COMPARACIÓN ENTRE LOS TIPOS DE TCP RENO, SACK Y VEGAS DESDE EL PUNTO DE VISTA DEL CONTROL DE FLUJO

A FLOW-CONTROL APPROACH TO COMPARING RENO TCP, SACK TCP AND VEGAS TCP

ABSTRACT
This paper attempts to compare the performance of various TCP implementations in terms of flow control. The comparison makes use of both quantitative and qualitative analyses of the characteristics of each model. In order to validate the theory, simulations are run using network simulator NS-2.

Key words: congestion, flow control, FTP, NS-2, packets, TCP RENO, TCP SACK, TCP VEGAS, Telnet, window.

RESUMEN
El artículo pretende hacer una comparación desde el punto de vista del control de flujo de las diferentes implementaciones de TCP haciendo uso de análisis cualitativo y análisis cuantitativos de las características inherentes a cada modelo. Para realizar las validaciones se va a emplear el software de redes NS-2 de libre distribución y utilización.

Palabras claves: congestión, control de flujo, FTP, NS-2, paquetes, TCP RENO, TCP SACK, TCP VEGAS, Telnet, ventana.

Nancy Yaneth Gélvez García
Magister en Ciencias de la Información y las Comunicaciones
Docente planta de la Universidad Distrital Francisco José de Caldas
nayagarcia@udistrital.edu.co
Bogotá, Colombia

Andrés Rogelio Córdoba Ramírez
Estudiante de Ingeniería Electrónica
Universidad Distrital Francisco José de Caldas
arcordobar@correo.udistrital.edu.co
Bogotá, Colombia

Jairo Andrés Supelano Rátiva
Estudiante de Ingeniería Electrónica
Universidad Distrital Francisco José de Caldas
jandres084@hotmail.com
Bogotá, Colombia

Tipo: Artículo de revisión

Fecha de Recepción: Septiembre 9 de 2012
Fecha de Aceptación: Noviembre 1 de 2012

1. INTRODUCCIÓN

Desde sus inicios el protocolo de nivel del transporte de datos TCP ha sufrido diferentes cambios, los cuales se han hecho con el fin de mejorarlo, algunas de esas implementaciones incluyen a TCP TAHOE, RENO, NEW RENO, SACK y VEGAS entre otras, cada una de ellas tiene sus particularidades para manejar el flujo de datos (control de congestión). Existe varios trabajos que muestran aspectos inherentes a cada uno de ellos en donde se investigan desde diferentes puntos de vista [1, 3, 5, 6, 7, 9, 10, 13, 14, 17, 19, 20, 21, 22, 24, 26, 27, 28, 31, 32, 36, 38], algunos de estos trabajos comparan las implementaciones de TCP entre ellas, otros solamente se dedican a evaluar el comportamiento de una sola implementación en particular.

Se ha encontrado que la herramienta NS-2 es muy usada a la hora de evaluar diferentes escenarios de redes a diferentes niveles y esta no va a ser la excepción [41], por lo tanto en la realización de las comprobaciones en este trabajo se emplearán scripts que implementan la topología de red propuesta aquí.

Este trabajo resulta importante debido a que toma un punto de vista para evaluar las implementaciones de TCP en el que primero se examinan las capacidades individuales para el control de flujo de una forma cualitativa y luego con ayuda de los modelos encontrados en trabajos relacionados se hace un análisis cuantitativo de su comportamiento.

También se hace un análisis de tráfico que se plantea en [3] y el cual pone a prueba los modelos analíticos planteados en el caso que se tenga un tráfico intermitente (on-off), el cual no es tenido en cuenta en la mayoría de trabajos relacionados.

Con todos los antecedentes mencionados se plantean dos objetivos en los cuales se va a hacer uso de tráficos tipo FTP y Telnet.

- Comparar el comportamiento de la ventana y cantidad de paquetes perdidos ante congestión en cada implementación de TCP.
- Validar los modelos matemáticos planteados de los tipos de TCP bajo los dos tipos de tráfico definidos anteriormente.

Dentro de esta temática se pueden encontrar varios trabajos previos que abordan diferentes aspectos de cada uno de los tipos de TCP.

Para obtener los modelos matemáticos fue necesario recurrir a diferentes trabajos que dan puntos de vista parecidos; en [3] se encuentra una innovación en cuanto a los modelos existentes [7] para TCP-VEGAS, porque en este trabajo se tiene en cuenta tráfico tipo HTTP, donde se inician y cierran conexiones frecuentemente, en comparación a los que se tenía que solo consideraba tráfico de grandes cantidades de información en una sola conexión (tipo FTP). También se da una forma general en la cual se puede comparar de manera consistente los diferentes tipos de TCP sin importar las topologías de red y si el tráfico es tipo FTP (grandes cantidades de datos sin cerrar conexión) o HTTP (pocos datos pero muchas conexiones).

En [4] se muestran algunas de las ventajas de VEGAS frente a RENO mediante análisis y simulaciones desde el punto de vista del uso de los recursos de la red, como son el ancho de banda y la distribución del mismo entre diferentes nodos que necesiten comunicarse.

En [39] se presentan simulaciones en NS-2 para ver el efecto de usar diferentes tipos de TCP en una red, para observar el efecto que esto tiene sobre la distribución de los recursos a cada uno de los nodos de la red. También se tienen en cuenta algoritmos de encolamiento como DropTail y RED, sin dejar de lado que hay consideraciones sobre el comportamiento de la ventana de congestión para las diferentes implementaciones de TCP.

