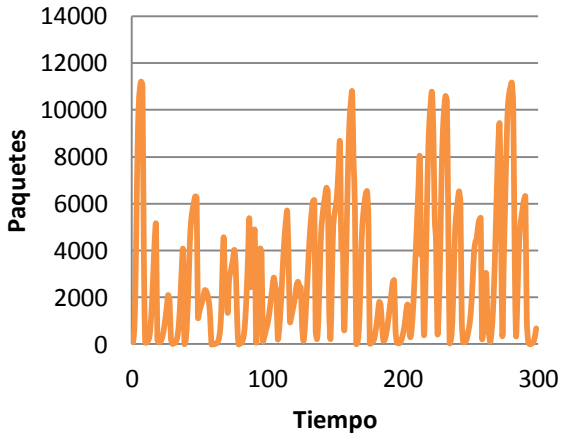
CWND en BIC en el tiempo con RTT=10[ms]



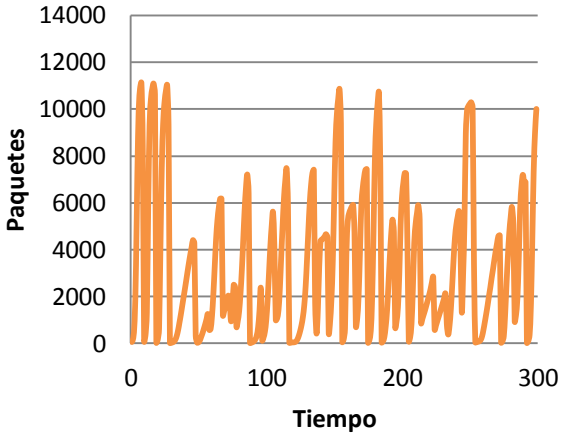
CWND en BIC en el tiempo con RTT=20[ms]



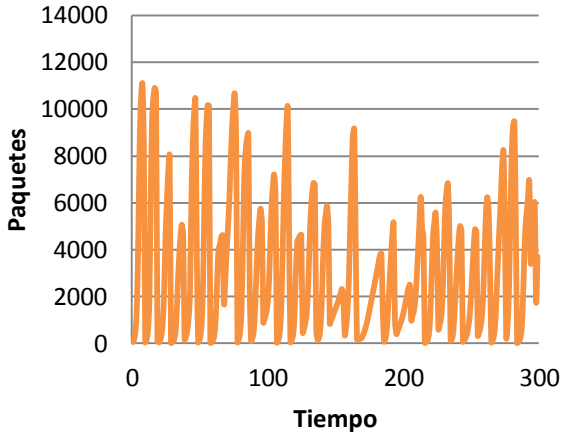
CWND en BIC en el tiempo con RTT=30[ms]



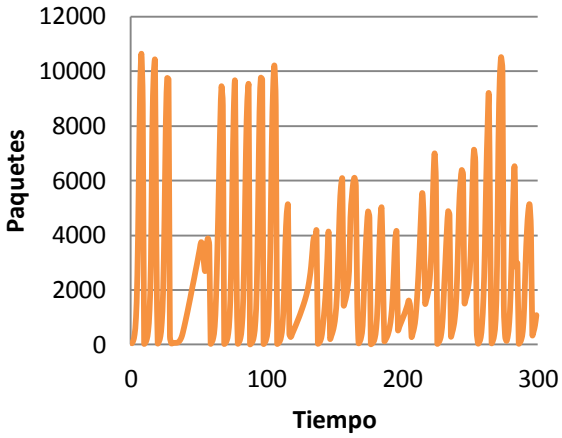
CWND en BIC en el tiempo con RTT=40[ms]



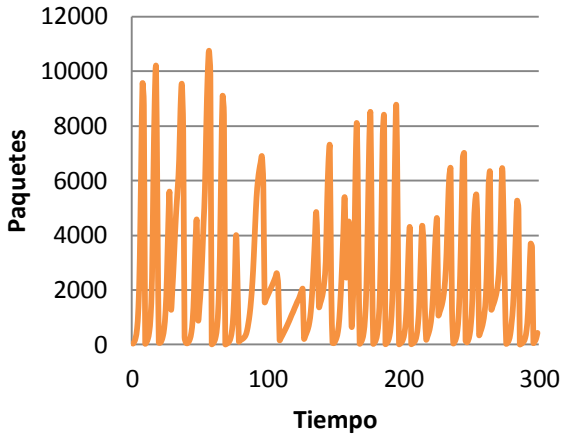
CWND en BIC en el tiempo con RTT=50[ms]



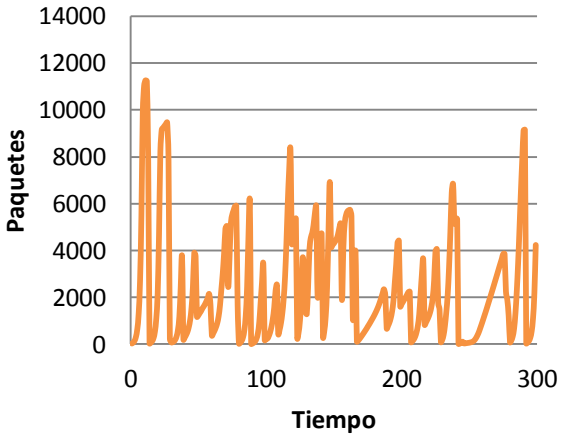
CWND en BIC en el tiempo con RTT=60[ms]



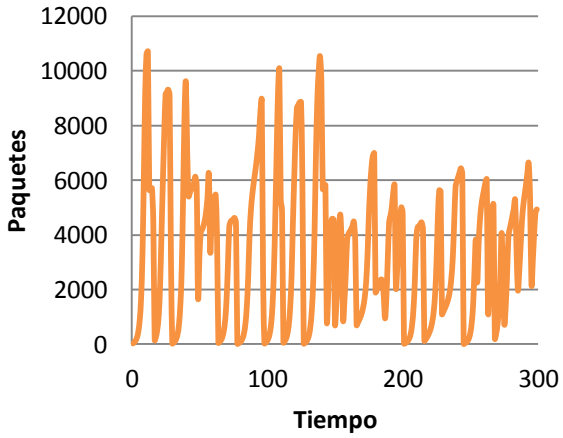
CWND en BIC en el tiempo con RTT=70[ms]



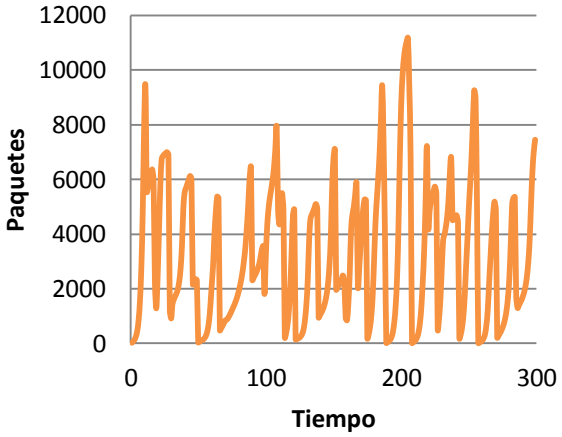
CWND en BIC en el tiempo con RTT=80[ms]



CWND en BIC en el tiempo con RTT=90[ms]

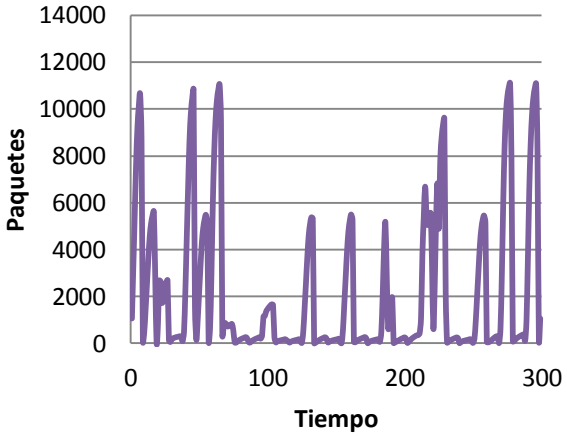


CWND en BIC en el tiempo con RTT=100[ms]

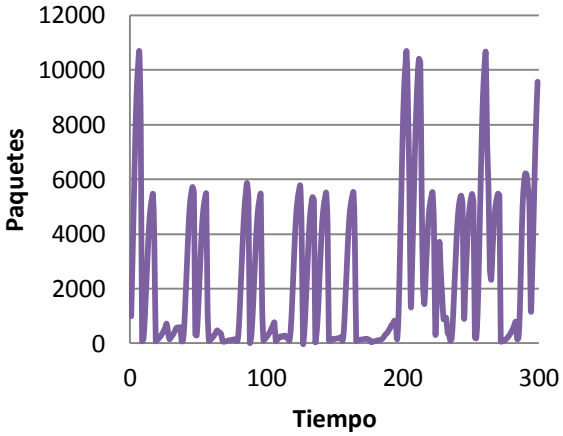


C. CUBIC

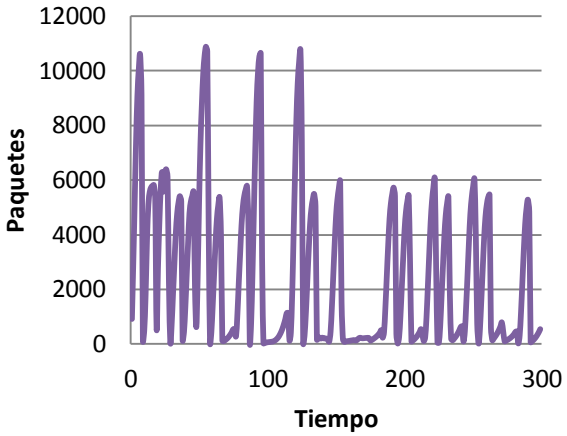
CWND en CUBIC en el tiempo con RTT=1[ms]



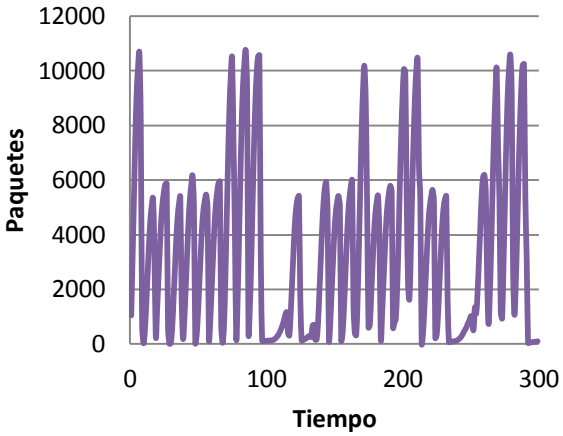
CWND en CUBIC en el tiempo con RTT=2[ms]



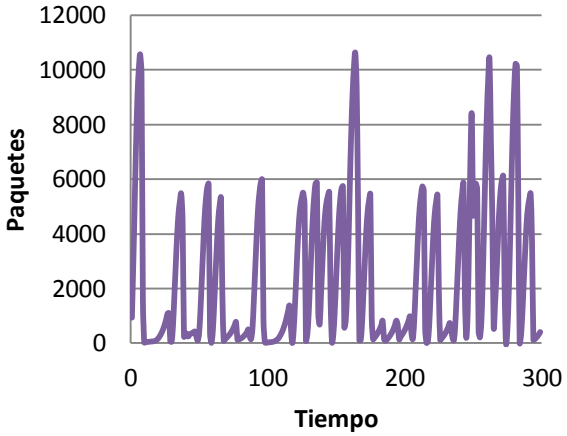
CWND en CUBIC en el tiempo con RTT=3[ms]



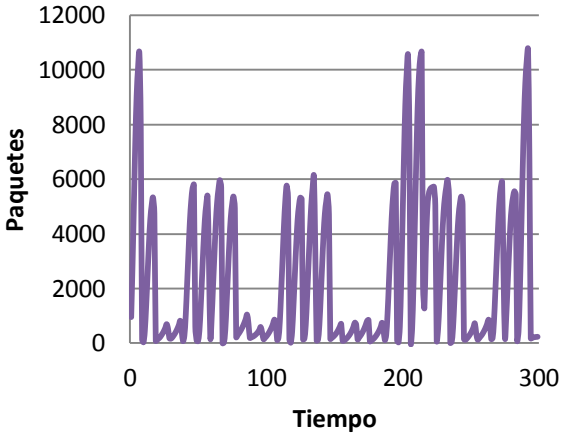
CWND en CUBIC en el tiempo con RTT=4[ms]



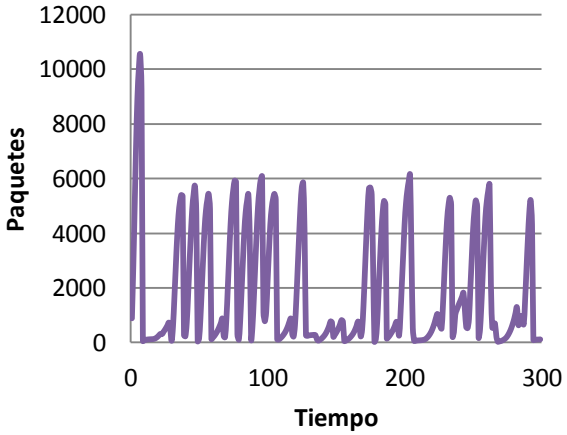
CWND en CUBIC en el tiempo con RTT=5[ms]



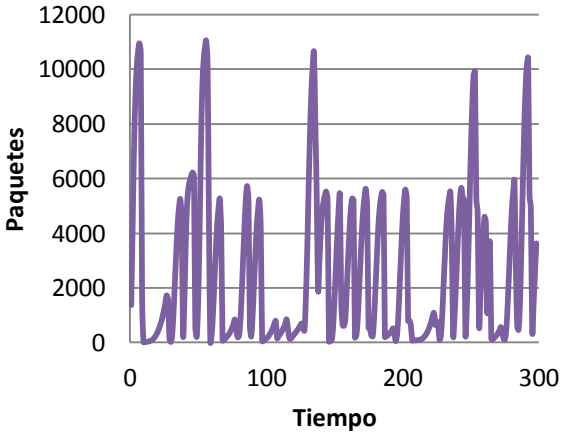
CWND en CUBIC en el tiempo con RTT=6[ms]



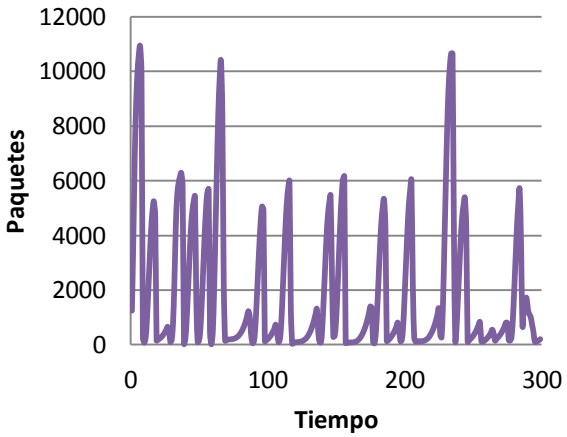
CWND en CUBIC en el tiempo con RTT=7[ms]



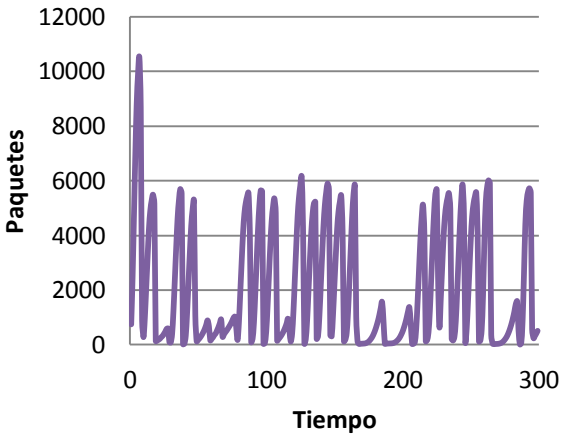
CWND en CUBIC en el tiempo con RTT=8[ms]



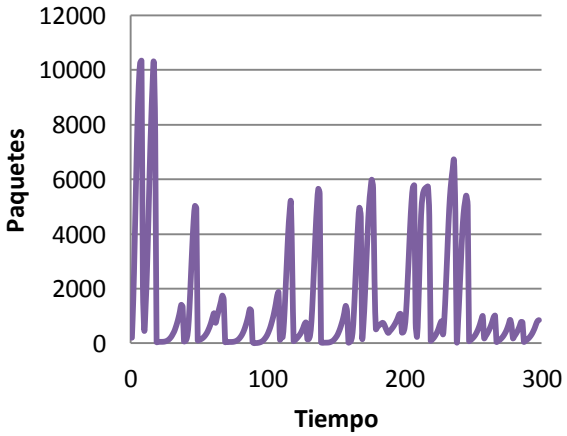
CWND en CUBIC en el tiempo con RTT=9[ms]



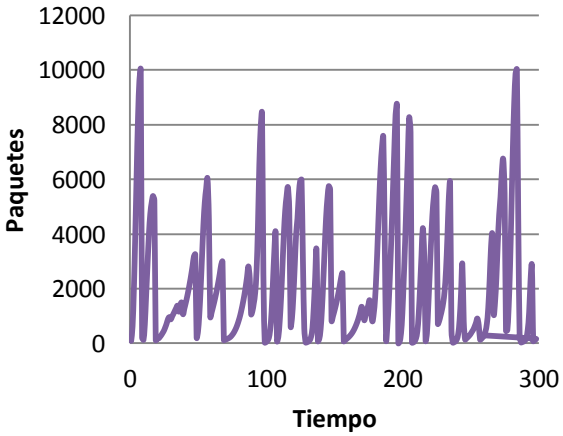
CWND en CUBIC en el tiempo con RTT=10[ms]



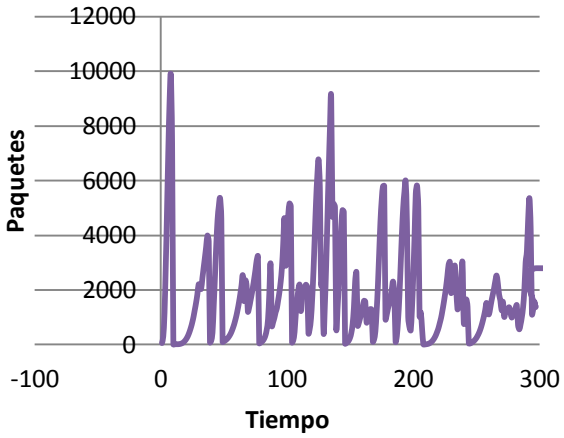
CWND en CUBIC en el tiempo con RTT=20[ms]



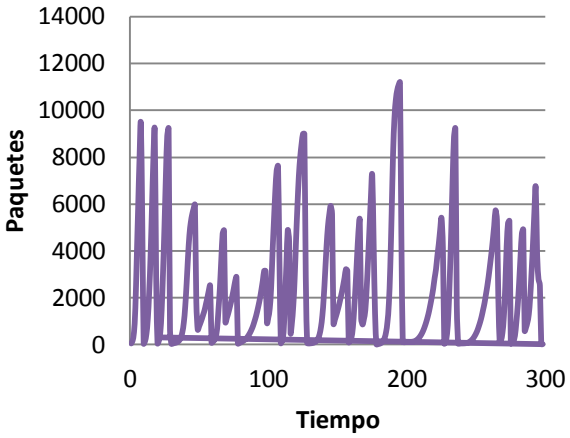
CWND en CUBIC en el tiempo con RTT=30[ms]



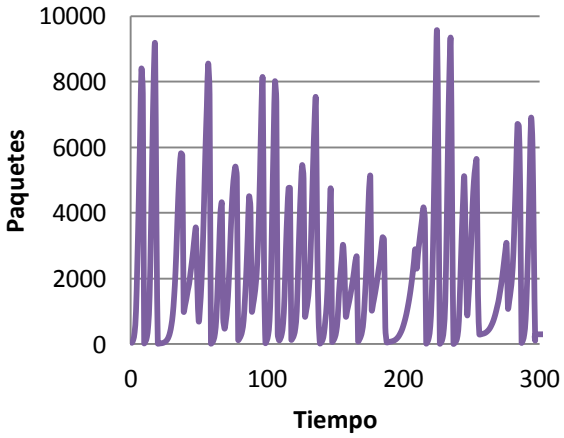
CWND en CUBIC en el tiempo con RTT=40[ms]



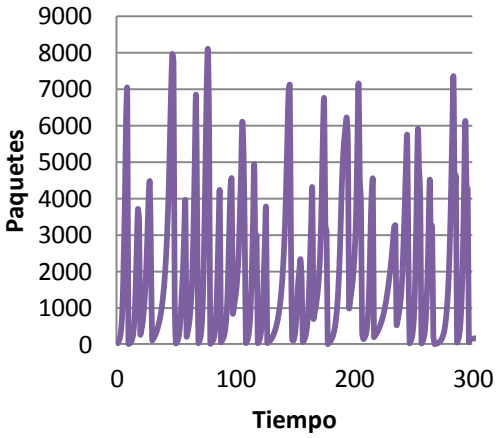
CWND en CUBIC en el tiempo con RTT=50[ms]



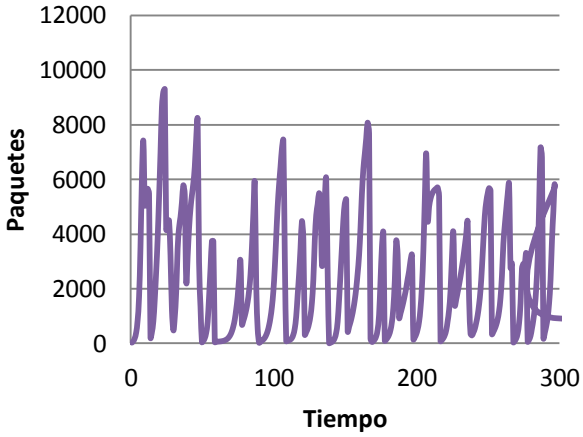
CWND en CUBIC en el tiempo con RTT=60[ms]