Este artículo tiene como resultados la comprobación de los planteamientos matemáticos propuestos en [3], junto con consideraciones de mejor comportamiento en el control de flujo tomando como parámetros la cantidad de paquetes perdidos y la cantidad de paquetes entregados para evaluar un modelo frente a otro.

En el documento se obtuvo evidencia de que la implementación de VEGAS obtiene el mejor desempeño al no perder ningún paquete incluso cuando aumenta la congestión.

Los modelos matemáticos presentados en [3] permiten corroborar las simulaciones efectuadas y permite visualizar que en redes que no son ideales, se tiene más robustez por parte de SACK, frente a RENO.

La propuesta de tráfico on-off permite evaluar el comportamiento de cada implementación y deducir que en tráficos TELNET, los 3 modelos de control de ventana de capa de transporte RENO, SACK y VEGAS necesitan un tiempo para el ajuste de su ventana, al no tenerlo la ventana tiene un comportamiento que no muestra tendencia a algún patrón determinado.

El documento está organizado de la siguiente manera: en la sección 2 se explica la manera por medio de la cual se harán las simulaciones de las implementaciones de TCP, en la sección 3 se muestran las similitudes, diferencias y modelos matemáticos de cada una de las implementaciones de TCP, en la sección 4 se muestran las pruebas que se llevaron a cabo para probar cada implantación, en la sección 5 se discuten los resultados obtenidos y por último en la sección 6 se muestran las conclusiones a las que se llegaron.

2. METODOLOGÍA

Para comprobar los planteamientos del funcionamiento de los diferentes algoritmos de control de flujo de TCP, se van a realizar simulaciones en el software de uso libre NS-2 [40]. Para observar los resultados se hace uso de las gráficas que produce el simulador. Para comprobar los modelos matemáticos primero se realizan las simulaciones con los parámetros nombrados en la sección 4 y luego se hace uso de las expresiones expuestas en 3 para evaluar la validez de las mismas. Se hace un análisis cualitativo basado en los resultados numéricos y gráficos arrojados por el simulador para lograr sacar conclusiones de cual de las implementaciones es mejor.

3. ALGORITMO DE CONTROL DE FLUJO, VARIANTES DE TCP Y SUS MODELOS ANALÍTICOS

En esta sección se describen los tipos de TCP SACK, RENO y VEGAS desde el punto de vista de su concepción y luego para cada uno se describe un modelo analítico que permite hacer predicciones sobre su comportamiento en redes reales.

3.1. Algoritmo de control de flujo de TCP

Para el correcto funcionamiento de TCP se hace necesario tener presente un algoritmo encargado de mantener la tasa de envío de paquetes en un nivel adecuado de tal manera que la cantidad de paquetes perdidos sea mínima y por lo tanto la cantidad de retransmisiones necesarias también lo sea.

El algoritmo de control de flujo de TCP incluye cuatro componentes esenciales, los cuales son implementados por todas las variantes de TCP excepto por TCP VEGAS [39].

- Inicio lento, es un mecanismo encargado de "censar" el ancho de banda disponible cuando se inicia una conexión.
- Incremento aditivo decrecimiento multiplicativo (AIMD por sus siglas en inglés), se encarga de disminuir el tamaño de la ventana de congestión cada vez que se encuentra con una ventana que contenga la pérdida de un paquete.
- Temporizador de retransmisión, se encarga de llevar cuenta el reconocimiento de envío de un paquete, si no llega se vence el temporizador y se realiza la retransmisión.
- Temporización de reconocimientos, es un mecanismo empleado en el nodo que envía para temporizar el envío de nuevos datos.

En la primera etapa el tamaño de la ventana de congestión tiene un tamaño de 1; a medida que va recibiendo reconocimientos de los paquetes que ha enviado, va agregando un paquete a la ventana de congestión, si recibe dos reconocimientos, se le agregan dos paquetes a la ventana y así sucesivamente, mostrando un

crecimiento exponencial. Cuando La ventana alcanza un tamaño umbral, deja de crecer de esa forma y pasa a la segunda etapa, llamada etapa de control de flujo, donde el crecimiento ahora es más lento; al ritmo de un paquete por cada w cantidad de paquetes (w es el tamaño de la ventana). Si se experimenta una pérdida de algún paquete, la ventana se ve disminuida drásticamente, completando de esa manera el esquema de crecimiento aditivo y decremento multiplicativo (AIMD) [40].

3.2. TCP RENO

TCP RENO incluye algunas mejoras con respecto a implementaciones anteriores de TCP, entre las cuales están algoritmos para evitar la congestión, retransmisión y recuperación rápida. La última mejora evita que el canal se quede desocupado luego de que se haga una retransmisión [40]. La idea tras la rápida recuperación es que el transmisor reenvíe el paquete que lo hizo entrar en ese modo. Luego que ha entrado en el modo de recuperación rápida no hay necesidad de entrar en el modo de inicio lento. Para evitar entrar en el modo de inicio lento RENO usa reconocimientos adicionales para temporizar posteriores paquetes [26].

El correcto funcionamiento del algoritmo de control de congestión de RENO depende de la pérdida de paquetes, estas pérdidas permiten determinar el ancho de banda disponible y así evolucionar el tamaño de la ventana. Luego de que se pasa el umbral preestablecido para el modo de inicio lento, se pasa al modo de control de flujo (Congestión avoidance), en este modo el tamaño de la ventana de RENO va a seguir aumentando en uno cada vez que hay un RTT disponible hasta que haya una pérdida de un paquete, momento en el cual RENO retransmite, y reduce el tamaño de su ventana a la mitad. Este comportamiento se llama incremento aditivo y decremento multiplicativo el cual es aplicado en el modo de control de flujo [39].

3.2.1. Modelo analítico

El modelo analítico empleado para RENO y los demás tipos de TCP está detalladamente descrito en [3]; aquí solamente se muestran los resultados obtenidos y el significado de cada expresión.

$$P_{to}^{R} = \begin{Bmatrix} \sum_{i=3}^{w} P_w(i) \;\; si \;\; w \geq 10 \\ \sum_{i=2}^{w} P_w(i) \;\; si \;\; 4 \leq w \leq 10 \\ 1 - P_w(0) \;\; si \;\; w < 4 \end{Bmatrix} \quad (1)$$

La ecuación (1) es una expresión que modela la probabilidad de que el temporizador (timeout) llegue a su final, w simboliza la longitud de la ventana en número de paquetes.

$$P_{fr/fr}^{R} = \begin{Bmatrix} P_w(1) + P_w(2) \;\; si \;\; w \geq 10 \\ P_w(1) \;\; si \;\; 4 \leq w \leq 10 \\ 0 \;\; si \;\; w < 4 \end{Bmatrix} \quad (2)$$

La ecuación (2) muestra la probabilidad de que encuentra retransmisión y recuperación rápida (fast retransmit/fast recovery).

$$P_w(i) = \binom{w}{i} P^i (1 - P)^{w-i} \quad (3)$$

La ecuación (3) señala la probabilidad de pérdida de i paquetes por fuera de una ventana w de paquetes.

3.3. TCP SACK

Los algoritmos para el control de la congestión implementados en SACK (selective AC Knowledgement), son básicamente los mismos que se implementan para RENO pero con la diferencia que SACK incluye una variable llamada pipe que lleva cuenta de la cantidad de paquetes que están moviéndose por la red en el modo de rápida recuperación, y permite tener un comportamiento diferente a la hora de tener múltiples paquetes perdidos en una misma ventana.

El nodo que transmite solamente retransmite paquetes cuando el número estimado de paquetes moviéndose por la red es menor que la ventana de congestión.

La variable *pipe* se incrementa en 1 cuando se

retransmite un paquete anterior o envía un paquete nuevo, y se reduce en 1 cuando el nodo que envía recibe un reconocimiento con la opción SACK indicando que paquetes nuevos han sido recibidos por el destinatario.

El uso de la variable *pipe* permite desacoplar la decisión de cuando enviar un paquete de la decisión de cual paquete enviar. Para lograr el reenvío selectivo el nodo que envía mantiene una estructura de datos llamada *scoreboard* que le permite llevar cuenta de reconocimientos de opciones SACK anteriores. Cuando al nodo que envía se le permite transmitir él envía el paquete que infiere que está faltando en el receptor. Si no está tal paquete y el receptor ha notificado que su ventana es lo suficientemente grande, el nodo que envía, pone nuevos paquetes en la red

Para mayores detalles en el funcionamiento de TCP SACK por favor referirse a [26, 31].

3.3.1. Modelo analítico

En [3] se asume que hay pérdidas de paquetes independientes y muestran el cálculo de la probabilidad de que el contador llegue a su límite en la ecuación (4).

$$P_{to}^{s}(w) = \sum_{i=1}^{w-3} P_w(i) \cdot (1-(1-p)^i) + \sum_{i=w-2}^{w} P_w(i), w \geq 4 \quad (4)$$

Y la probabilidad de que entre en modo de reenvío y recuperación rápida mediante la ecuación (5).

$$\begin{aligned} P_{fr/fr}^{s} &= 1 - P_{to}^{s}(w) - P_w(0) \\ P_{fr/fr}^{s} &= \sum_{i=1}^{w-3} P_w(1-P)^i, w \geq 4 \end{aligned} \quad (5)$$

3.4. TCP VEGAS

TCP VEGAS propone mecanismos para evitar la congestión que difieren de aquellos usados por TCP SACK y RENO. Mientras que TCP RENO requiere tener la pérdida de paquetes para censar el ancho de banda disponible, TCP VEGAS tiene como objetivo detectar la congestión en sus etapas iniciales y reducir su tasa de envío de datos (throughput) para prevenir que haya pérdida de paquetes.

Una de las ventajas de VEGAS es que tiene un comportamiento de inicio lento diferente con respecto a los otros TCP. Para lograrlo VEGAS revisa a *diff* (Número estimado de paquetes transitando por la red) cada vez que hay un RTT (round trip time).

En el modo de inicio lento el tamaño de la ventana es doblado cada vez que se recibe un RTT para ver el retardo de los paquetes con cada tamaño de ventana. Este retarde es usado para calcular *diff* y decidir si continuar en el modo de inicio lento. Específicamente si *diff* supera el valor del umbral $\gamma=(\alpha+\beta)/2$, (en donde α y β representan umbrales predeterminados para la longitud de la ventana), VEGAS incrementa el tamaño de la ventana por un paquete [3].

Como en TCP RENO tres reconocimientos duplicados resultan en la transmisión de paquetes, pero los temporizadores implementados ayudan a detectan la pérdida de paquetes antes, haciendo necesario solamente uno o dos reconocimientos [20].

Cuando hay una retransmisión debido a un reconocimiento duplicado, el tamaño de la ventana se reduce a $3w/4$ en comparación a la reducción de RENO que es de $w/2$.