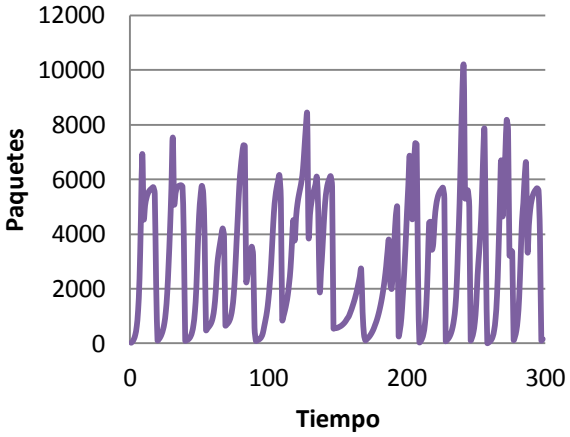
CWND en CUBIC en el tiempo con RTT=70[ms]



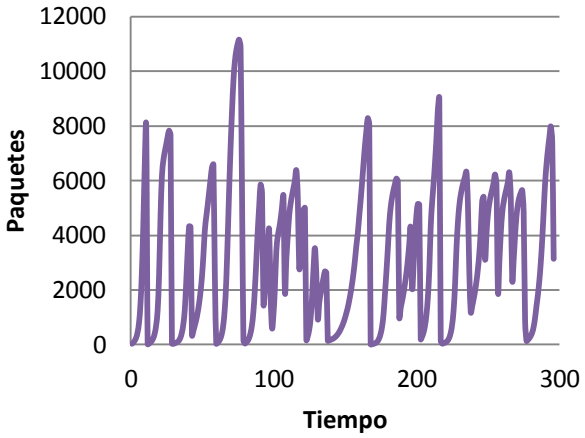
CWND en CUBIC en el tiempo con RTT=80[ms]



CWND en CUBIC en el tiempo con RTT=90[ms]

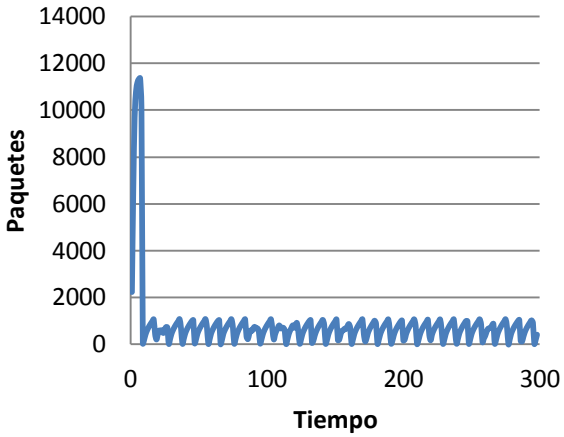


CWND en CUBIC en el tiempo con RTT=100[ms]

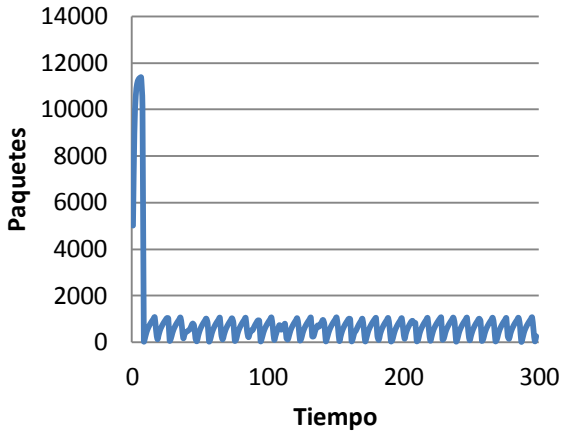


D. RENO

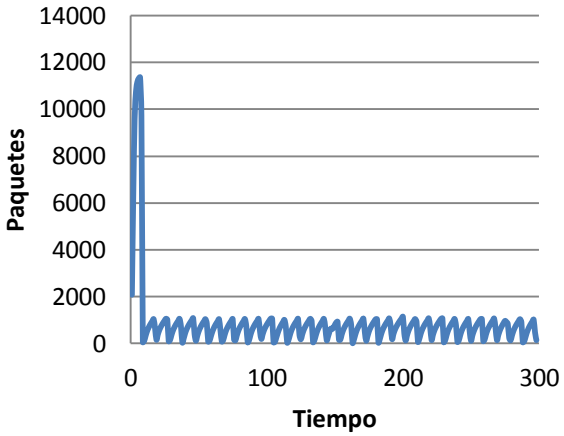
CWND en RENO en el tiempo con RTT=1[ms]



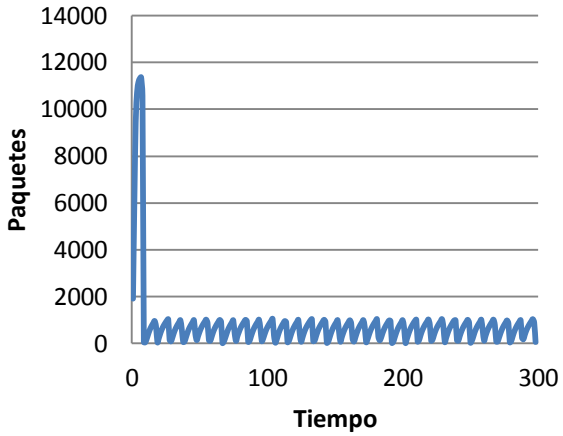
CWND en RENO en el tiempo con RTT=2[ms]



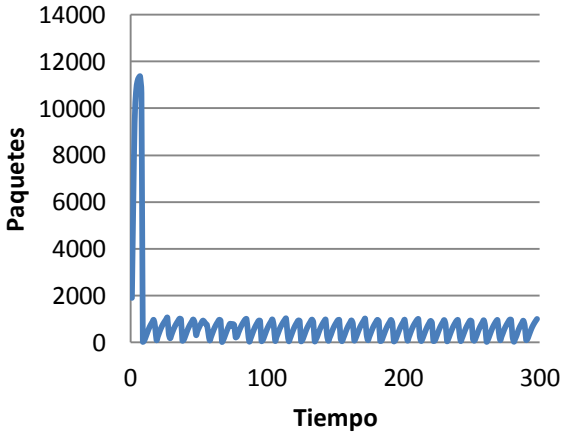
CWND en RENO en el tiempo con RTT=3[ms]



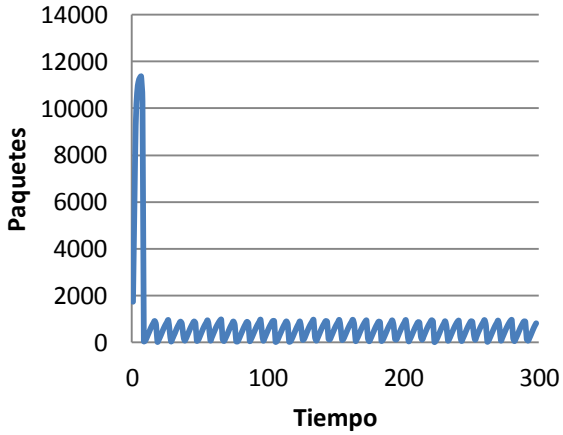
CWND en RENO en el tiempo con RTT=4[ms]



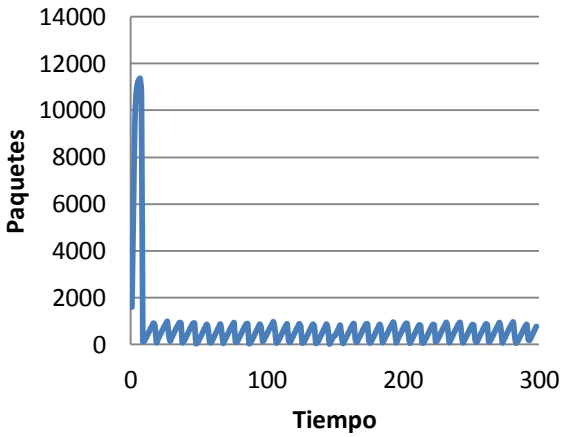
CWND en RENO en el tiempo con RTT=5[ms]



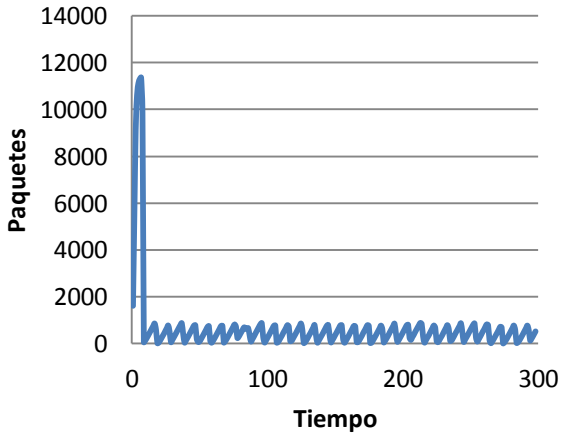
CWND en RENO en el tiempo con RTT=6[ms]



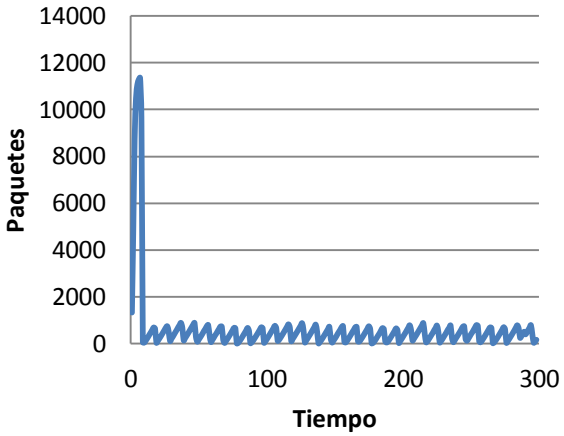
CWND en RENO en el tiempo con RTT=7[ms]



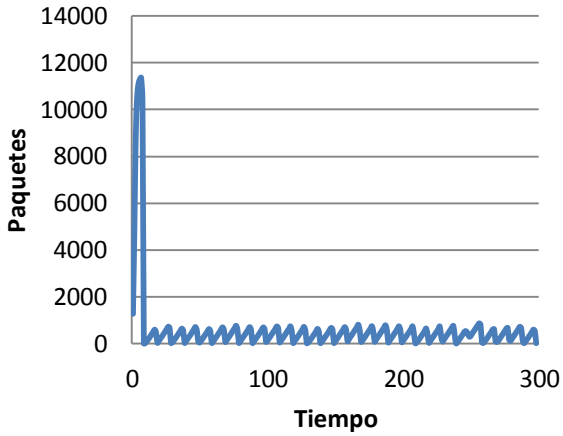
CWND en RENO en el tiempo con RTT=8[ms]



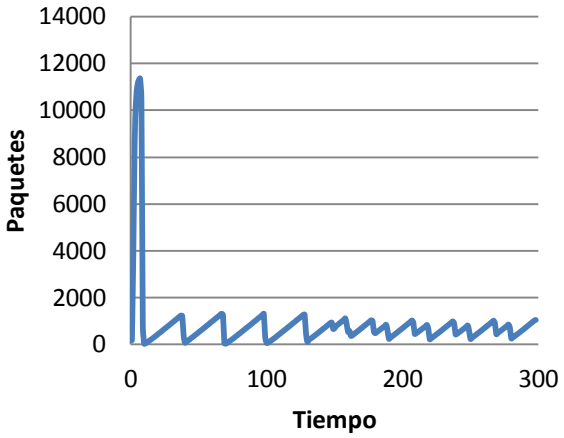
CWND en RENO en el tiempo con RTT=9[ms]



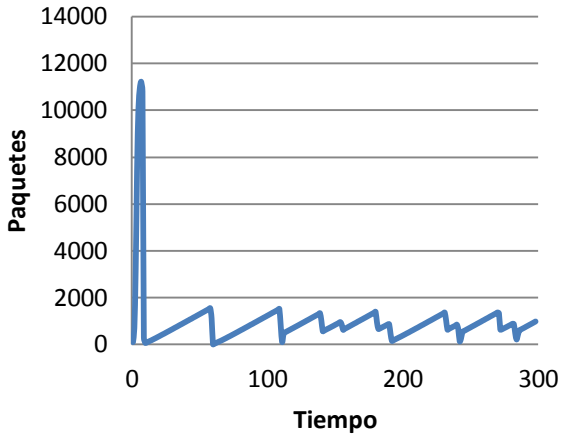
CWND en RENO en el tiempo con RTT=10[ms]



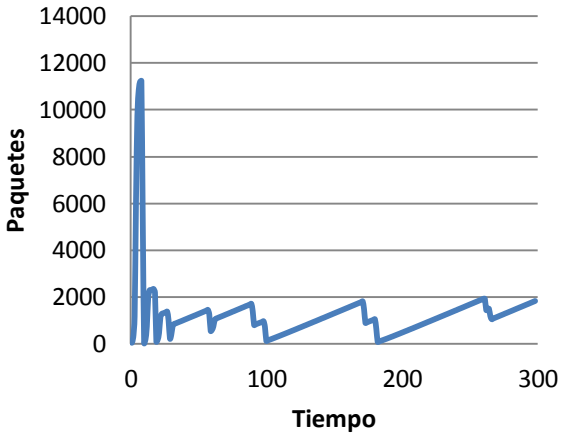
CWND en RENO en el tiempo con RTT=20[ms]



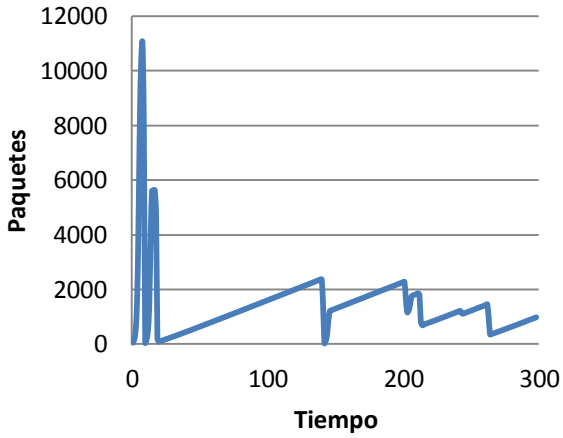
CWND en RENO en el tiempo con RTT=30[ms]



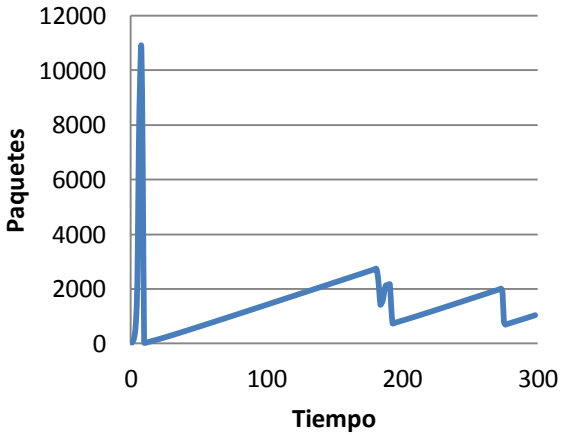
CWND en RENO en el tiempo con RTT=40[ms]



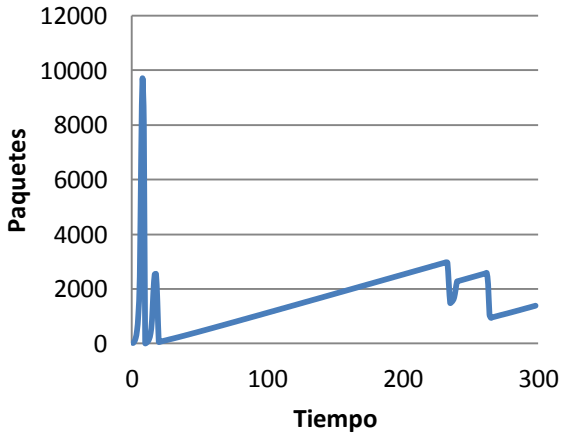
CWND en RENO en el tiempo con RTT=50[ms]



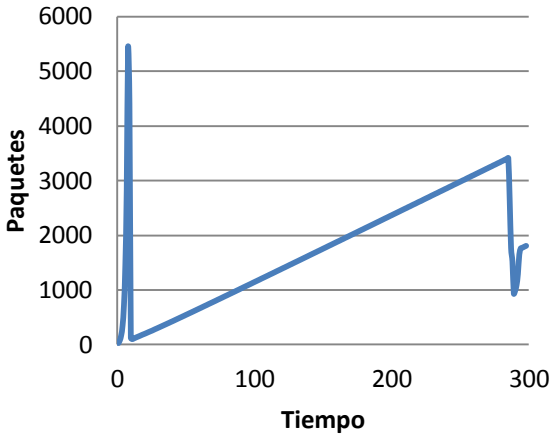
CWND en RENO en el tiempo con RTT=60[ms]



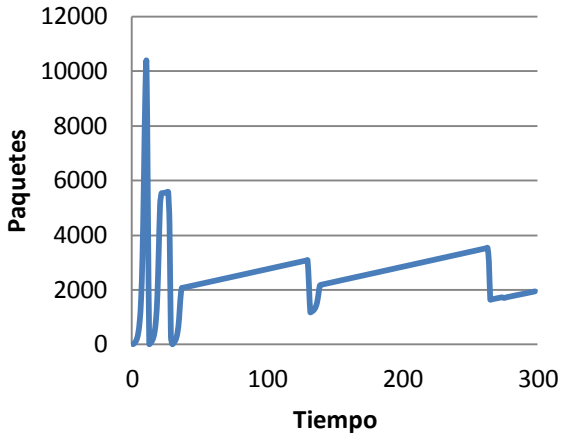
CWND en RENO en el tiempo con RTT=70[ms]



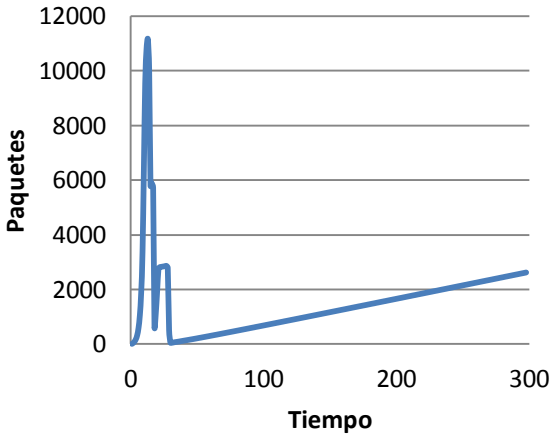
CWND en RENO en el tiempo con RTT=80[ms]



CWND en RENO en el tiempo con RTT=90[ms]



CWND en RENO en el tiempo con RTT=100[ms]

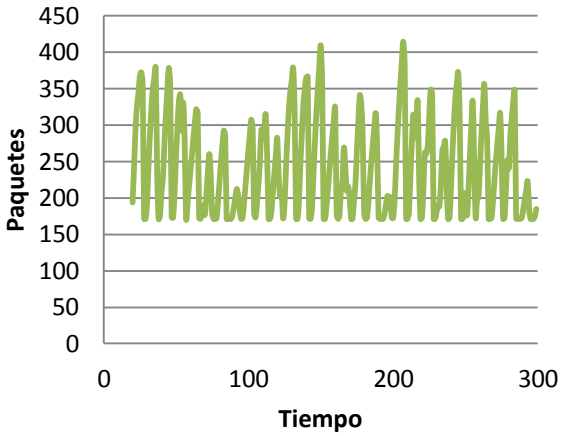


2. SSTHRESHOLD DE DATOS EN EL TIEMPO CON DIFERENTES RTT

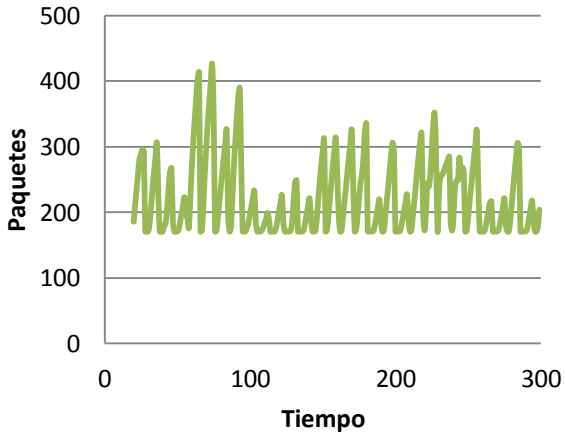
Para esta sección se omitieron los datos del 1 al 20 segundos, esto con el fin de poder graficar ya que los primeros valores son extremadamente grandes.