3.4.1. Modelo analítico

Para que VEGAS detecte congestión, se hace el cálculo de *diff* (ecuación (6)) cada vez que hay un RTT; para ello se usa el tamaño de la ventana actual W, el último RTT, y el RTT más pequeño observado hasta el momento.

$$\begin{aligned} diff &= \left(\frac{W}{baseRTT} - \frac{W}{RTT}\right) \cdot baseRTT \\ diff &= \left(\frac{RTT - baseRTT}{RTT}\right) \cdot W \end{aligned} \quad (6)$$

Debido a que (*RTT - baseRTT*) es el retardo de encolamiento de la red y *W/RTT* es un estimado de la tasa de transferencia actual; el producto de estas dos es un estimativo de la cantidad

de paquetes transmitiéndose por la red.

El objetivo del algoritmo de control de congestión de VEGAS es mantener a *diff* dentro de un rango predefinido por dos límites; α y β, de esa maneta cada vez que hay un RTT nuevo y si no está en el modo de inicio lento, VEGAS ajustan el tamaño de su ventana mediante la ecuación (7) [20].

$$W = \begin{cases} W + 1, diff < \alpha \\ W \quad , \alpha \leq diff \leq \beta \\ W - 1, diff > \beta \end{cases} \qquad (7)$$

4. REALIZACIÓN DE PRUEBAS

Para cumplir los objetivos es necesario hacer unas simulaciones que validen los planteamientos hechos sobre la red. En primer lugar se describe la topología de la red empleada y luego su dinámica.

Para la realización de las simulaciones se emplean 6 nodos que funcionan de la siguiente forma: Los nodos marcados como 0, 1, 4, 5 en la figura 1 son las terminales de datos, los nodos 2 y 3 se encargan de encolar y reenviar paquetes, 0 y 1 son los emisores, que envían información a 4 y a 5 respectivamente, el nodo 0 implementa TCP con sus diferentes implementaciones y 4 le envía reconocimientos, 1 envía tráfico con tasa de envío constante y es usado solamente para generar congestión en la conextión intermedia entre 2 y 3. Las conexiones tienen un ancho de banda y retraso de acuerdo a los que se especifican en la figura 1. Las gráficas que se muestran en las pruebas son las correspondientes al tamaño de ventana y pérdida de paquetes de 0 en su conexión con 4.

Para las pruebas de FTP sobre las implementaciones de TCP que se realizaron se mantiene la siguiente dinámica: Primero se inicia una sección TCP entre las termínales 0 y 4, en la cual se comienzan a enviar datos en masa en el momento desde que inicia la simulación, luego a los 60s de corrida la simulación 1 inicia transferencia de datos en masa hacia 5, y solamente se hace para generar congestión entre los nodos 2 y 3, la simulación corre por 125s.

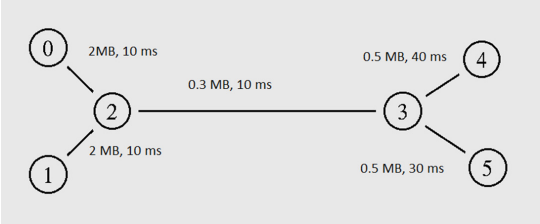

Figura 1. Esquema de red empleado.

Para las pruebas hechas con tráfico Telnet se hace de la misma forma que el caso anterior pero con la diferencia que ahora el nodo 0 envía datagramas al nodo 4 con una aplicación Telnet en vez de FTP para observar las misma variables.

4.1. TCP RENO con trafico FTP

La primera prueba se realizó fue con la implementación de TCP RENO, en la cual se monitoreó la evolución del tamaño de la ventana en el tiempo, y la cantidad de paquetes que la implementación dejó perder.

En la figura 2 se pueden observar las diferentes etapas por las que pasa el tamaño de la ventana en RENO para iniciar su modo de control de congestión.

Al inicio de la conexión se nota que el tamaño de la ventana crece de una forma muy rápida, puede decirse que exponencialmente, hasta que se reporta la pérdida de más de 18 paquetes, es en este momento en el que el protocolo se encuentra censando el ancho de banda disponible; luego de la pérdida de esta cantidad de paquetes el tamaño de la venta se reduce hasta 0 y de nuevo se nota que aparece el inicio lento, generando de esa manera un crecimiento exponencial, hasta que se sobrepasa el umbral para cambiar de fase marcado por la línea azul, y de inmediato se nota que el crecimiento en la fase de control de congestión tiende a ser un poco más lineal, en ese momento se encuentra en la fase de control de flujo, cuando se pierde un paquete adicional, pasa al estado de retransmisión rápida, cuando se reduce el tamaño de su ventana a la mitad, está en la fase de rápida

recuperación [42].

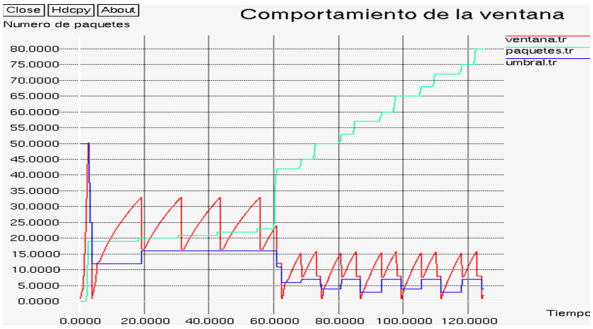

Figura 2. Tamaño de la ventana, paquetes perdidos y umbral con TCP RENO con tráfico FTP, tiempo (s), paquetes (unidades de 1000 bytes).

Se nota que la cantidad de paquetes se mantiene constante en más de 18 a medida que el tamaño de la ventana crece, cuando aparece la pérdida de un paquete adicional, de inmediato el tamaño de la ventana se reduce a la mitad y su tamaño comienza a crecer de nuevo hasta que se encuentra con la pérdida de otro paquete, esta pérdida adicional se produce porque el tamaño de la ventana crece al punto de crear este efecto. Se nota que cuando se mantiene esta dinámica de tener solamente la pérdida de un paquete es ocasionada por el crecimiento de su propia ventana y se obtiene un patrón oscilatorio periódico en el tamaño de la ventana.

Se impone un caso diferente cuando hay pérdida de paquetes debido a una razón externa como lo que se observa luego de los 60s de simulación que es cuando el nodo 1 de la figura 1 comienza a enviar paquetes a una tasa constante.

En el momento que el nodo 1 inicia su transferencia de paquetes se nota de inmediato que la cantidad de paquetes que pierde el nodo 0 es muy grande a comparación a lo que se venía teniendo, pasa de 23 a 43 paquetes, esto afecta de inmediato el tamaño de la ventana del nodo 0, la cual no alcanza a llegar al máximo que se había establecido por su propia dinámica, reacciona según su protocolo reduciendo su tamaño a la mitad, pero este tamaño se ve aún más reducido porque en esos mismo instantes se detecta la pérdida de más paquetes, lo que ocasiona que el tamaño de la ventana se caiga hasta 0, en donde RENO debe empezar con el inicio lento en lugar de seguir con la dinámica de rápida recuperación.

En la simulación de la figura 2 se observa que el umbral para cambiar de modo de inicio rápido a modo de control de congestión cambia también de acuerdo con las condiciones impuestas por el tráfico en la red, esto es evidente por la forma el punto en que se comporta el tamaño de la ventana luego de los 60s, porque ahora cuando se disminuye a la mitad el tamaño lo hace a un valor inferior al que se tenía inicialmente. El hecho de que el umbral cambie constantemente luego de los 60s puede ser considerado una evidencia de que RENO ante la pérdida de más de un paquete en una ventana produce que su algoritmo de rápido retransmisión y rápida recuperación no funciona muy bien en esa situación.

En total RENO perdió 80 paquetes que tuvieron que ser retransmitidos ocupando recursos adicionales de la red.

4.2. TCP SACK con trafico FTP

La figura 3 muestra los resultados obtenidos al hacer la simulación de la misma red y las misma condiciones anteriores pero usando la implementación de TCP SACK.

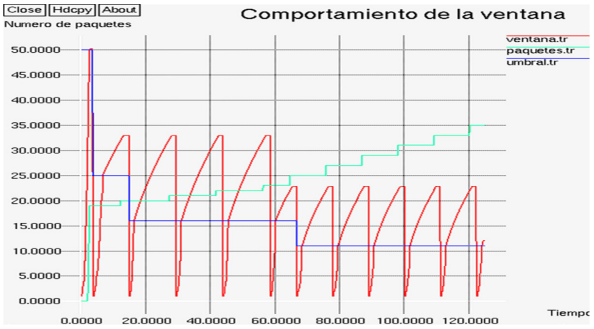

Figura 3. Tamaño de la ventana y paquetes perdidos con TCP SACK con tráfico FTP, tiempo (s), paquetes (unidades de 1000 bytes).

Teniendo en cuenta que la implementación de SACK en esencia es la misma que la de RENO, se nota un comportamiento parecido en el ta-

maño de la ventana en el cual su tamaño oscila en el tiempo.

A diferencia de RENO, SACK hace que su ventana entre en modo de inicio lento cada vez que se detecta que hay pérdida de un paquete, esto es evidente porque luego de que se estabiliza el inicio, el umbral (línea azul de la figura 3) para cambiar de fase se mantiene y es en ese número de paquetes que ventana deja de crecer exponencialmente y comienza a crecer de manera casi lineal.

En cuanto a la pérdida de paquetes es evidente que la cantidad de paquetes que SACK deja perder es inferior a las que deja perder RENO, siendo en este caso de tan solo 35 paquetes.

4.3. TCP VEGAS con trafico FTP

En la figura 4 se muestran los resultados obtenidos con la implementación de TCP VEGAS.

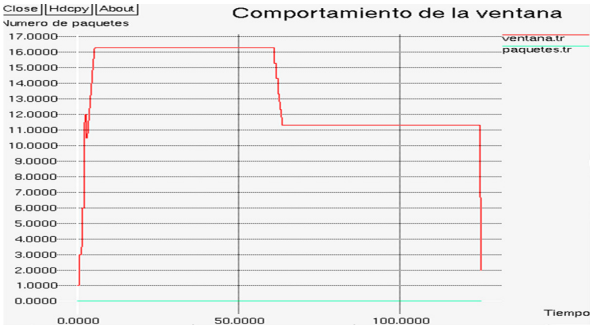

Figura 4. Tamaño de la ventana y paquetes perdidos TCP VEGAS con tráfico FTP tiempo (s), paquetes (unidades de 1000 bytes).

En el caso de VEGAS se nota de inmediato que la pérdida de paquetes es nula y que la ventana se mantiene en un valor constante mientras las condiciones de congestión de la red no cambien.

La figura 4 es evidencia de que la implementación de TCP VEGAS es muy efectiva a la hora de calcular un tamaño de ventana conveniente para no dejar que se pierda ningún paquete en el transporte de información por la red.

Tan pronto el nodo 1 de la figura 1 inicia su transferencia de datos el tamaño de ventana que calcula VEGAS disminuye lo necesario para mantener una comunicación muy confiable. Esto da luces sobre VEGAS en la medida que muestra lo acertado que es al calcular un tamaño de ventana adecuado para evitar la pérdida de paquetes.