A. ESTP

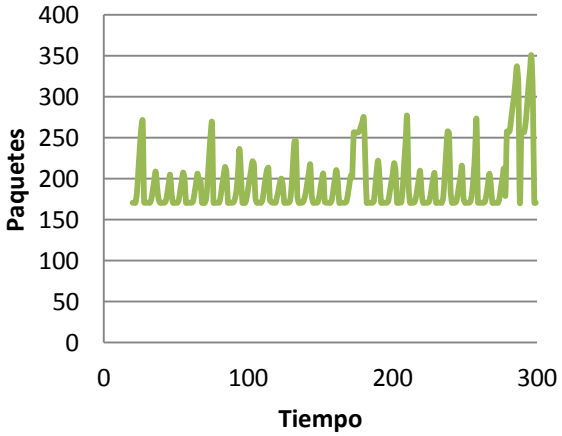
SSThreshold en ESTP en el tiempo con RTT=1[ms]



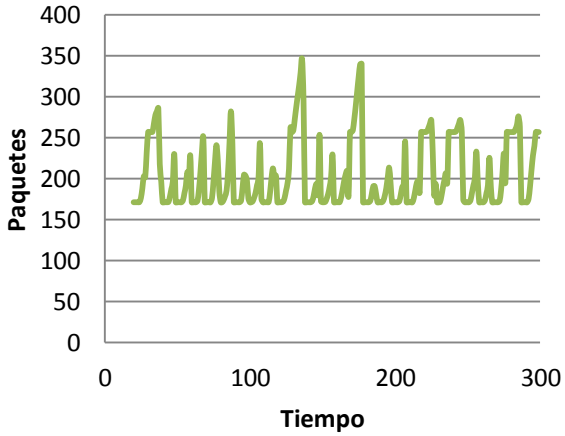
SSThreshold en ESTP en el tiempo con RTT=2[ms]



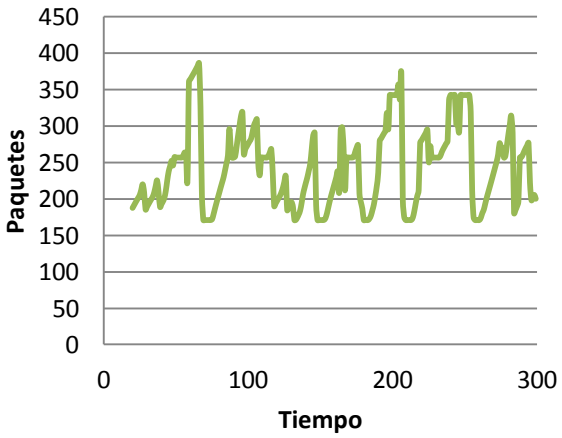
SSThreshold en ESTP en el tiempo con RTT=3[ms]



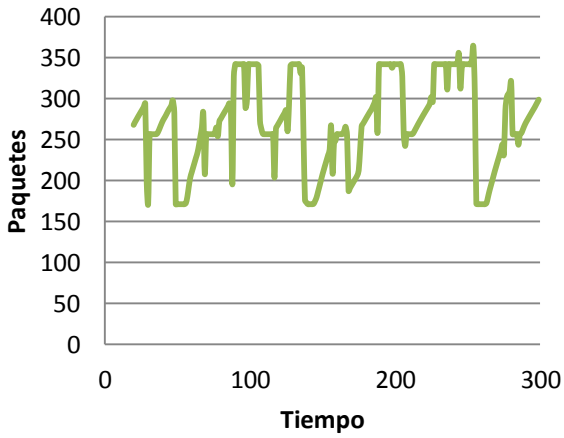
SSThreshold en ESTP en el tiempo con RTT=4[ms]



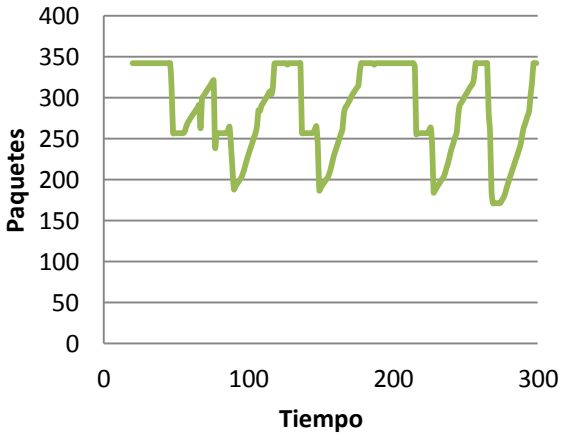
SSThreshold en ESTP en el tiempo con RTT=5[ms]



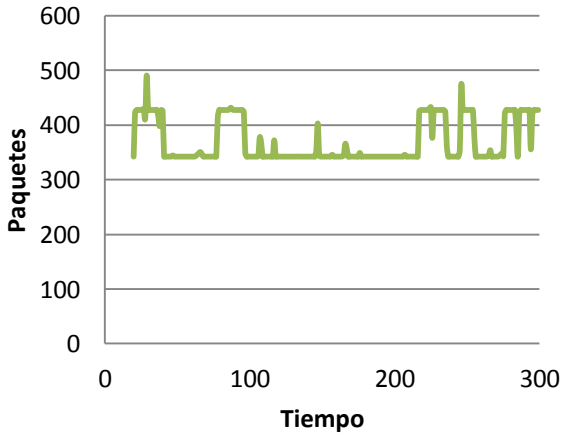
SSThreshold en ESTP en el tiempo con RTT=6[ms]



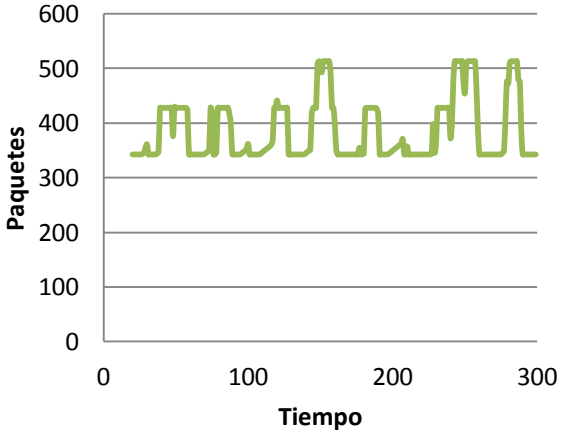
SSThreshold en ESTP en el tiempo con RTT=7[ms]



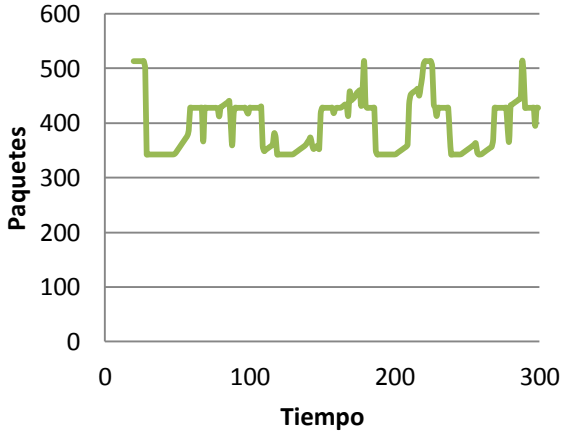
SSThreshold en ESTP en el tiempo con RTT=8[ms]



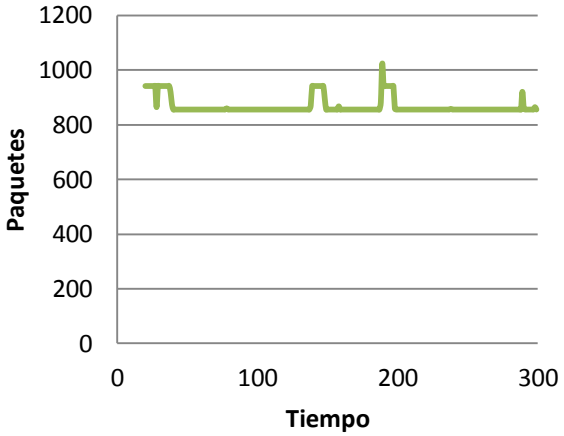
SSThreshold en ESTP en el tiempo con RTT=9[ms]



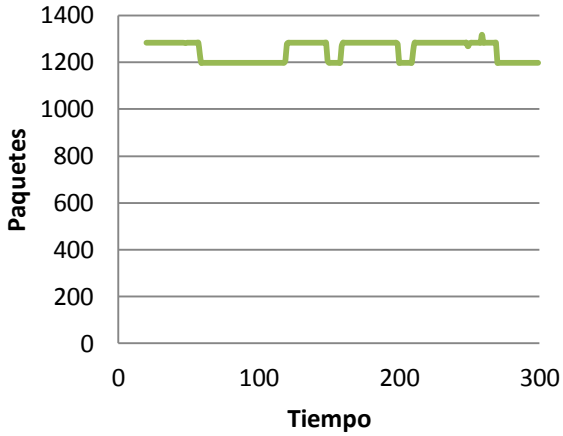
SSThreshold en ESTP en el tiempo con RTT=10[ms]



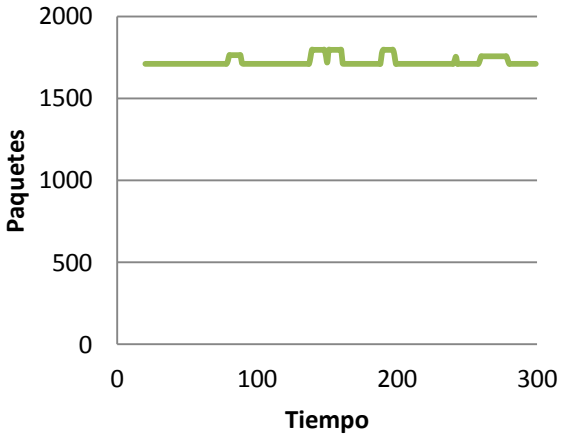
SSThreshold en ESTP en el tiempo con RTT=20[ms]



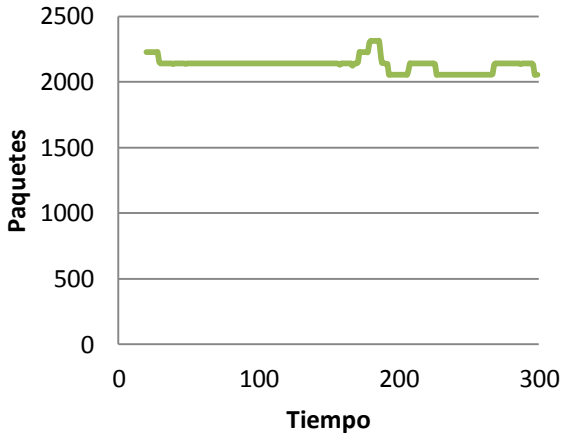
SSThreshold en ESTP en el tiempo con RTT=30[ms]



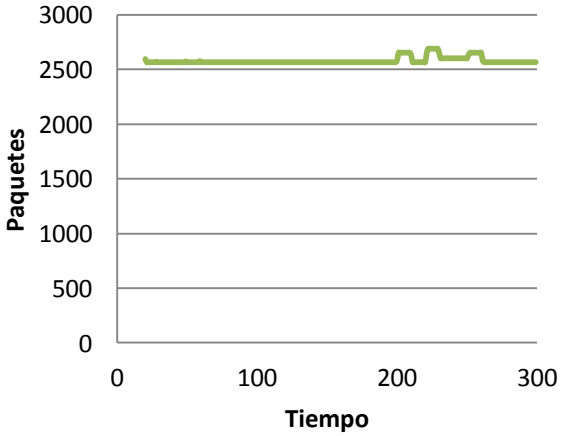
SSThreshold en ESTP en el tiempo con RTT=40[ms]



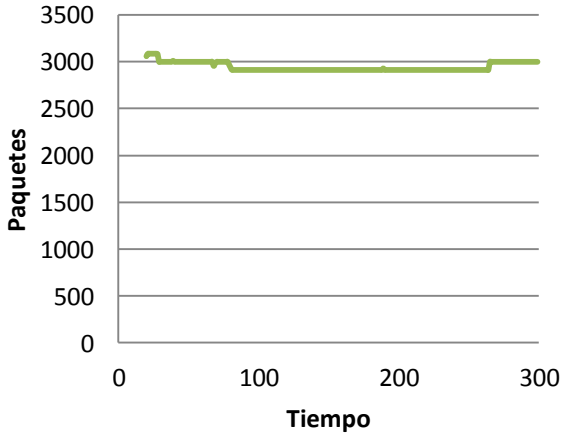
SSThreshold en ESTP en el tiempo con RTT=50[ms]



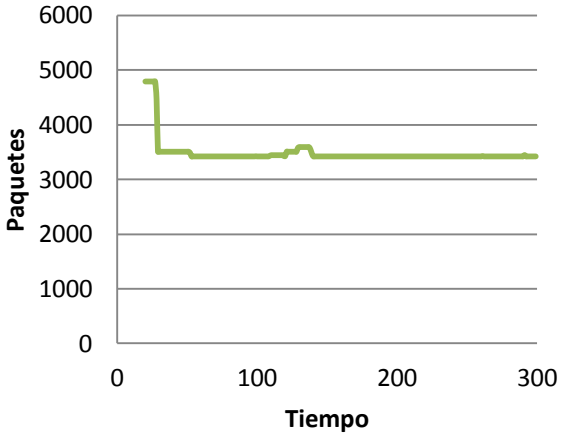
SSThreshold en ESTP en el tiempo con RTT=60[ms]



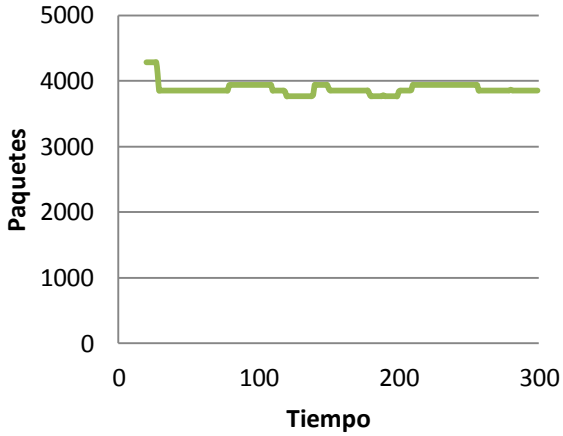
SSThreshold en ESTP en el tiempo con RTT=70[ms]



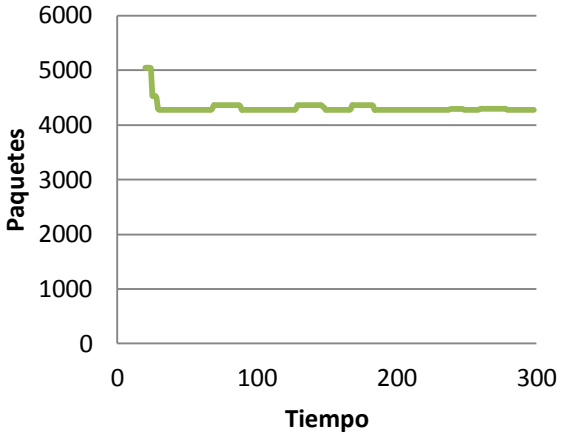
SSThreshold en ESTP en el tiempo con RTT=80[ms]



SSThreshold en ESTP en el tiempo con RTT=90[ms]

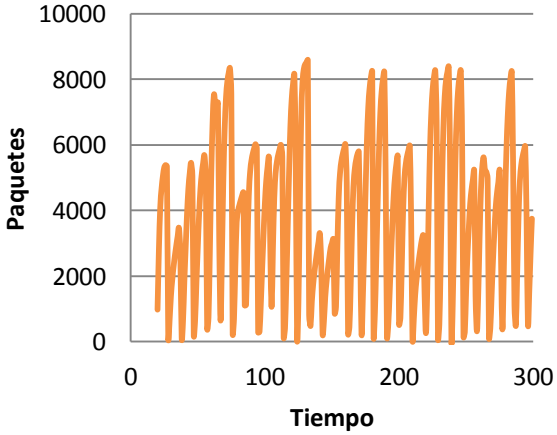


SSThreshold en ESTP en el tiempo con RTT=100[ms]

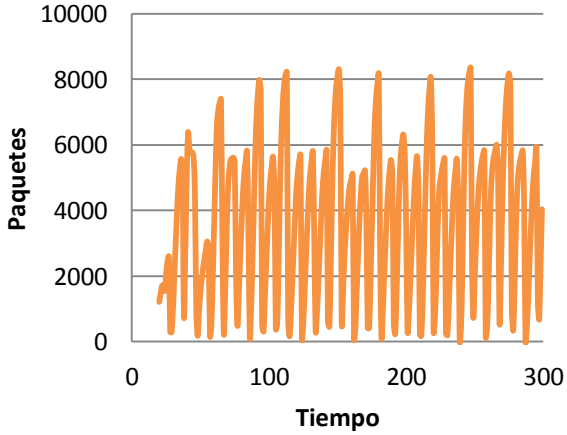


B. BIC

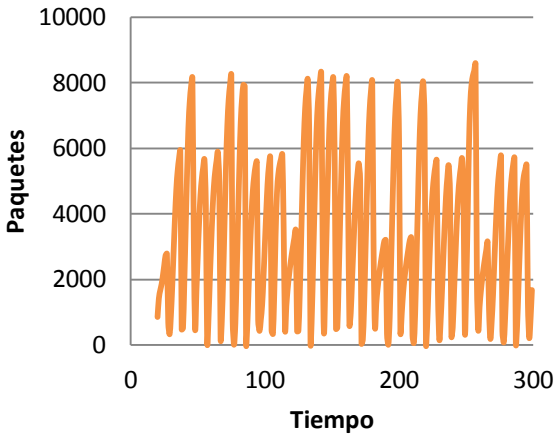
SSThreshold en BIC en el tiempo con RTT=1[ms]



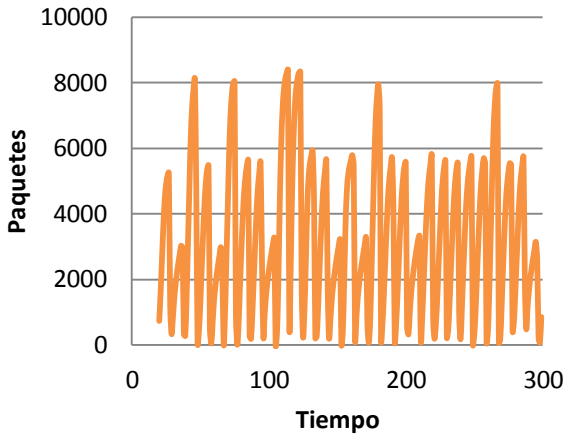
SSThreshold en BIC en el tiempo con RTT=2[ms]



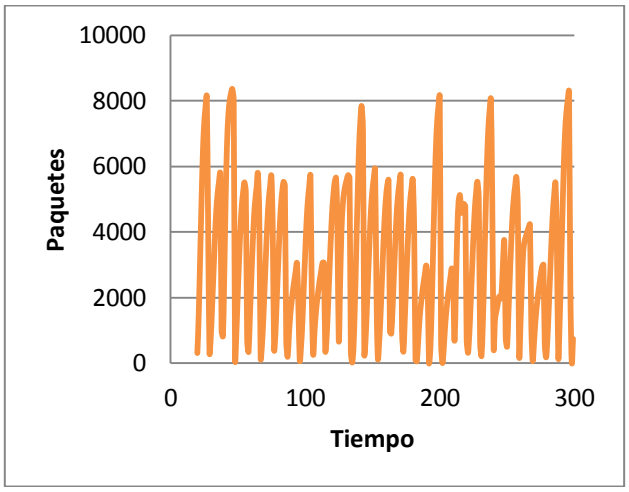
SSThreshold en BIC en el tiempo con RTT=3[ms]



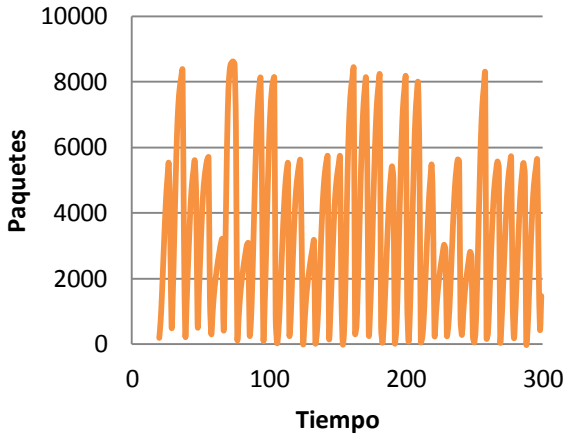
SSThreshold en BIC en el tiempo con RTT=4[ms]



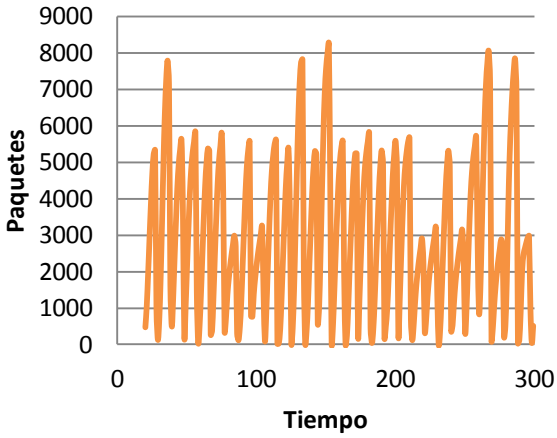
SSThreshold en BIC en el tiempo con RTT=5[ms]



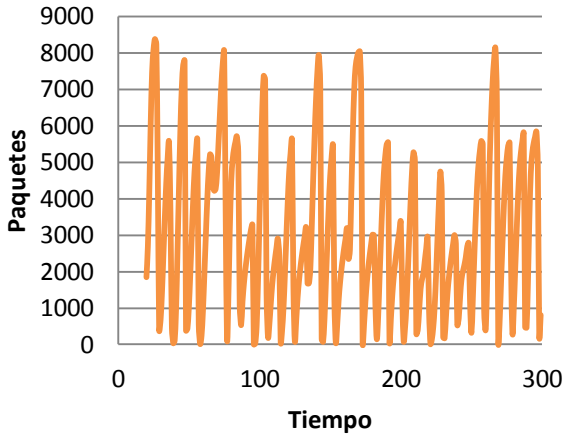
SSThreshold en BIC en el tiempo con RTT=6[ms]



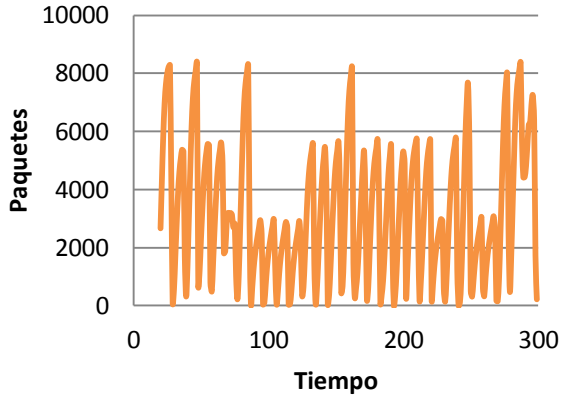
SSThreshold en BIC en el tiempo con RTT=7[ms]