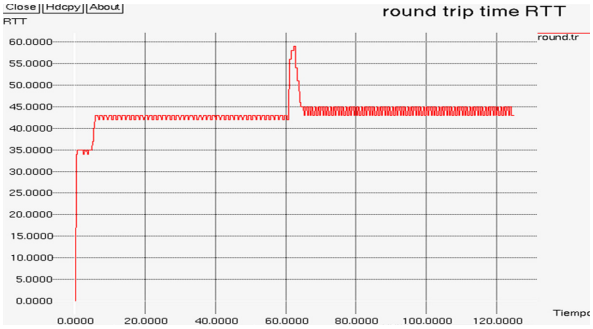

Figura 5. Round trip time para el enlace entre el nodo 0 y el nodo 4 usado por VEGAS para calcular el tamaño de la ventana tiempo (s), paquetes (unidades de 1000 bytes).

La figura 5 muestra el RTT (round trip time) que se censa todo el tiempo por TCP VEGAS para calcular el tamaño de su ventana y lograr mantener el flujo bajo control.

En la tabla 1 se muestran los resultados de la cantidad de paquetes enviados en cada implementación de TCP, junto con la cantidad de paquetes que se perdieron en el trayecto para el tráfico FTP.

Tabla 1. Relación de los paquetes perdidos junto con el total de paquetes recibidos en el nodo 4 de la figura 1 en cada una de las implementaciones de TCP para tráfico tipo FTP.

Implementación	Paquetes enviados	Paquetes recibidos
RENO	80	3297
SACK	35	3095
VEGAS	0	3777

La tabla 1 es muestra del informe que se entrega por consola luego de que se hace una simulación en el NS-2, los paquetes perdidos son el mismo valor final de la curva a verde de (paquetes) en cada una de las gráficas obtenidas

para tráfico FTP de las diferentes implementaciones de TCP.

Las siguientes simulaciones de TCP son con tráfico Telnet, el cual se ha hecho para simular tráfico que inicia y termina sesiones constantemente y ver cómo afecta a cada una de las implementaciones de TCP en su tamaño de ventana.

4.4. TCP RENO con tráfico Telnet

La figura 6 muestra los resultados al emplear tráfico generado con la aplicación telnet transportada con la implementación de TCP RENO.

En el caso de la implementación RENO se puede observar que cuando hay transmisión de paquetes, el protocolo no alcanza siquiera a salir del inicio lento, por eso no se aprecia el comportamiento de ventana oscilatoria que se tiene cuando se mantiene una sesión larga de envió de paquetes.

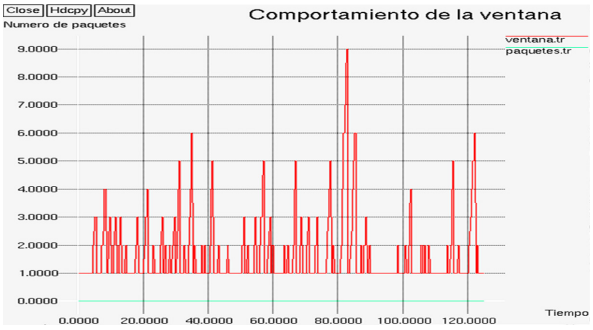

Figura 6. Tamaño de ventana y paquetes perdidos TCP RENO con tráfico Telnet, tiempo (s), paquetes (unidades de 1000 bytes).

El hecho de que haya más congestión en la red no afecta sustancialmente al protocolo; según lo visualizado en la gráfica.

4.5. TCP SACK con tráfico Telnet

En esta simulación (figura 7) se observa un comportamiento similar al que exhibe RENO en la misma situación, la el protocolo no sale del inicio lento y por lo tanto no se puede observar su comportamiento habitual.

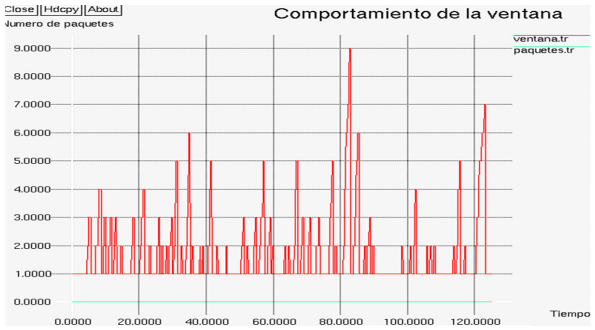

Figura 7. Tamaño de ventana y paquetes perdidos TCP SACK con tráfico Telnet tiempo (s), paquetes (unidades de 1000 bytes).

4.6. TCP VEGAS con tráfico Telnet

La simulación efectuada empleando TCP VEGAS (figura 8), no muestra una realidad diferente a la que muestran RENO y SACK, debido al corto periodo de envío de datos, la ventana no alcanza a converger a ningún valor en específico, ni siquiera se nota algún cambio sustancial luego de los 60s de simulación que es cuando el otro nodo de la red inicia la transferencia de paquetes generando congestión.

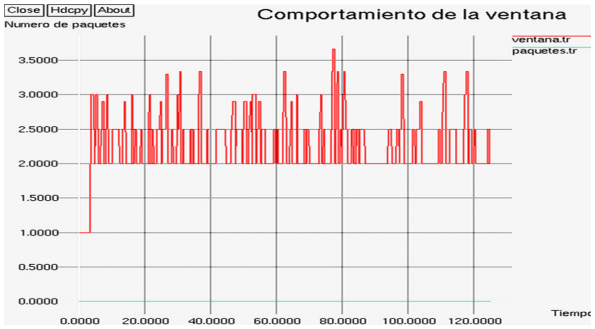

Figura 8. Tamaño de ventana y paquetes perdidos TCP VEGAS con tráfico Telnet, tiempo (s), paquetes (unidades de 1000 bytes).