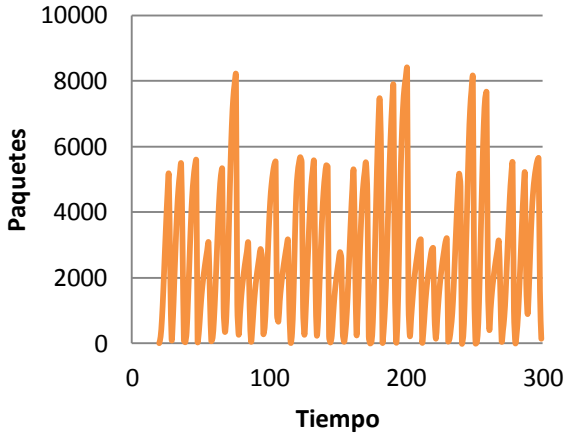
SSThreshold en BIC en el tiempo con RTT=8[ms]



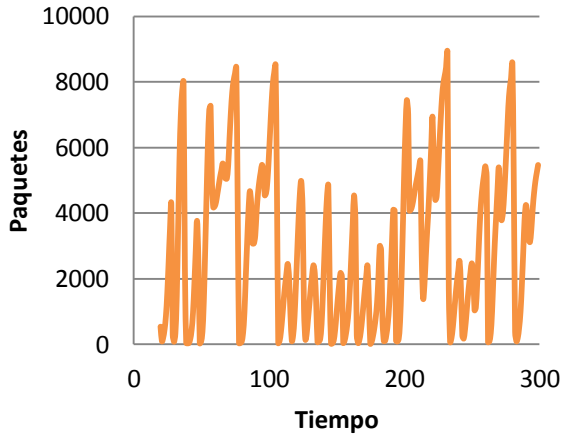
SSThreshold en BIC en el tiempo con RTT=9[ms]



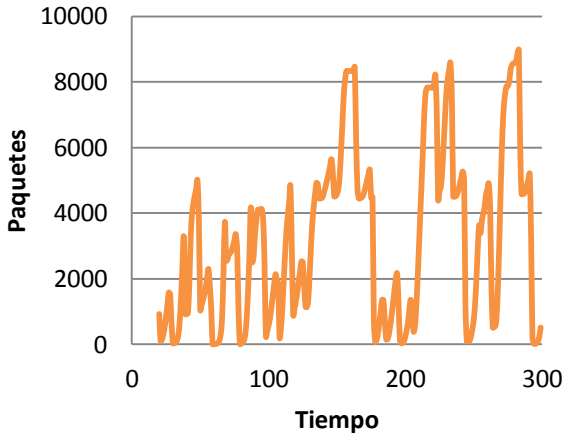
SSThreshold en BIC en el tiempo con RTT=10[ms]



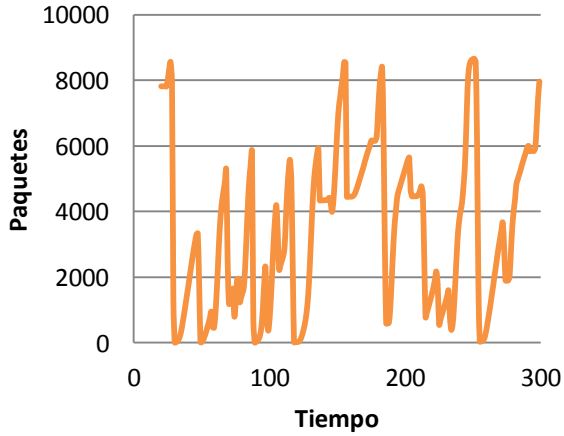
SSThreshold en BIC en el tiempo con RTT=20[ms]



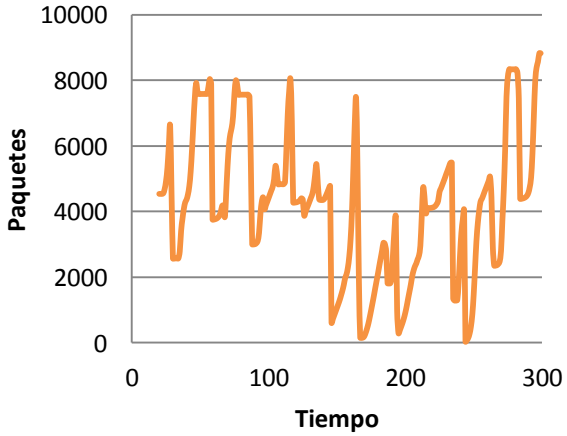
SSThreshold en BIC en el tiempo con RTT=30[ms]



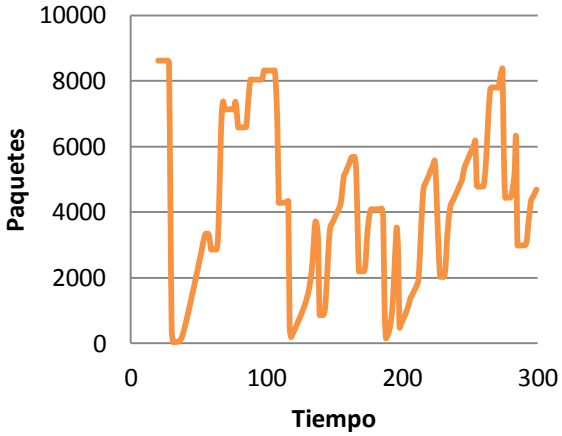
SSThreshold en BIC en el tiempo con RTT=40[ms]



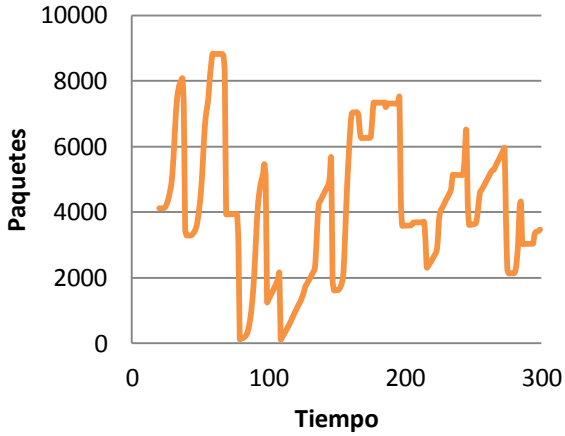
SSThreshold en BIC en el tiempo con RTT=50[ms]



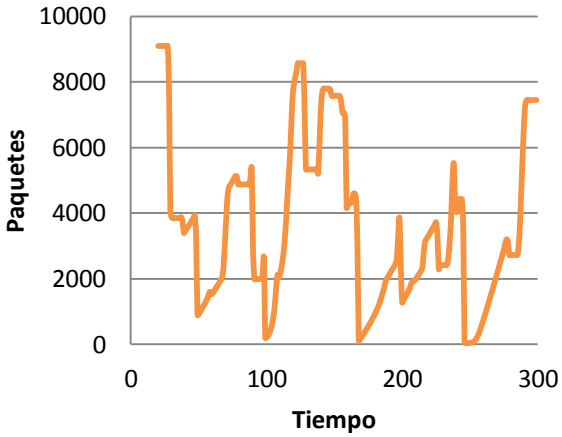
SSThreshold en BIC en el tiempo con RTT=60[ms]



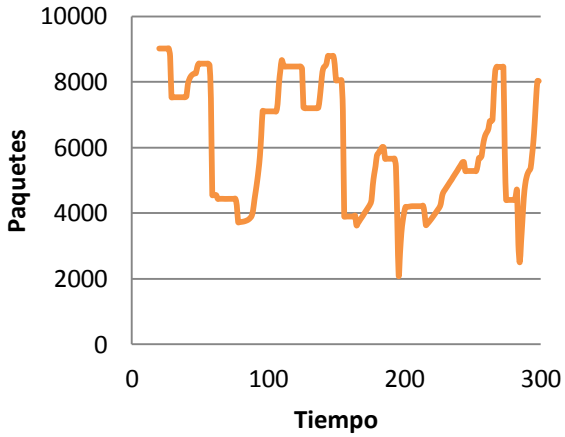
SSThreshold en BIC en el tiempo con RTT=70[ms]



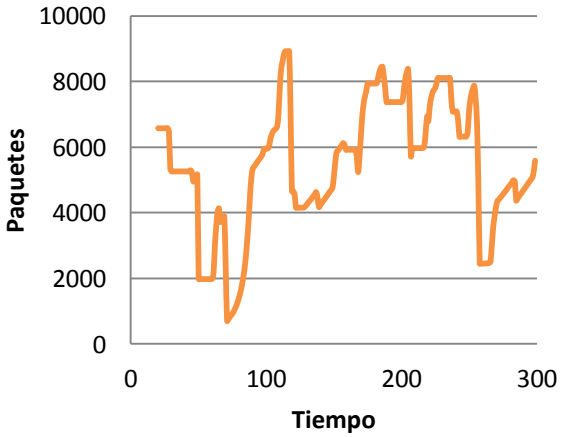
SSThreshold en BIC en el tiempo con RTT=80[ms]



SSThreshold en BIC en el tiempo con RTT=90[ms]

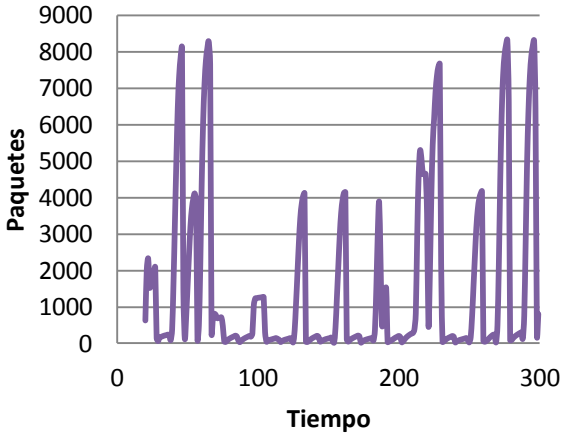


SSThreshold en BIC en el tiempo con RTT=100[ms]

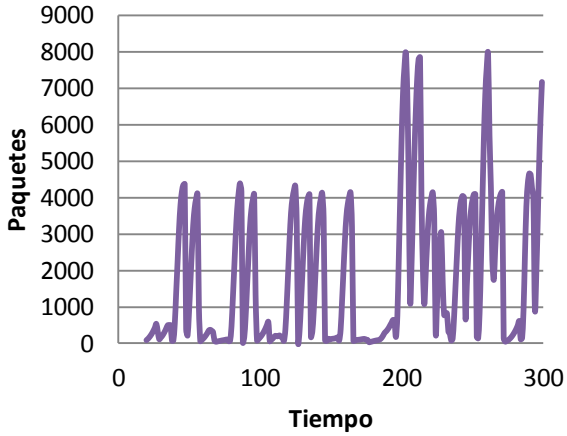


C. CUBIC

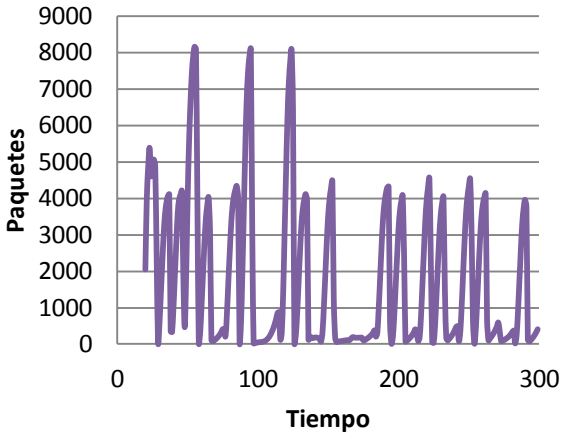
SSThreshold en CUBIC en el tiempo con RTT=1[ms]



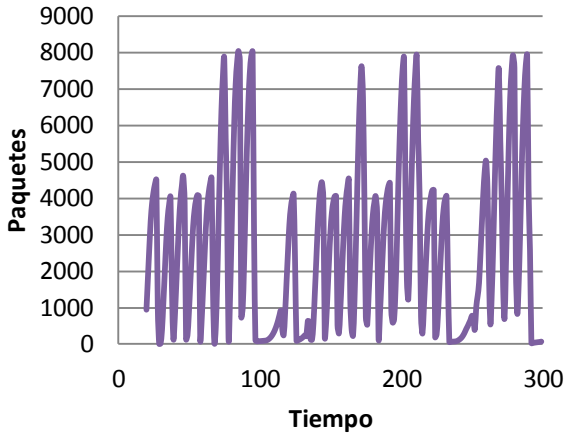
SSThreshold en CUBIC en el tiempo con RTT=2[ms]



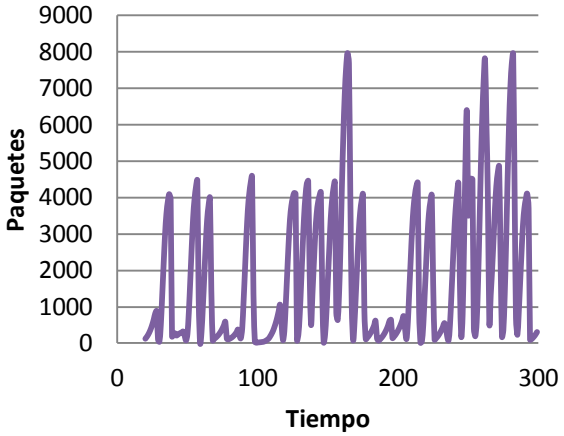
SSThreshold en CUBIC en el tiempo con RTT=3[ms]



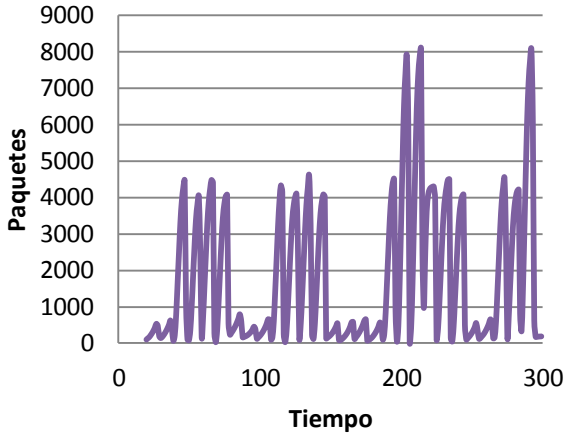
SSThreshold en CUBIC en el tiempo con RTT=4[ms]



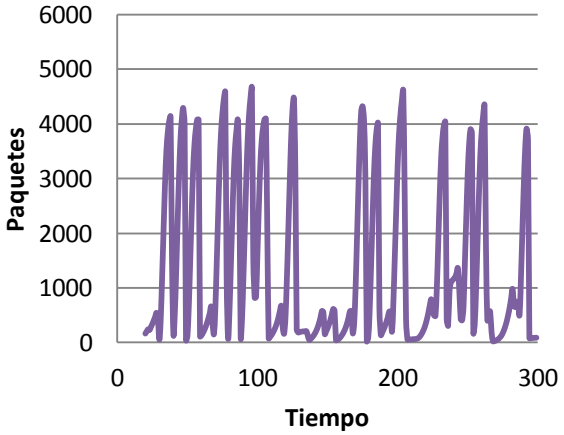
SSThreshold en CUBIC en el tiempo con RTT=5[ms]



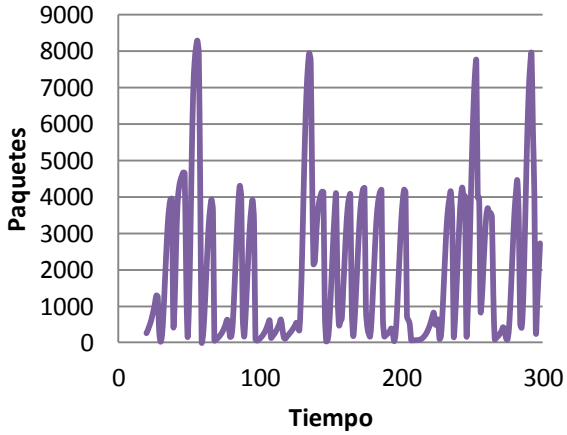
SSThreshold en CUBIC en el tiempo con RTT=6[ms]



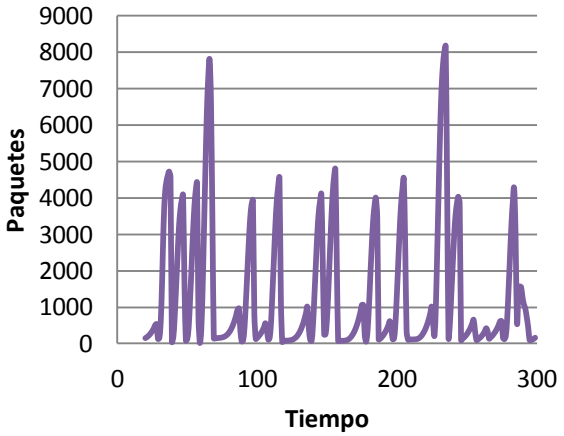
SSThreshold en CUBIC en el tiempo con RTT=7[ms]



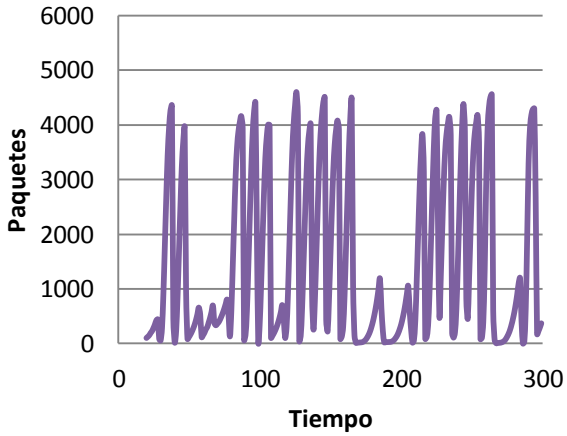
SSThreshold en CUBIC en el tiempo con RTT=8[ms]



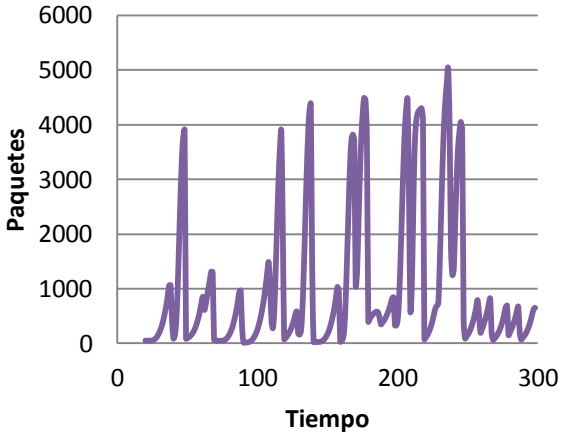
SSThreshold en CUBIC en el tiempo con RTT=9[ms]



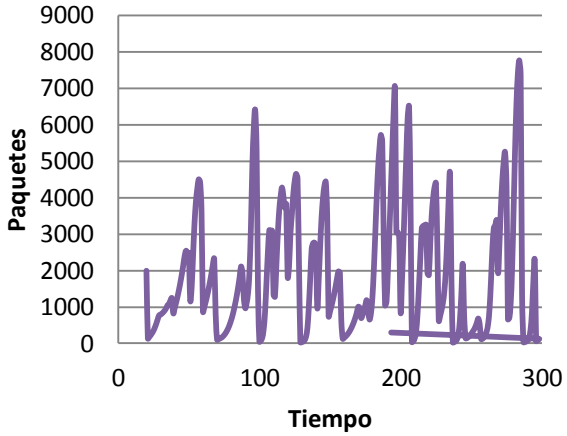
SSThreshold en CUBIC en el tiempo con RTT=10[ms]



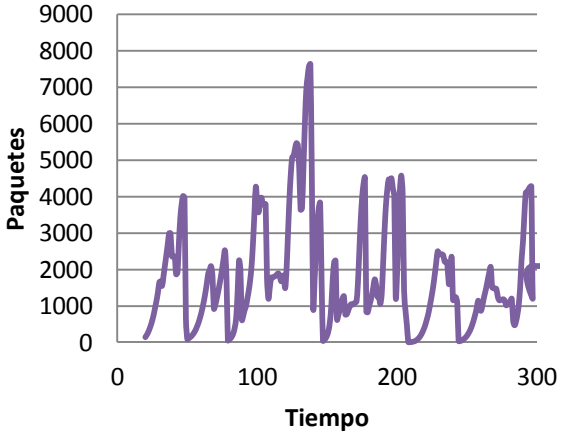
SSThreshold en CUBIC en el tiempo con RTT=20[ms]



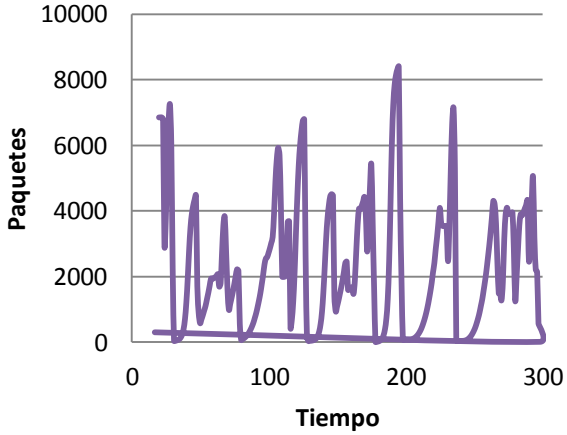
SSThreshold en CUBIC en el tiempo con RTT=30[ms]



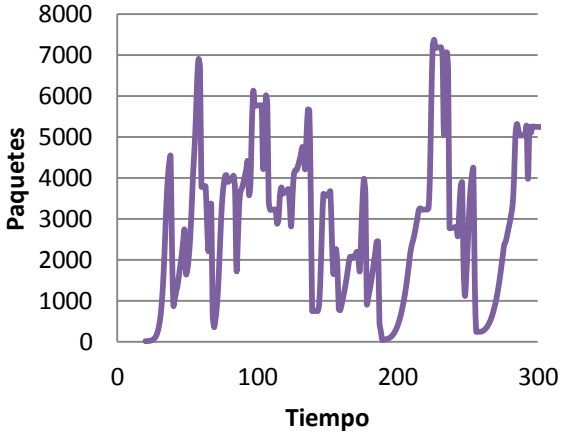
SSThreshold en CUBIC en el tiempo con RTT=40[ms]



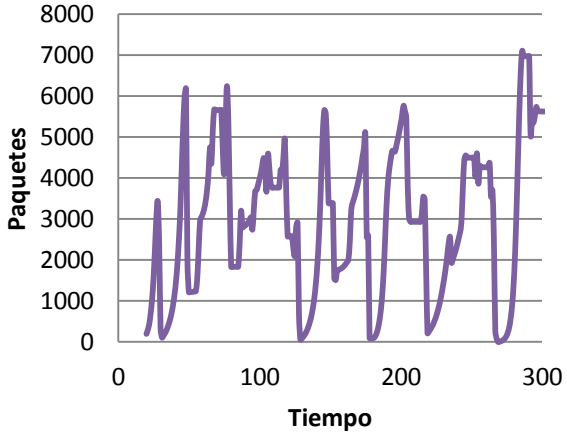
SSThreshold en CUBIC en el tiempo con RTT=50[ms]



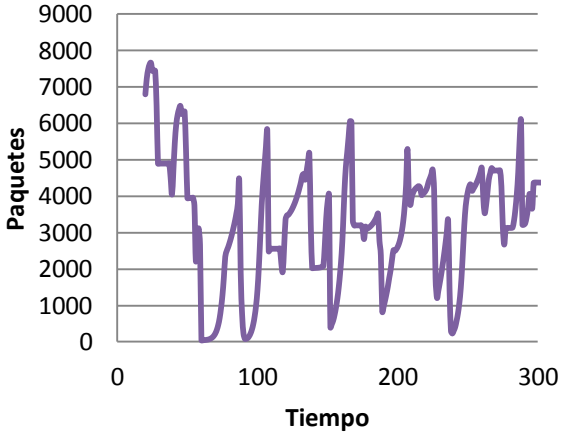
SSThreshold en CUBIC en el tiempo con RTT=60[ms]



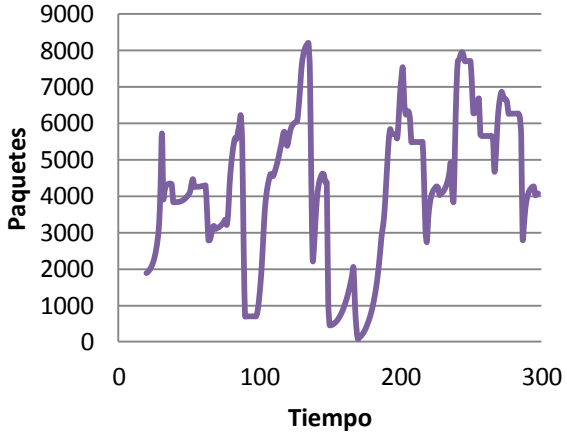
SSThreshold en CUBIC en el tiempo con RTT=70[ms]



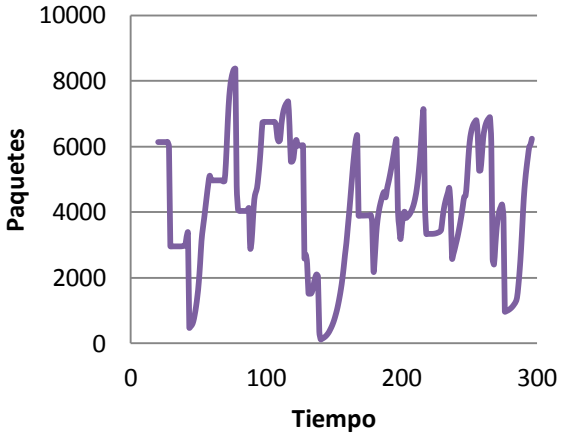
SSThreshold en CUBIC en el tiempo con RTT=80[ms]



SSThreshold en CUBIC en el tiempo con RTT=90[ms]

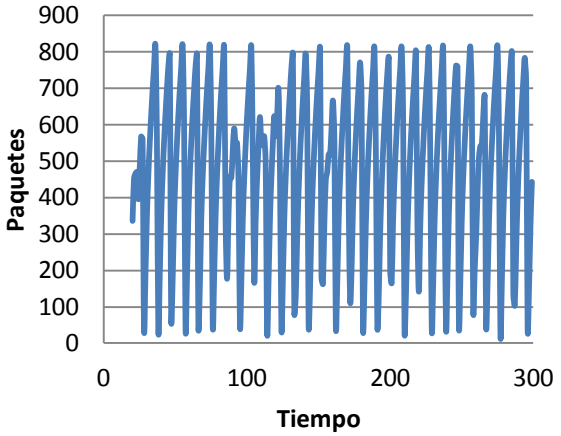


SSThreshold en CUBIC en el tiempo con RTT=100[ms]

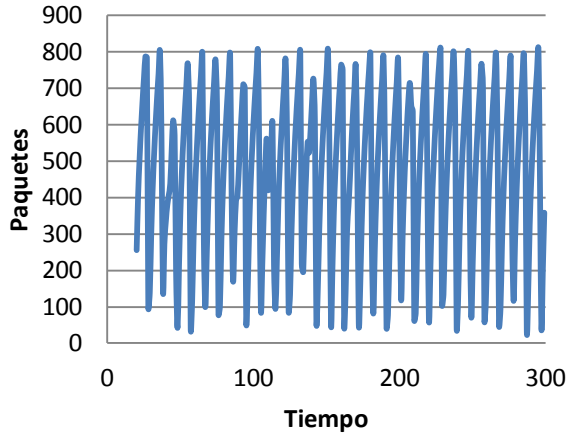


D. RENO

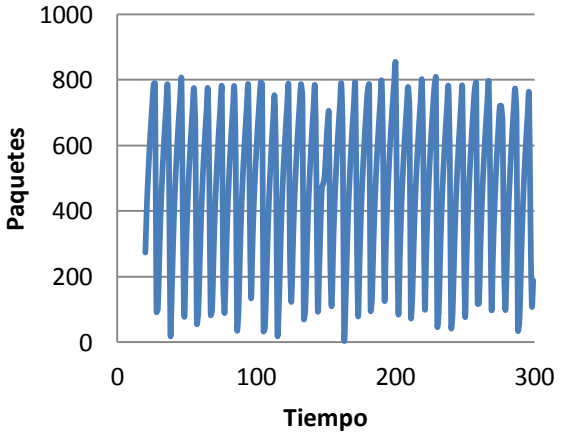
SSThreshold en RENO en el tiempo con RTT=1[ms]



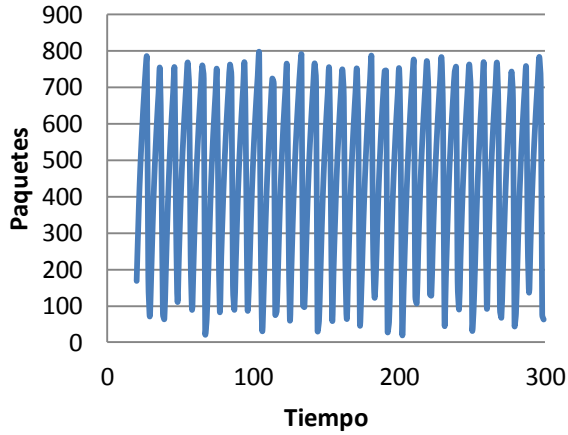
SSThreshold en RENO en el tiempo con RTT=2[ms]



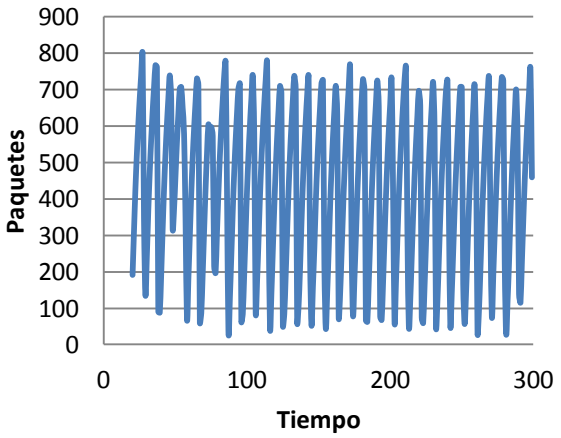
SSThreshold en RENO en el tiempo con RTT=3[ms]



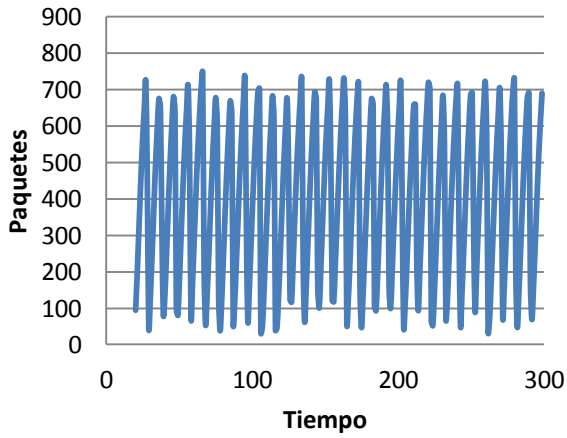
SSThreshold en RENO en el tiempo con RTT=4[ms]



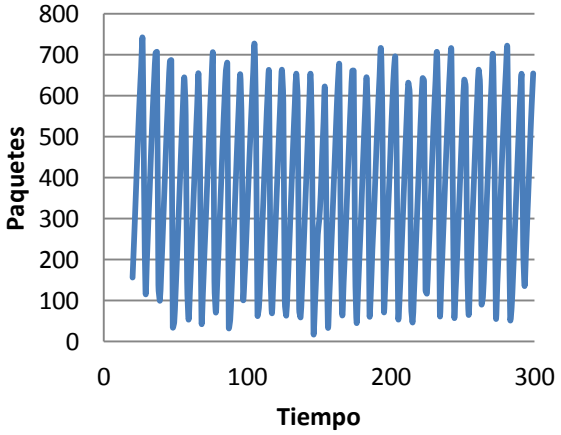
SSThreshold en RENO en el tiempo con RTT=5[ms]



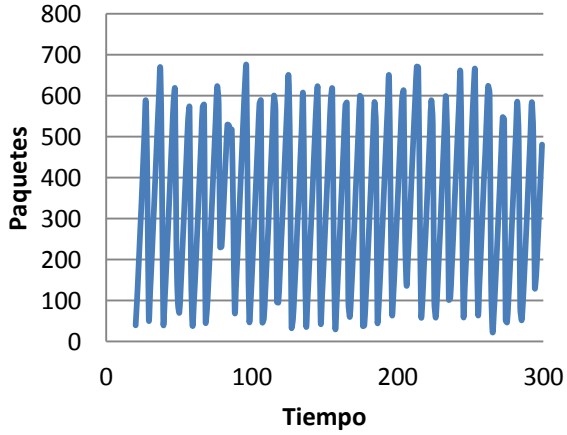
SSThreshold en RENO en el tiempo con RTT=6[ms]



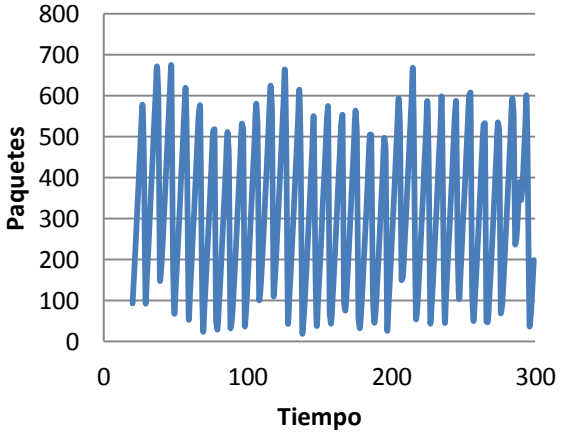
SSThreshold en RENO en el tiempo con RTT=7[ms]



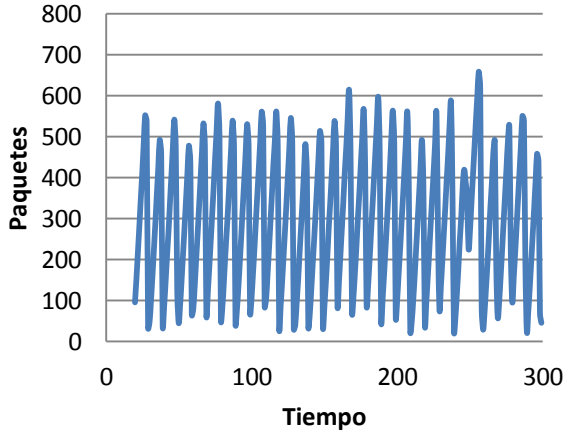
SSThreshold en RENO en el tiempo con RTT=8[ms]



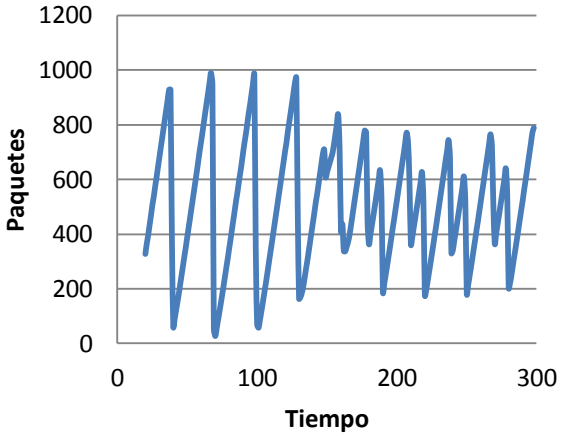
SSThreshold en RENO en el tiempo con RTT=9[ms]



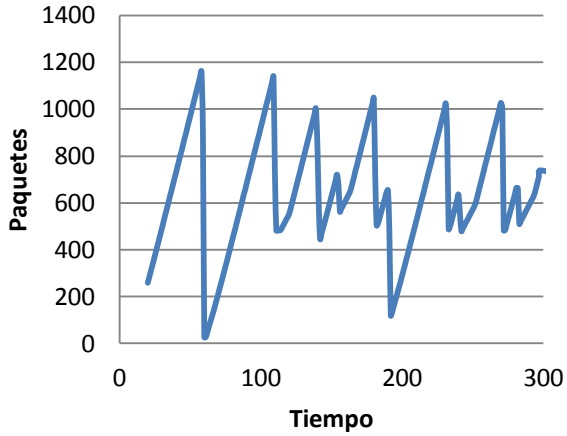
SSThreshold en RENO en el tiempo con RTT=10[ms]



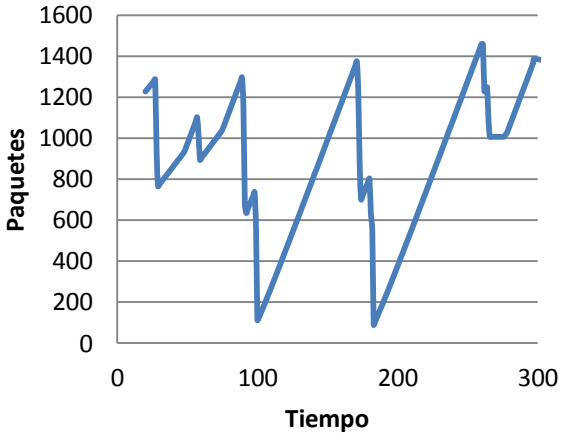
SSThreshold en RENO en el tiempo con RTT=20[ms]



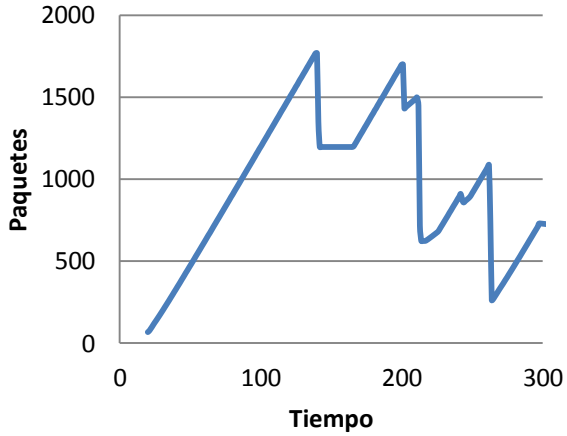
SSThreshold en RENO en el tiempo con RTT=30[ms]



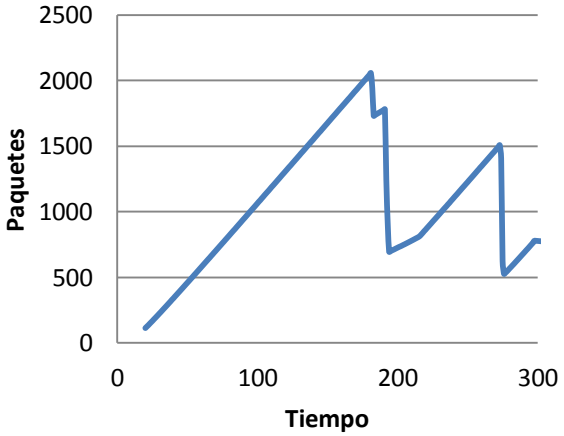
SSThreshold en RENO en el tiempo con RTT=40[ms]



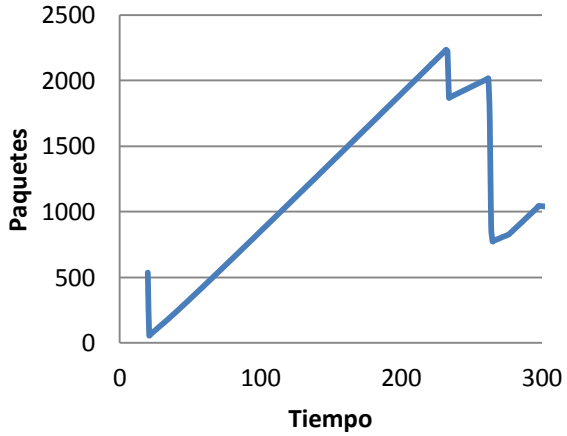
SSThreshold en RENO en el tiempo con RTT=50[ms]



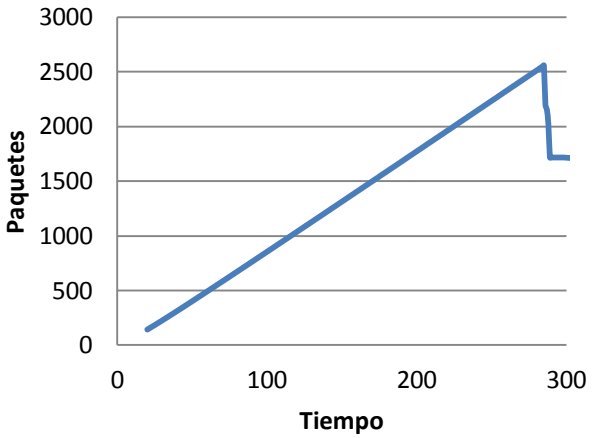
SSThreshold en RENO en el tiempo con RTT=60[ms]



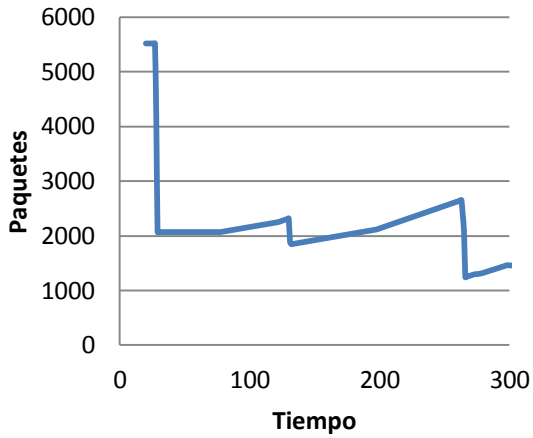
SSThreshold en RENO en el tiempo con RTT=70[ms]



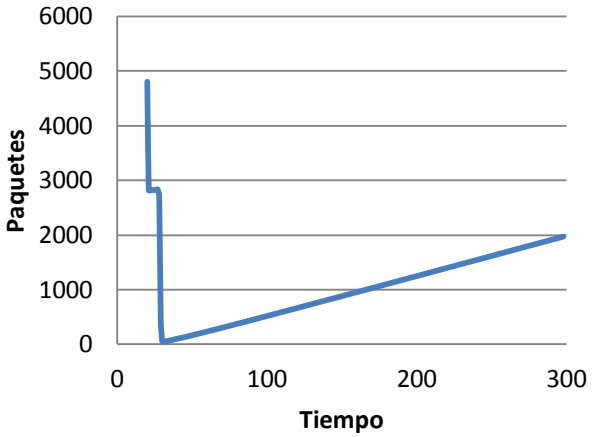
SSThreshold en RENO en el tiempo con RTT=80[ms]



SSThreshold en RENO en el tiempo con RTT=90[ms]



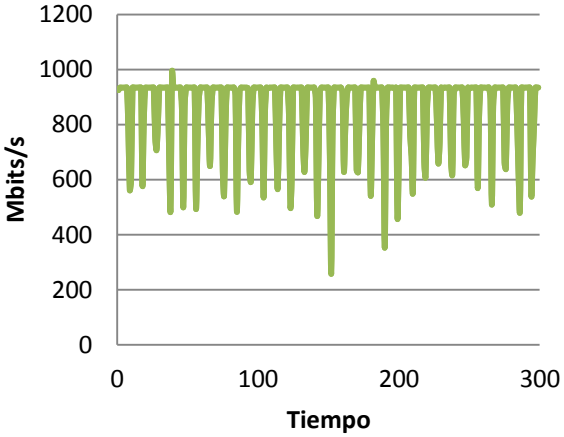
SSThreshold en RENO en el tiempo con RTT=100[ms]