Tabla 2. Relación de paquetes perdidos junto con la cantidad de paquetes recibidos en el nodo 4 de la figura 1 en cada una de las implementaciones de TCP con tráfico Telnet.

Implementación	Paquetes enviados	Paquetes recibidos
RENO	0	108
SACK	0	108
VEGAS	0	108

5. ANÁLISIS DE RESULTADOS

En esta sección primero se van a validar los resultados obtenidos en las simulaciones anteriores mediante los modelos matemáticos planteados en la sección 3, y luego se van a analizar comparativamente para decidir cual de las implementaciones resulta mejor desde el punto de vista del control de flujo.

5.1. Validación matemática

5.1.1. RENO y SACK con tráfico FTP

El comportamiento de control de flujo RENO y SACK, a nivel de probabilidad de recuperación y retransmisión del control de flujo, además la probabilidad de encontrar un tiempo de espera, se visualizan en la tabla 3 y 5, usando las ecuaciones (2) y (5). En la tabla 3, se visualiza el comportamiento de los primeros 60s, congestión baja. Los últimos 60s, congestión alta, se contemplan en la tabla 5.

Se presenta en RENO como en SACK, un comportamiento en el primer momento de cambio de ventana, una probabilidad realmente baja de recuperación y rápida retransmisión impactando directamente el funcionamiento del control de ventana, volviendo a un reinicio en el tamaño de la misma.

Tabla 3. Muestra los primeros 60s, w como el tamaño de la ventana en números de bytes y P número de bytes perdidos.

Probabilidad de rápida retransmisión y rápida recuperación			
w(bytes) P(bytes)	RENO (%)	w(bytes) P(bytes)	SACK (%)
w=63000 p=32000	$1{,}21 \times 10^{-15}$	w=63000 p=32000	$6{,}84 \times 10^{-7}$
w=33000 p=1000	56,03	w=32000 p=1000	60,72
		w=33000 p=1000	60,79

Luego se tiene un comportamiento de reacción del control de la ventana al presentarse la pérdida de 1000 bytes, donde la posibilidad de recuperación en los dos controles de flujo es superior al 50%, y donde comienza el funcionamiento normal de los controles de ventana. Para RENO se tiene una disminución de la ventana a la mitad cuando se presenta la pérdida de 1000 bytes, pero para SACK se presenta un reinicio y rápida recuperación de la ventana como se espera teniendo en cuenta que se tiene un 60.72% de posibilidad que se recupere el flujo correcto de datos.

Ahora tanto para RENO como para SACK, después de presentarse el caso de ventana de w igual a 33000 bytes, para RENO como para SACK, se observa un comportamiento periódico en el tiempo del manejo de la ventana.

La probabilidad de tener un estado de espera en la transmisión es mostrada en la tabla 4 usando las ecuaciones (1) y (4) tanto para RENO como SACK.

Tabla 4. Probabilidad de timeout, w es el ancho de la ventana en número de bytes, p es la perdida de bytes. El análisis se basa solo en la presencia de bytes perdidos.

Probabilidad de rápida retransmisión y rápida recuperación			
w(bytes) P(bytes)	RENO (%)	w(bytes) P(bytes)	SACK (%)
w=63000 p=32000	100	w=63000 p=32000	100
w=33000 p=1000	7,74	w=32000 p=1000	3,08

El inicio de la transmisión tiene una pérdida de 32000 bytes con una ventana de 63000 bytes, donde la probabilidad a caer en un estado de espera es del 100%, para el funcionamiento continuo del ancho de ventana a un byte perdido y un ancho de ventana de 33000 bytes, se trabaja de forma continua con una probabilidad de 7,74% para RENO, y de 3,08% para SACK donde se logra visualizar que el método de control de ventana SACK es más robusto que RENO, solo en funcionamiento periódico (establecimiento de la ventana aproximada).

Luego de los primeros 60 s, se cambia la tasa de datos del flujo CBR, produciendo congestión en el canal de comunicación, que se refleja en cambios de los parámetros antes vistos para RENO

y SACK (tablas 5 y 6).

Se visualiza que en el cambio de tráfico en el canal, el comportamiento del control de ventana RENO, tiene una estabilidad baja dando desde 0,16%, de probabilidad de recuperación y una tendencia a llegar a 0% (produce reinicio de la ventana), luego se logra subir este comportamiento a 36,2% para trabajar de forma periódica lo cual presenta gran debilidad en este método de control de ventana RENO.

Tabla 5. Probabilidad de rápida retransmisión y recuperación luego de 60s, w es el ancho de la ventana en número de bytes y p es el número de bytes perdidos, el análisis se hace solo en presencia de paquetes perdidos, sin ellos la probabilidad se mantiene en 100 %.

Probabilidad de rápida retransmisión y rápida recuperación (60s)			
w(bytes) P(bytes)	RENO (%)	w(bytes) P(bytes)	SACK (%)
w=25000 p=1000	0,16	w=27000 p=6000	25,36
w=10000 p=9000	$4{,}79 \times 10^{-4}$	w=23000 p=2000	71,64
w=15000 p=5000	7,71		
w=15000 p=3000	36,2		

Para SACK se tiene una caída de la probabilidad al 25,36%, que refleja que fue afectado pero tiene un porcentaje mayor a recuperarse, luego en un cambio de ventana abrupto, se trabaja con un 71,64%, que muestra la robustez de este protocolo ante cambios en la red, como puede llegar a ser la congestión de la misma.