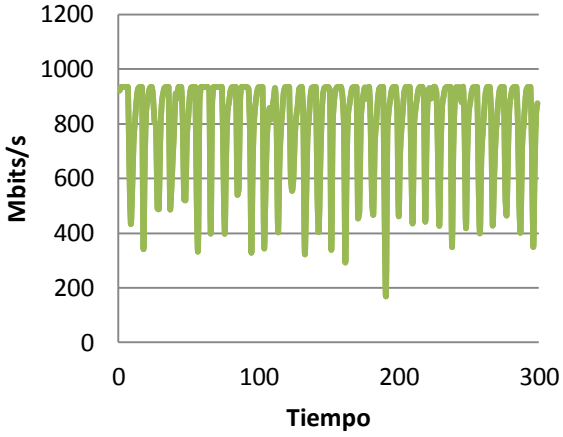
3. FLUJO DE DATOS EN EL TIEMPO CON DIFERENTES RTT

A. ESTP

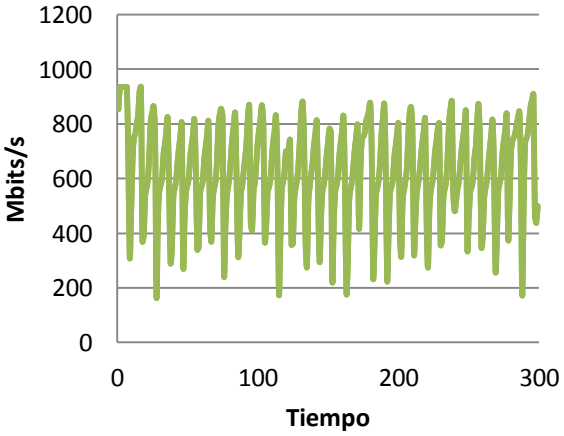
Flujo de Datos en ESTP en el tiempo con RTT=1[ms]



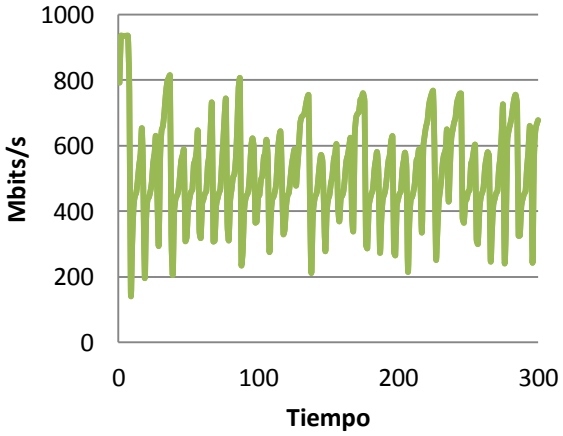
Flujo de Datos en ESTP en el tiempo con RTT=2[ms]



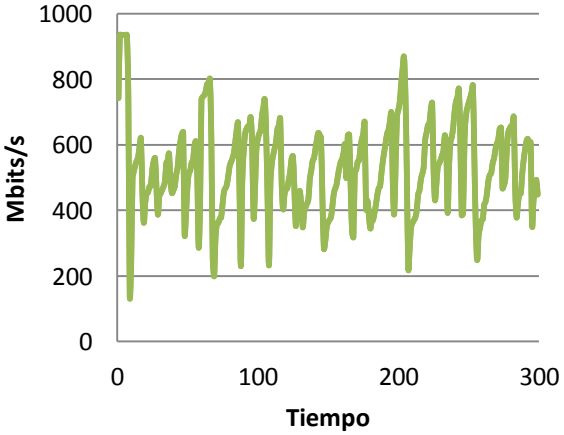
Flujo de Datos en ESTP en el tiempo con RTT=3[ms]



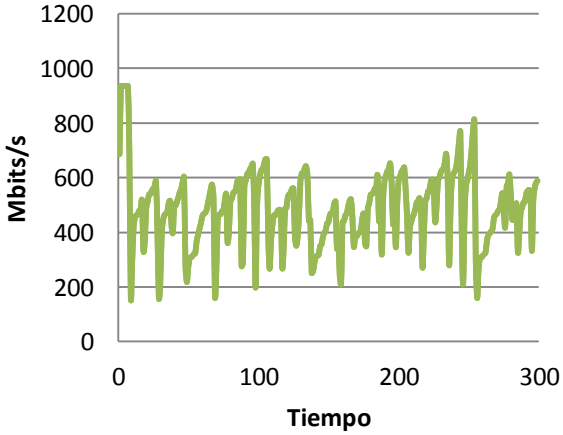
Flujo de Datos en ESTP en el tiempo con RTT=4[ms]



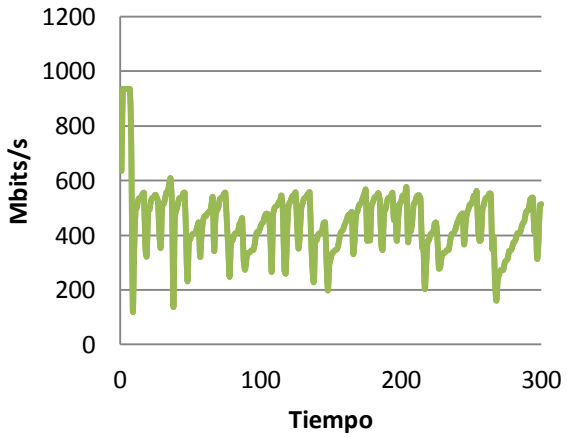
Flujo de Datos en ESTP en el tiempo con RTT=5[ms]



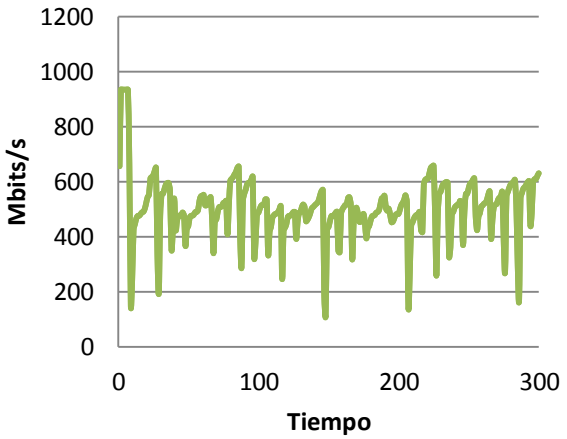
Flujo de Datos en ESTP en el tiempo con RTT=6[ms]



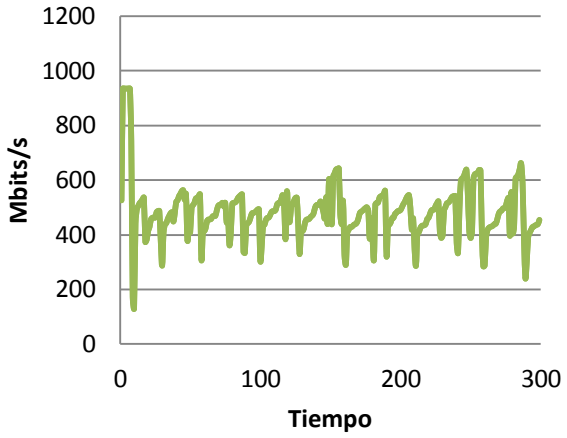
Flujo de Datos en ESTP en el tiempo con RTT=7[ms]



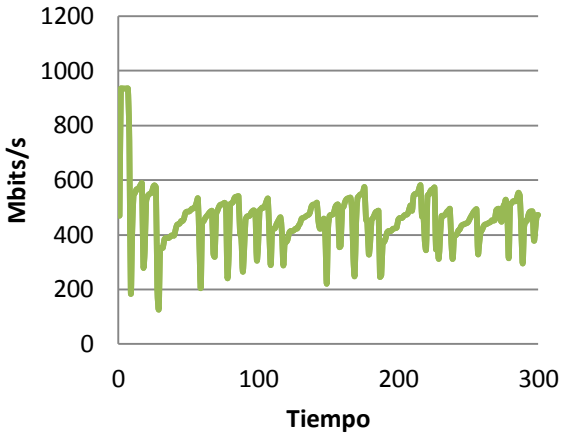
Flujo de Datos en ESTP en el tiempo con RTT=8[ms]



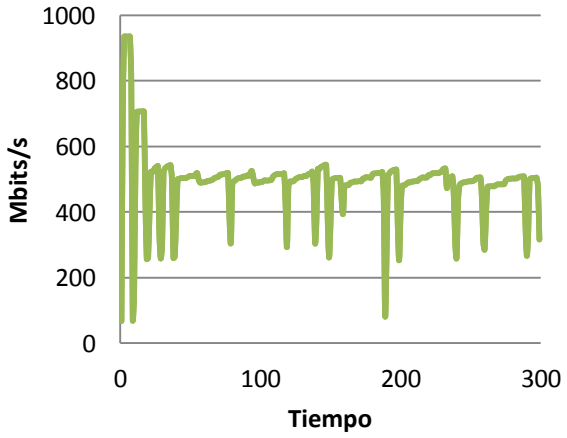
Flujo de Datos en ESTP en el tiempo con RTT=9[ms]



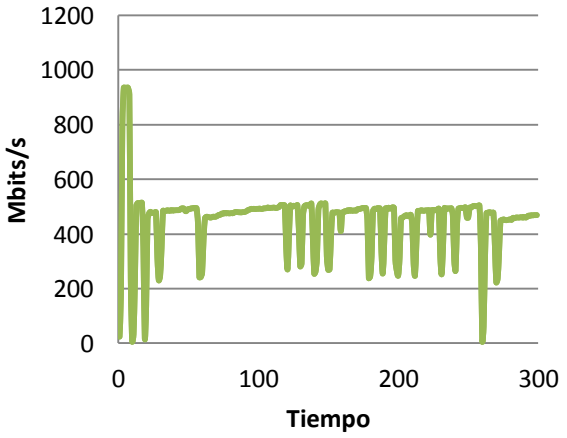
Flujo de Datos en ESTP en el tiempo con RTT=10[ms]



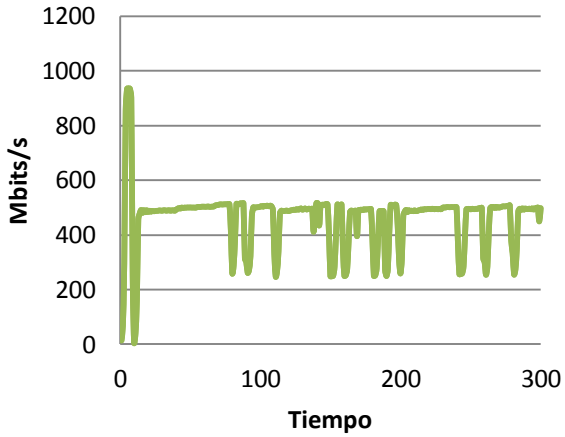
Flujo de Datos en ESTP en el tiempo con RTT=20[ms]



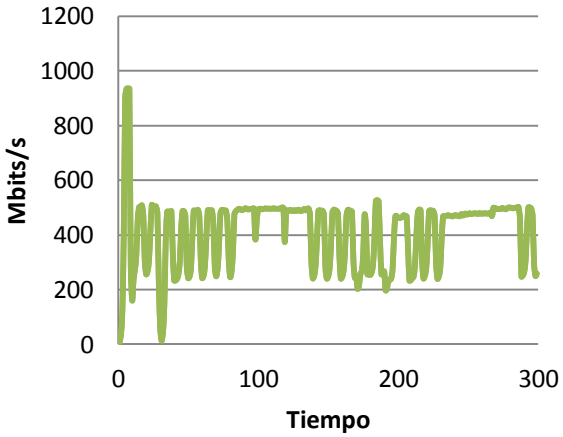
Flujo de Datos en ESTP en el tiempo con RTT=30[ms]



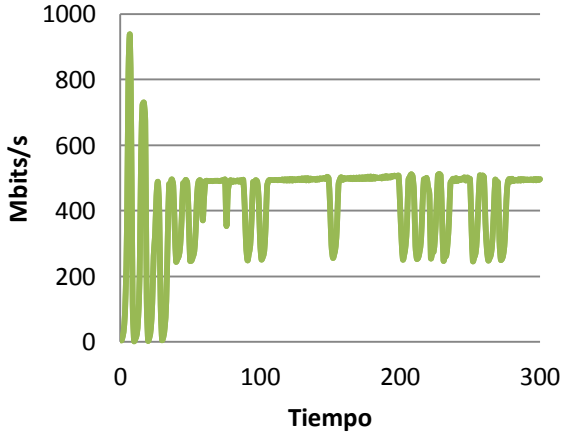
Flujo de Datos en ESTP en el tiempo con RTT=40[ms]



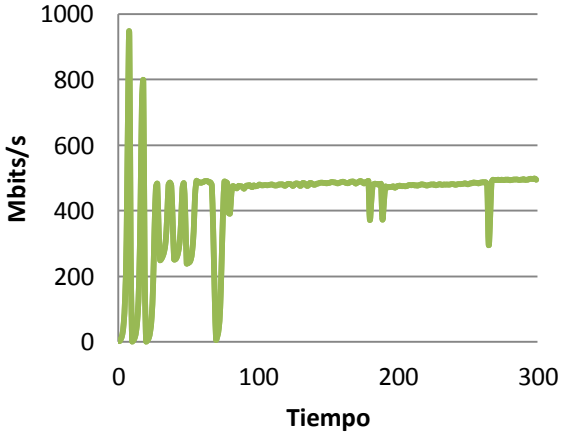
Flujo de Datos en ESTP en el tiempo con RTT=50[ms]



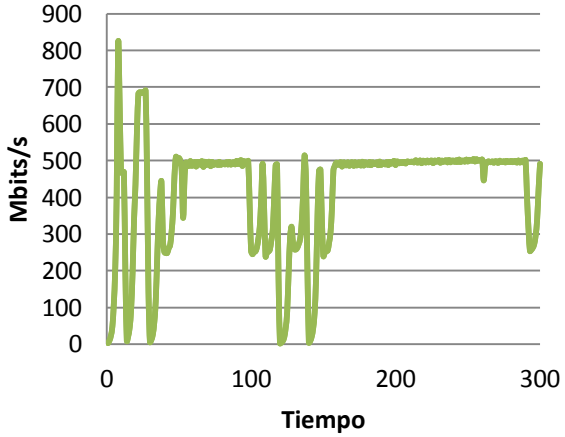
Flujo de Datos en ESTP en el tiempo con RTT=60[ms]



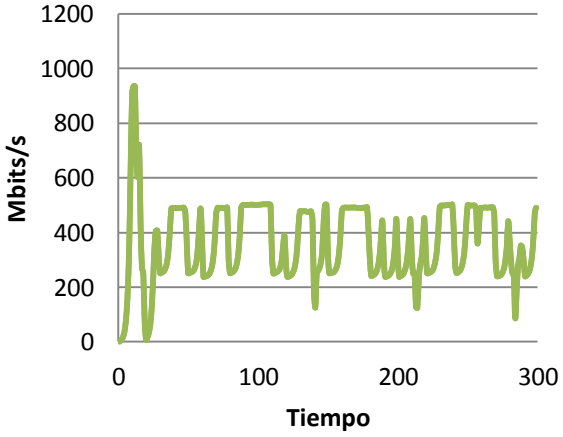
Flujo de Datos en ESTP en el tiempo con RTT=70[ms]



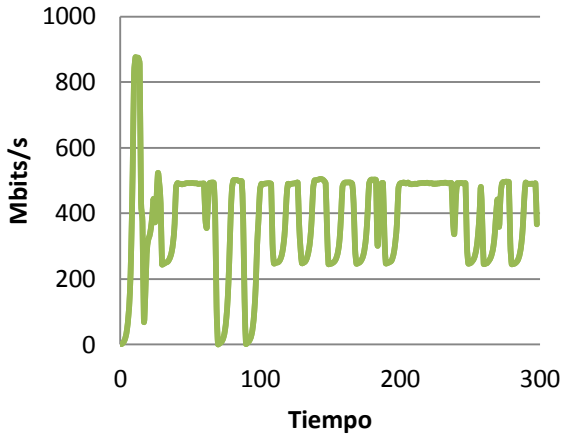
Flujo de Datos en ESTP en el tiempo con RTT=80[ms]



Flujo de Datos en ESTP en el tiempo con RTT=90[ms]

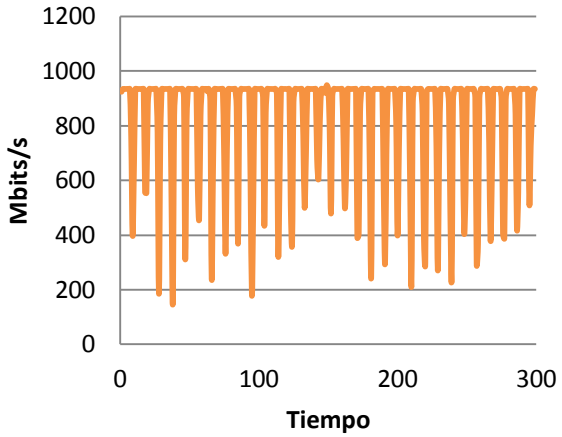


Flujo de Datos en ESTP en el tiempo con RTT=100[ms]

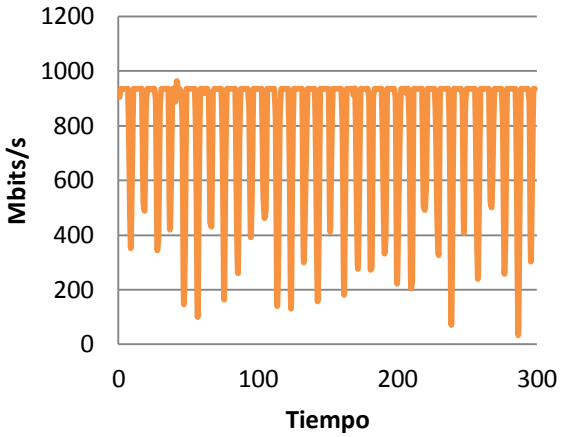


B. BIC

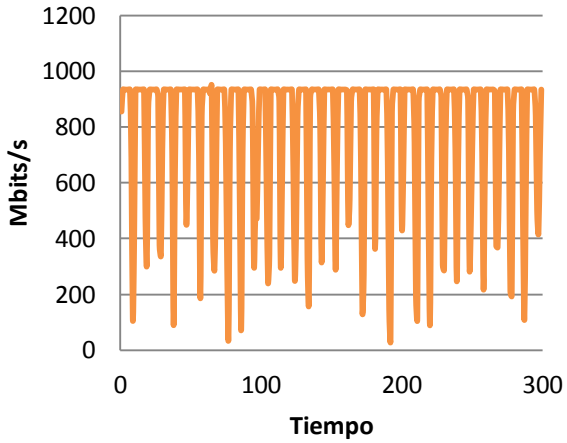
Flujo de Datos en BIC en el tiempo con RTT=1[ms]



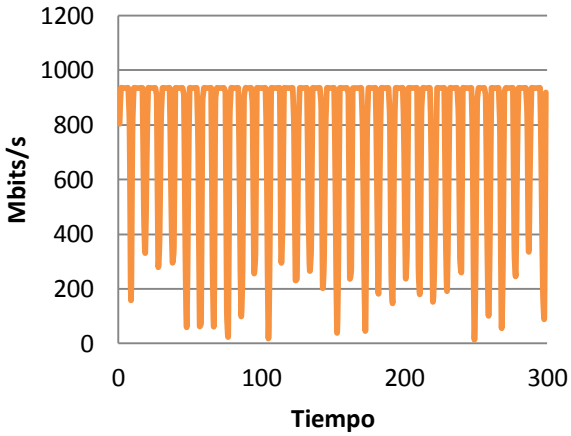
Flujo de Datos en BIC en el tiempo con RTT=2[ms]



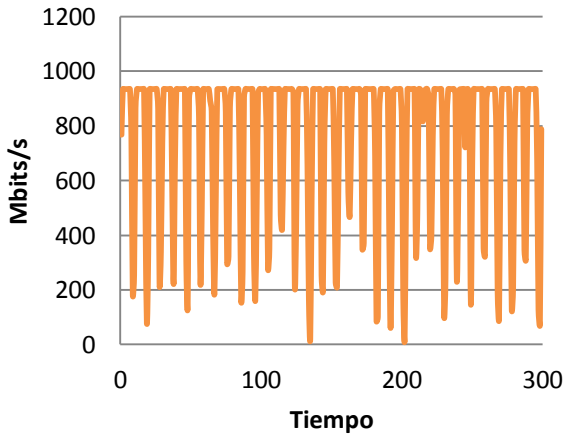
Flujo de Datos en BIC en el tiempo con RTT=3[ms]



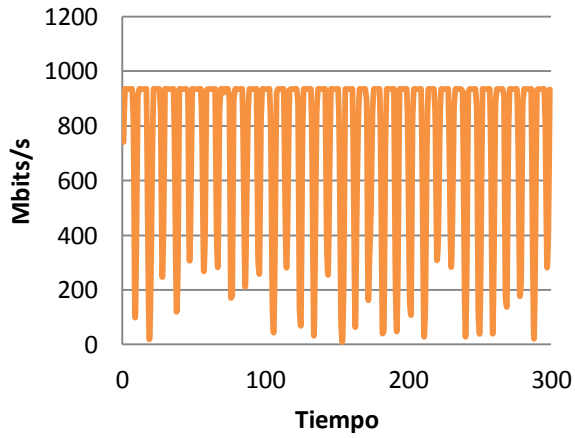
Flujo de Datos en BIC en el tiempo con RTT=4[ms]



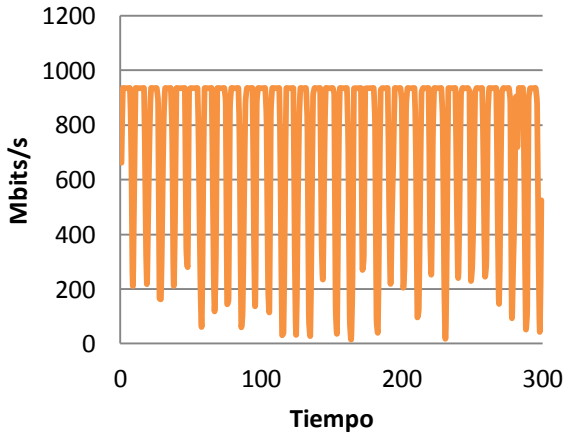
Flujo de Datos en BIC en el tiempo con RTT=5[ms]



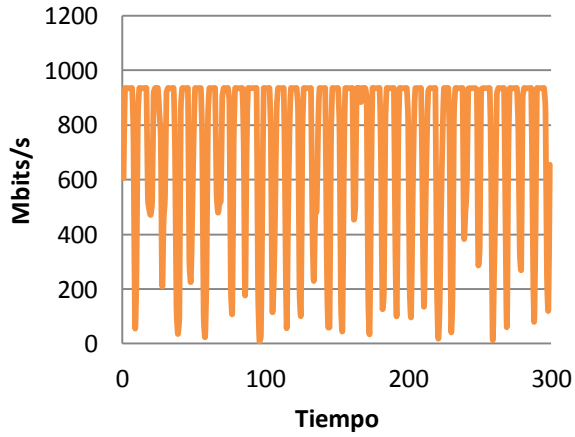
Flujo de Datos en BIC en el tiempo con RTT=6[ms]



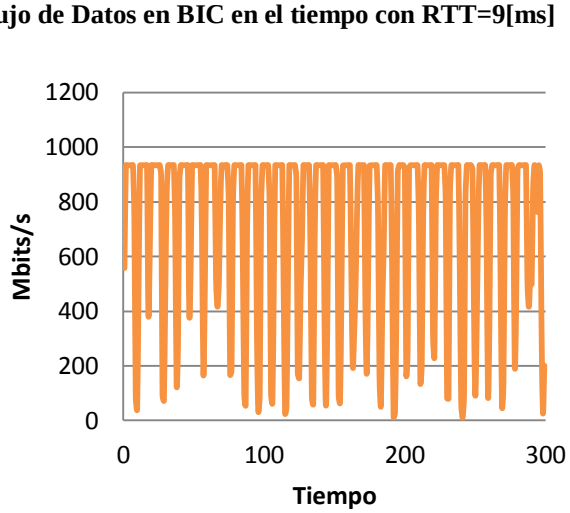
Flujo de Datos en BIC en el tiempo con RTT=7[ms]



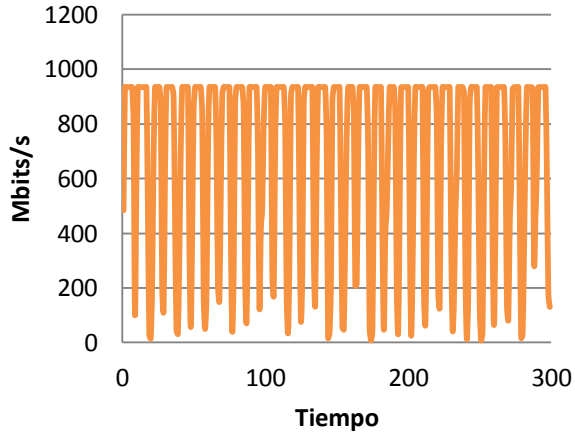
Flujo de Datos en BIC en el tiempo con RTT=8[ms]



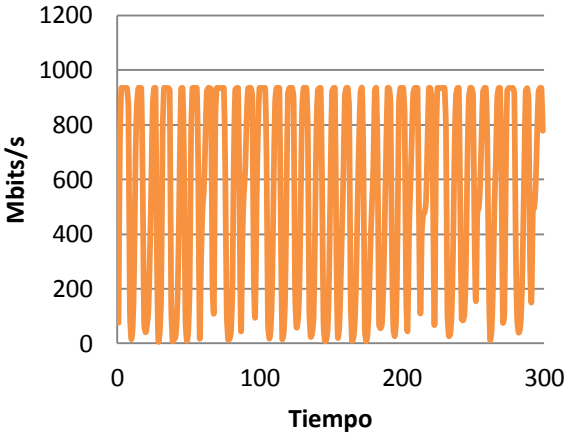
Flujo de Datos en BIC en el tiempo con RTT=9[ms]



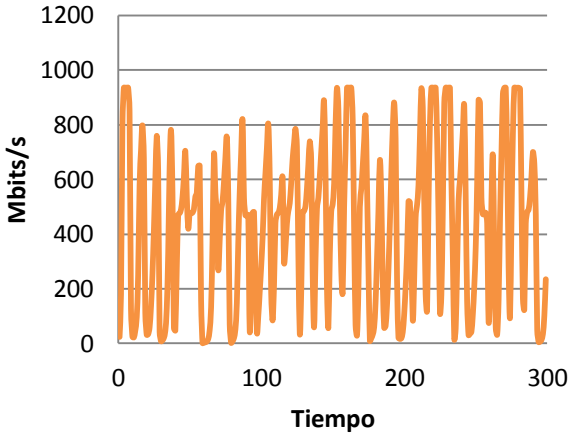
Flujo de Datos en BIC en el tiempo con RTT=10[ms]



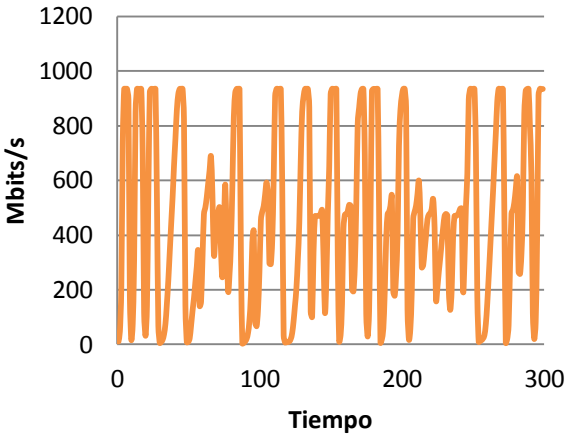
Flujo de Datos en BIC en el tiempo con RTT=20[ms]



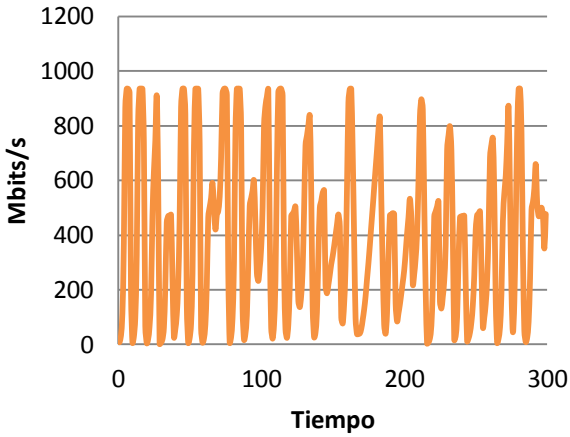
Flujo de Datos en BIC en el tiempo con RTT=30[ms]



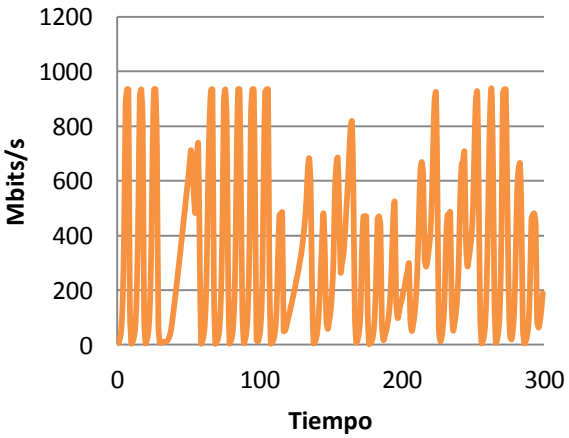
Flujo de Datos en BIC en el tiempo con RTT=40[ms]



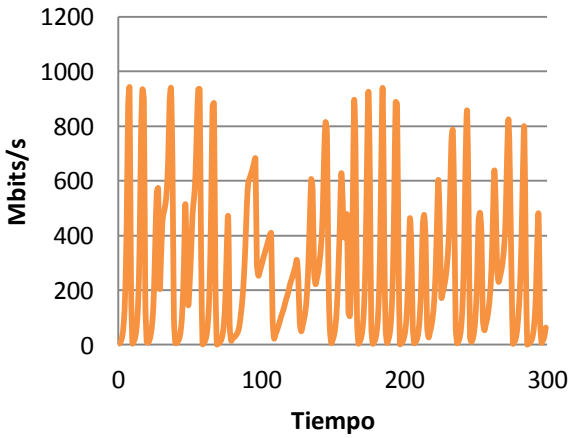
Flujo de Datos en BIC en el tiempo con RTT=50[ms]



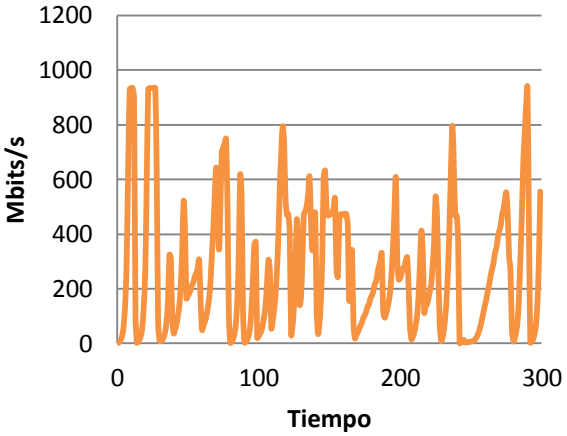
Flujo de Datos en BIC en el tiempo con RTT=60[ms]



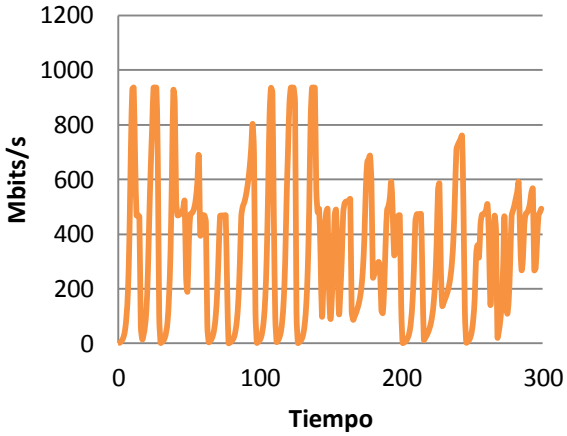
Flujo de Datos en BIC en el tiempo con RTT=70[ms]



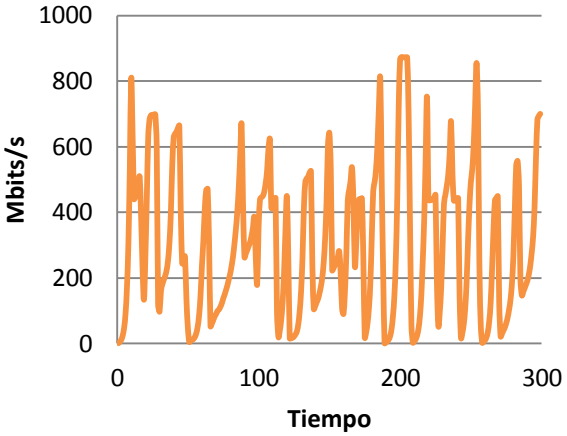
Flujo de Datos en BIC en el tiempo con RTT=80[ms]



Flujo de Datos en BIC en el tiempo con RTT=90[ms]

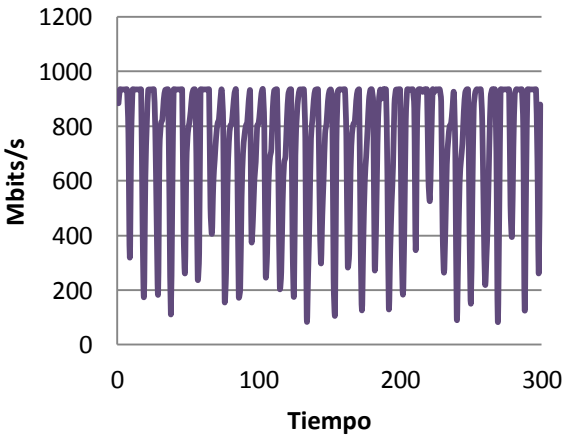


Flujo de Datos en BIC en el tiempo con RTT=100[ms]

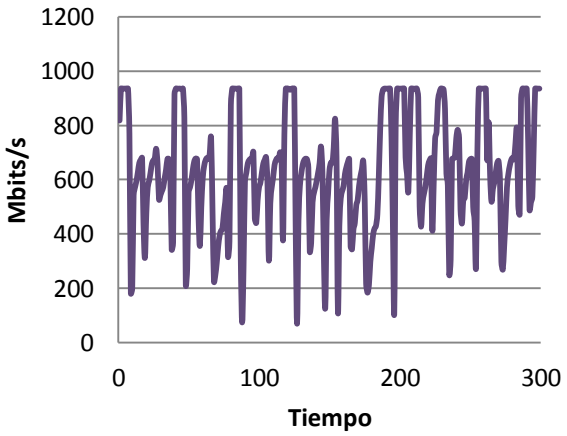


C. CUBIC

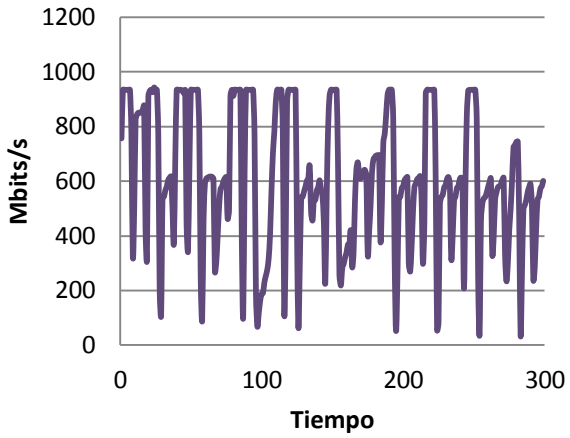
Flujo de Datos en CUBIC en el tiempo con RTT=1[ms]



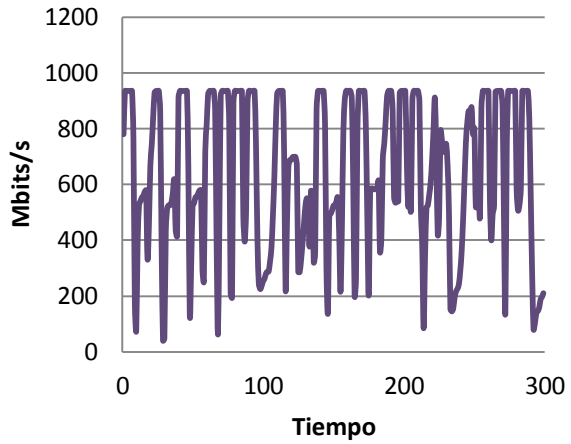
Flujo de Datos en CUBIC en el tiempo con RTT=2[ms]



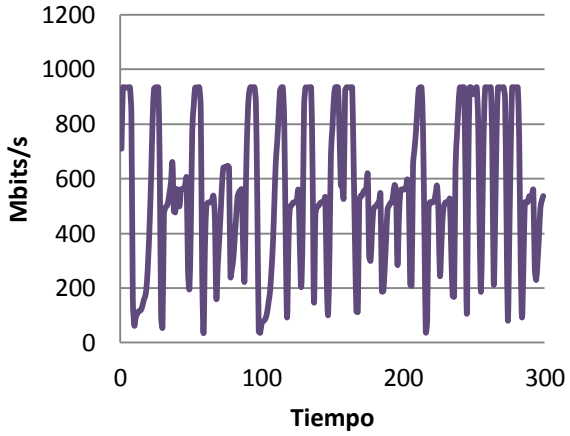
Flujo de Datos en CUBIC en el tiempo con RTT=3[ms]



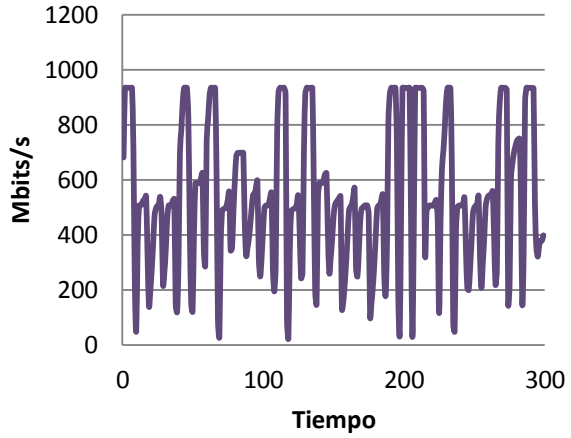
Flujo de Datos en CUBIC en el tiempo con RTT=4[ms]



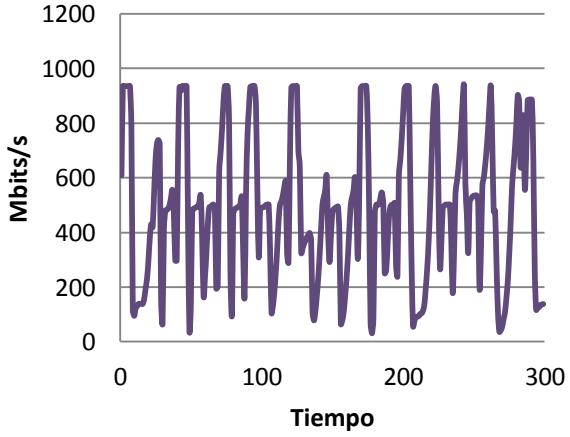
Flujo de Datos en CUBIC en el tiempo con RTT=5[ms]



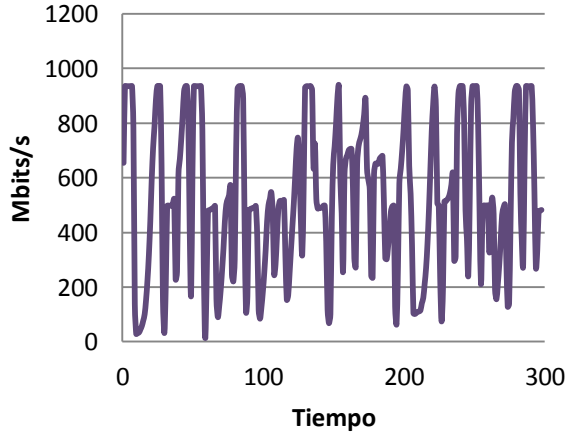
Flujo de Datos en CUBIC en el tiempo con RTT=6[ms]



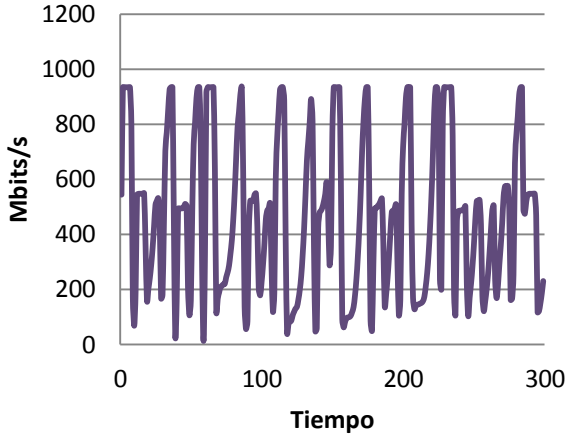
Flujo de Datos en CUBIC en el tiempo con RTT=7[ms]



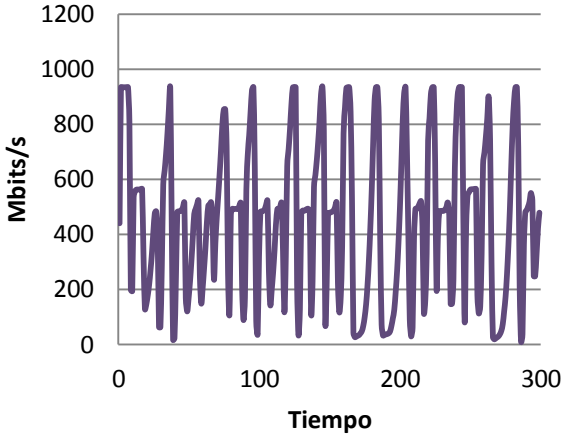
Flujo de Datos en CUBIC en el tiempo con RTT=8[ms]



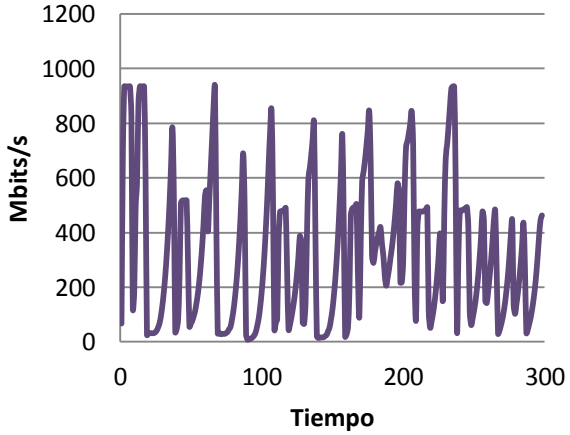
Flujo de Datos en CUBIC en el tiempo con RTT=9[ms]



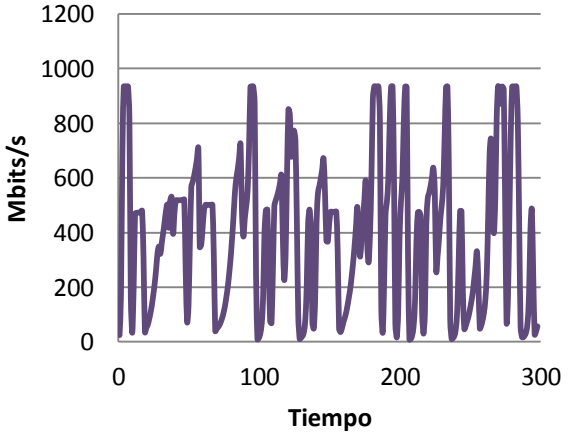
Flujo de Datos en CUBIC en el tiempo con RTT=10[ms]



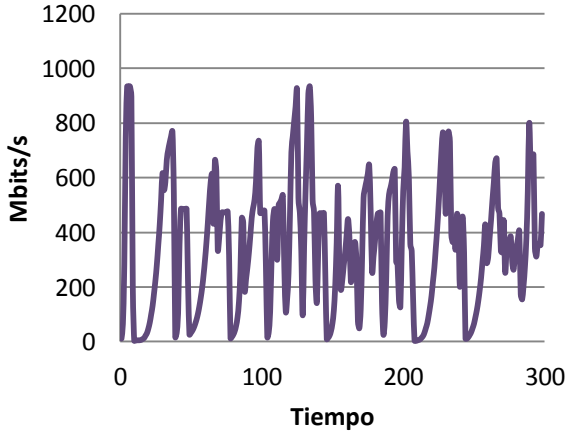
Flujo de Datos en CUBIC en el tiempo con RTT=20[ms]



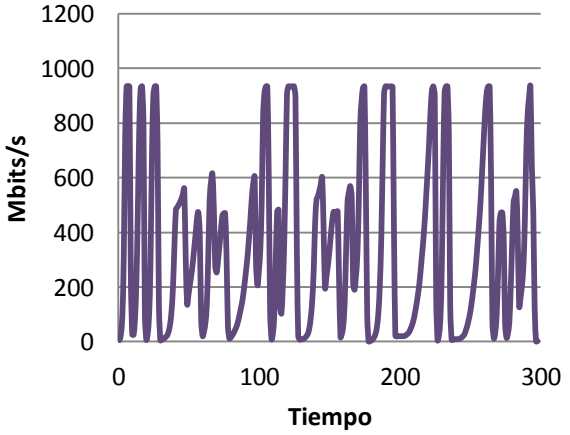
Flujo de Datos en CUBIC en el tiempo con RTT=30[ms]



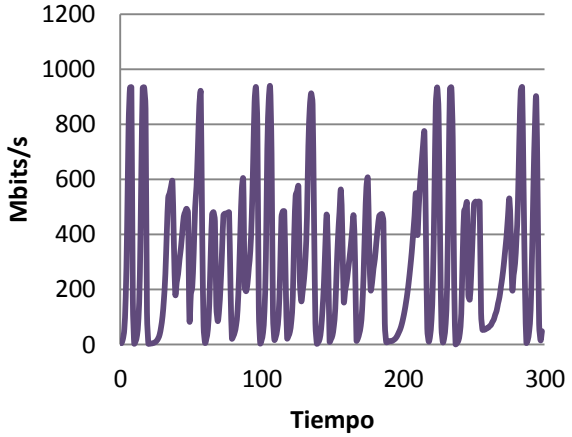
Flujo de Datos en CUBIC en el tiempo con RTT=40[ms]



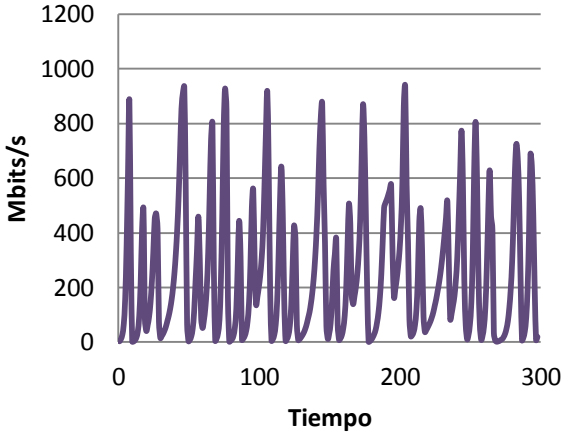
Flujo de Datos en CUBIC en el tiempo con RTT=50[ms]



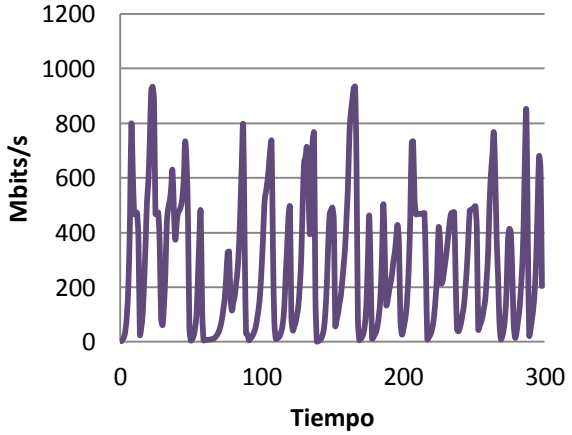
Flujo de Datos en CUBIC en el tiempo con RTT=60[ms]



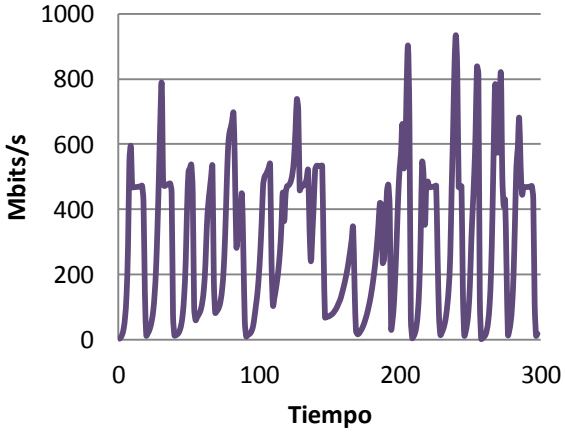
Flujo de Datos en CUBIC en el tiempo con RTT=70[ms]



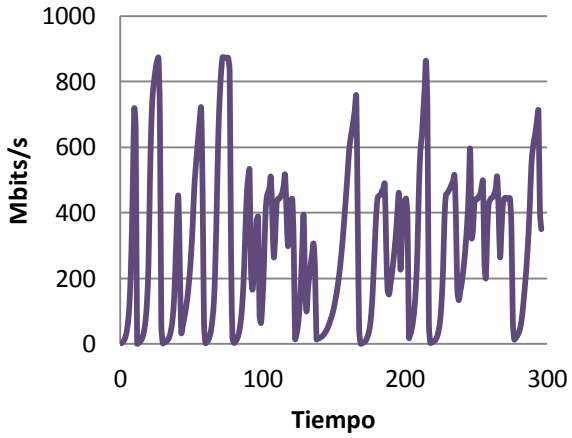
Flujo de Datos en CUBIC en el tiempo con RTT=80[ms]



Flujo de Datos en CUBIC en el tiempo con RTT=90[ms]

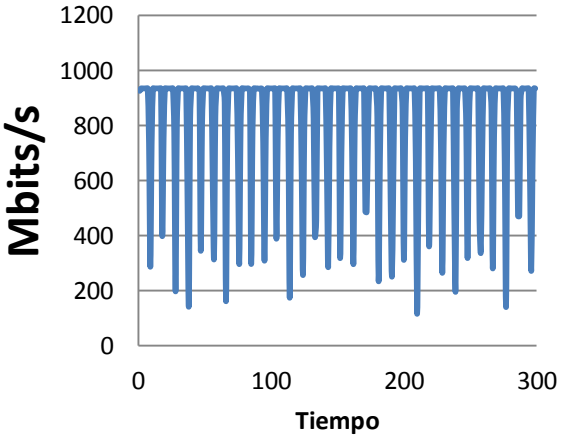


Flujo de Datos en CUBIC en el tiempo con RTT=100[ms]

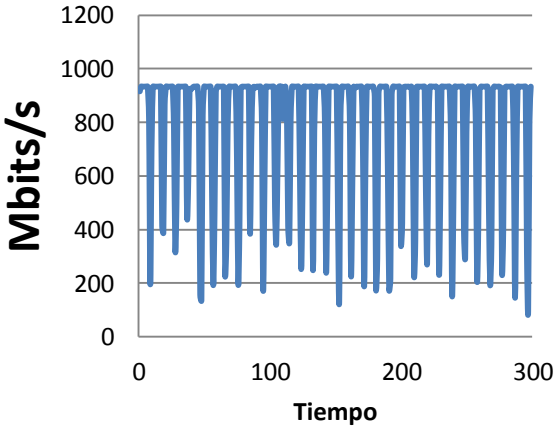


D. RENO

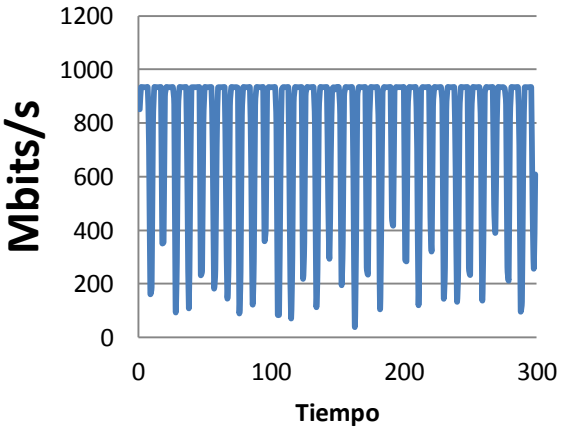
Flujo de Datos en RENO en el tiempo con RTT=1[ms]



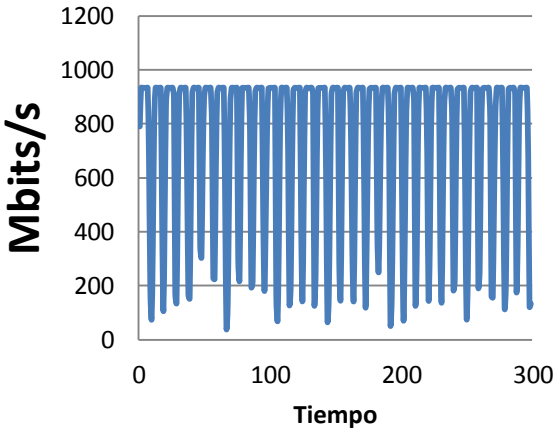
Flujo de Datos en RENO en el tiempo con RTT=2[ms]



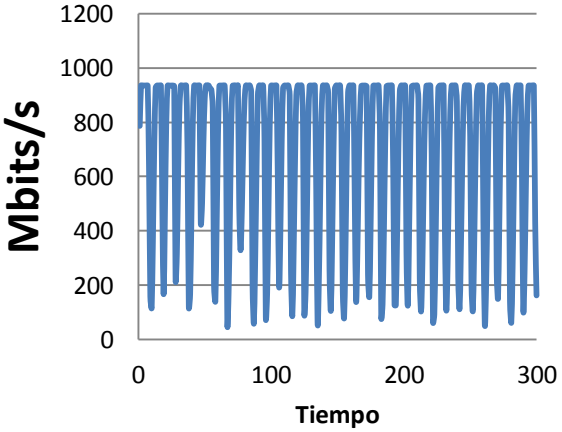
Flujo de Datos en RENO en el tiempo con RTT=3[ms]



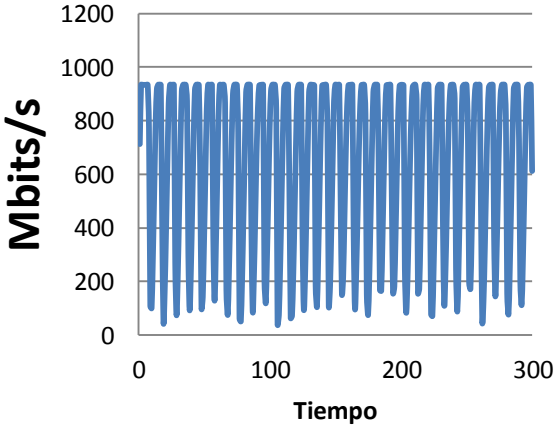
Flujo de Datos en RENO en el tiempo con RTT=4[ms]



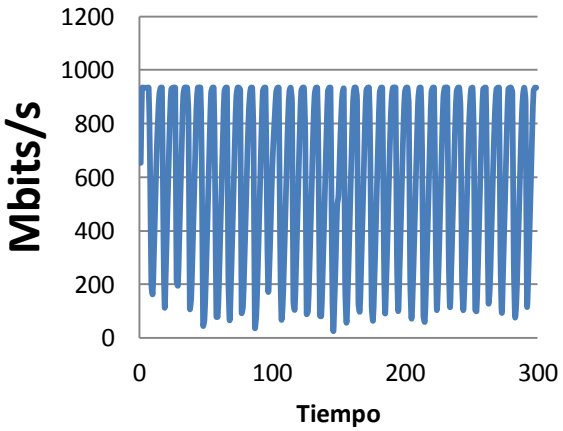
Flujo de Datos en RENO en el tiempo con RTT=5[ms]



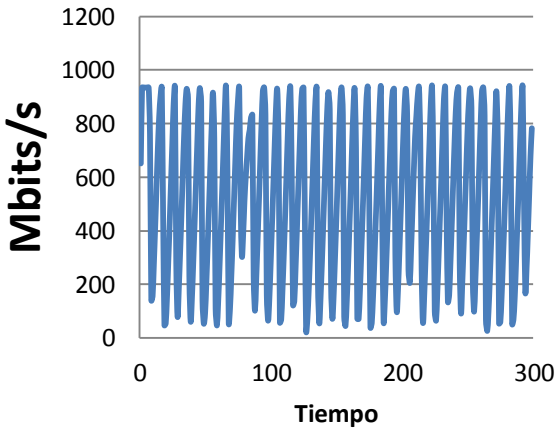
Flujo de Datos en RENO en el tiempo con RTT=6[ms]



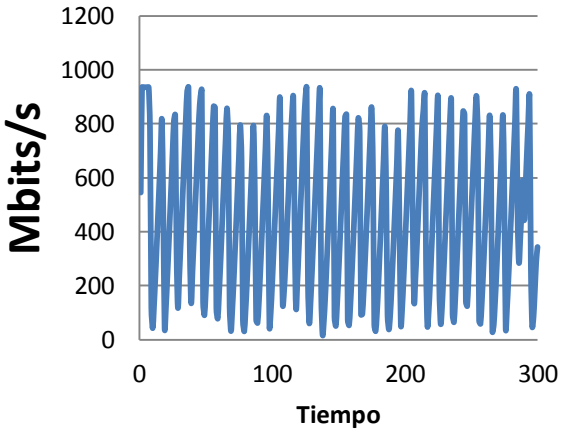
Flujo de Datos en RENO en el tiempo con RTT=7[ms]



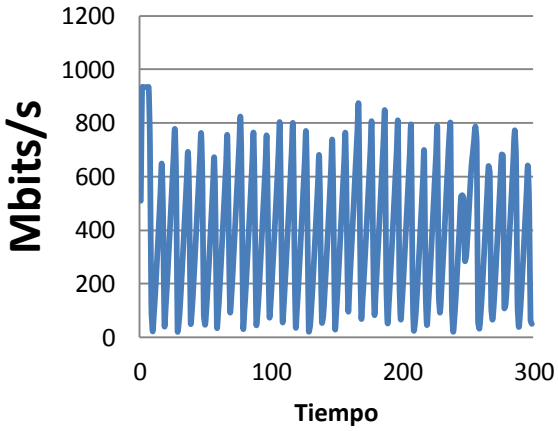
Flujo de Datos en RENO en el tiempo con RTT=8[ms]



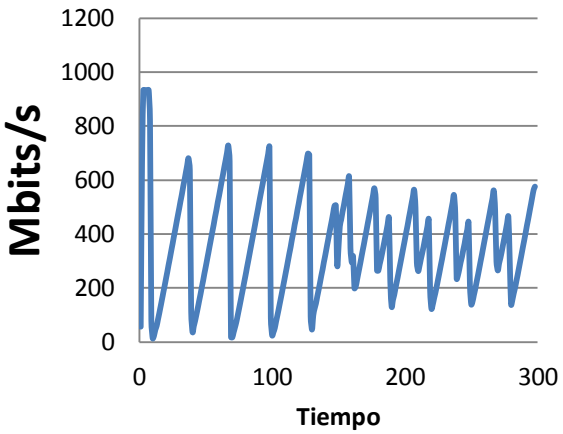
Flujo de Datos en RENO en el tiempo con RTT=9[ms]



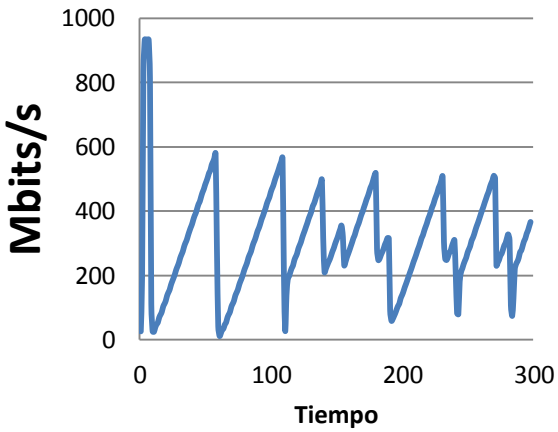
Flujo de Datos en RENO en el tiempo con RTT=10[ms]



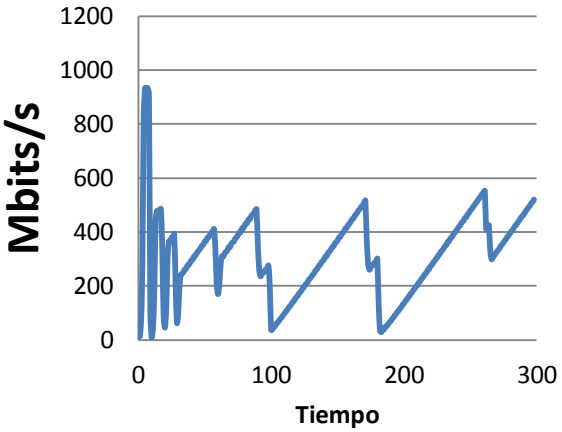
Flujo de Datos en RENO en el tiempo con RTT=20[ms]



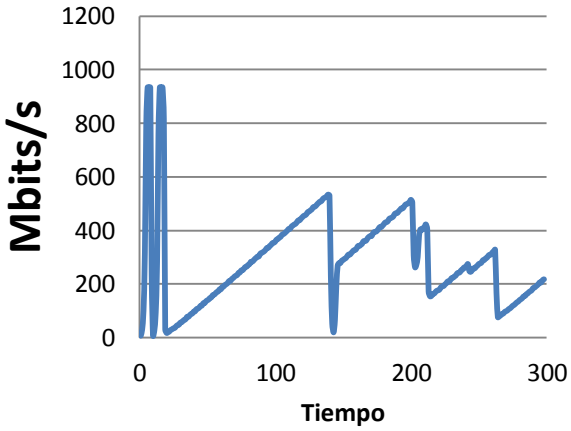
Flujo de Datos en RENO en el tiempo con RTT=30[ms]



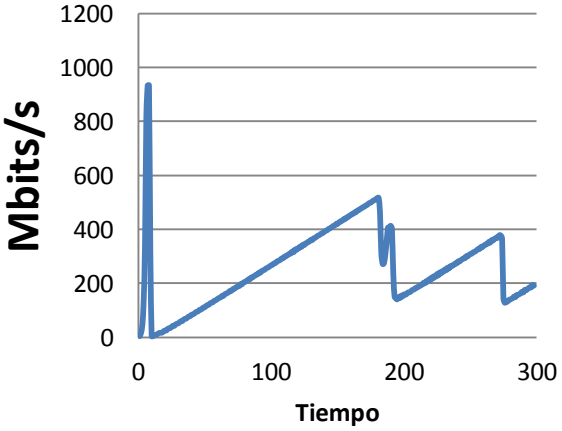
Flujo de Datos en RENO en el tiempo con RTT=40[ms]



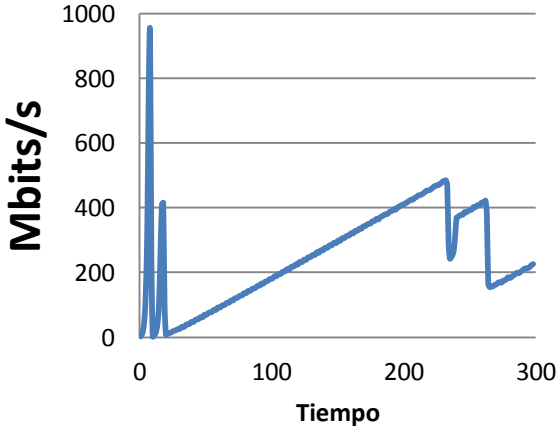
Flujo de Datos en RENO en el tiempo con RTT=50[ms]



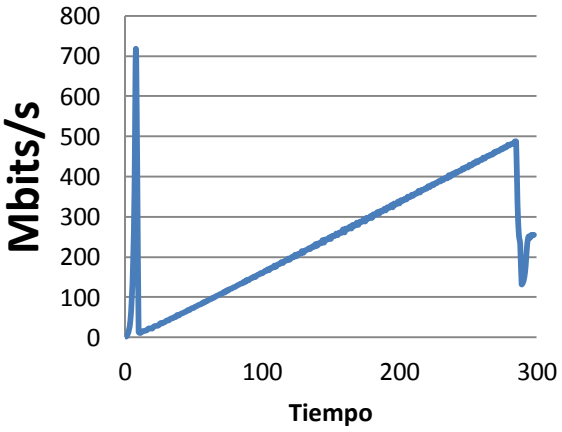
Flujo de Datos en RENO en el tiempo con RTT=60[ms]



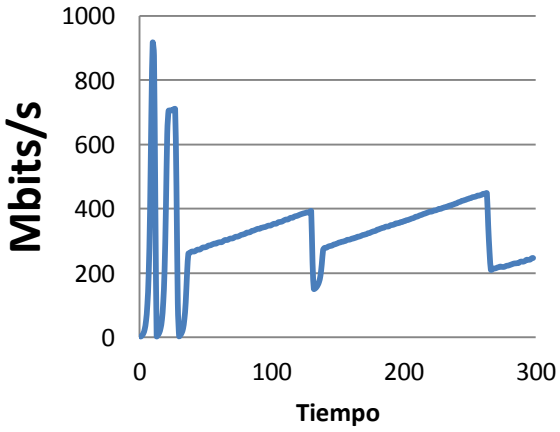
Flujo de Datos en RENO en el tiempo con RTT=70[ms]



Flujo de Datos en RENO en el tiempo con RTT=80[ms]



Flujo de Datos en RENO en el tiempo con RTT=90[ms]



Flujo de Datos en RENO en el tiempo con RTT=100[ms]