Tabla 6. Probabilidad de timeout, w es el ancho de la ventana en número de paquetes, p es número de paquetes perdidos. Se analiza solo en presencia de perdida de paquetes, en este caso se tiene 0% de posibilidad de entrar en espera.

Probabilidad de rápida retransmisión y rápida recuperación (60s)			
w(bytes) P(bytes)	RENO (%)	w(bytes) P(bytes)	SACK (%)
w=25000 p=1000	99,84	w=27000 p=6000	74,52
w=10000 p=9000	100	w=23000 p=2000	16,02
w=15000 p=5000	92,06		
w=15000 p=3000	60,2		

Para el cambio de la congestión de la red, se presenta en RENO un estado de aproximadamente 100% de caer en el estado de espera, con una demora de 10s, luego baja esta posibilidad de espera a 60,2%, donde queda en trabajo periódico. Esto demuestra una alta posibilidad de demora en el envió de tráfico, y a su vez la demora en el establecimiento de la ventana de trabajo alto. Para la práctica, estos 10s que tarda en establecer la ventana es demasiado alto y la probabilidad de caer en tiempos de espera a su vez vuelve ineficaz este método.

3.1.1. VEGAS con tráfico FTP

Los resultados obtenidos en el control de flujo VEGAS, se obtiene un comportamiento de comprobación de la red por medio de variaciones de la ventana. Para el caso se tiene valores iniciales dados por [43], con valores de α=1000 bytes, β=3000 bytes, RTT=1s (donde solo son constantes α,β y RTT es valor inicial igual a 1), para modo de inicio no lento se tiene γ=1000 bytes. En la tabla 7 se observa algunos valores de diff (calculado a partir de la ecuación(6)), donde el control realizado por VEGAS, logra encontrar su punto de equilibrio (ecuación (7)).

Se logra observar, como la ventana inicia con tamaño de 1000 bytes, luego se incrementa hasta llegar al valor de 16000 bytes, siempre censando la variable RTT, manteniendo constante el tamaño de la ventana se logra ver como RTT aumenta a valor de 0,43s y varia a 0,42s. Llegado a este estado el control trabaja de forma periódica, hasta los 60s, donde se presenta una variación en la congestión de la red.

Tabla 7. Comportamiento algoritmo de control de flujo VEGAS, variables iniciales α=1000, β=3000 y γ=1000. Estos datos son tomados de los primeros 60s donde el RTT base es de 0,34s, w es el tamaño de la ventana en número de bytes, RTT tiempo de ida y regreso, diff es la variable que censa el tráfico para VEGAS

w (bytes) RTT (s)	diff (bytes)	Cambio de W (bytes)	Causa
w=1000 RTT=1	660	w=2000	diff<1000
w=2000 RTT=0,35	571	w=3000	diff<1000
w=3000 RTT=0,35	85,7	w=4000	diff<1000
w=4000 RTT=0,35	114,3	w=5000	diff<1000
w=10000 RTT=0,35	285,7	w=11000	diff<1000
w=15000 RTT=0,36	830	w=16000	diff<1000
w=16000 RTT=0,37	1297,3	w=16000	1000<diff<3000
w=16000 RTT=0,43	3348,8	w=15000	diff>3000
w=16000 RTT=0,42	3000	w=16000	1000<diff<3000

Tabla 8. Estimativo del retardo de encolamiento de la red, la tasa de transferencia actual y cantidad de paquetes transmitiéndose por la red, con variables iniciales α=1000, β=3000 y γ=1000. Estos datos son tomados en los primeros 60s donde el RTT base es de 0,34s, w es el tamaño de la ventana en número de bytes, RTT tiempo de ida y regreso.

w (bytes) RTT (s)	Retardo cola (s)	Estimativo tasa transferencia (bytes/s)	Cantidad transmitiéndose (bytes)
w=1000 RTT=1	0,66	1000	660
w=2000 RTT=0,35	0,01	5714,285714	57,14285714
w=3000 RTT=0,35	0,01	8571,428571	85,71428571
w=4000 RTT=0,35	0,01	11428,57143	114,2857143
w=10000 RTT=0,35	0,01	28571,42857	285,7142857
w=15000 RTT=0,36	0,02	41666,66667	833,3333333
w=16000 RTT=0,37	0,03	43243,24324	1297,297297
w=16000 RTT=0,43	0,09	37209,30233	3348,837209
w=16000 RTT=0,42	0,08	38095,2381	3047,619048

El comportamiento general del protocolo (tabla 8) busca la mejor tasa de cantidad de paquetes transmitidos, presentando un tamaño de ventana constante, y a su vez asegurando un nivel de confiabilidad alto por no presentar ningún byte perdido. Para el caso 3000 bytes aproximadamente.

En la aparición de congestión en la red, a los sesenta segundos el control de flujo presenta características mostradas en la tabla 9.

Al observar la tabla se ve la regulación de la ventana ante una aparición de subida de uno de los parámetros como lo es el RTT, que sube de 0,43s a 0,56s (76% de variación), y el control de ventana hace reducción hasta llegar a estabilizarse en un valor de 11 paquetes aproximadamente y un retardo de ida y regreso de 0,43s. Vale aclarar que no se presentó ningún dato perdido.

Tabla 9. Efectos de congestión en la red, donde el RTT base es constante. Donde w es el ancho de la ventana en número de paquetes y RTT es el tiempo de ida y regreso.

w (bytes) RTT (s)	diff (bytes)	cambio de w (bytes)	causa
w=16000 RTT=0,56	6285,7	w=15000	diff>3000
w=13500 RTT=0,58	5586,2	w=12500	diff>3000
w=12750 RTT=0,59	5402,5	w=11750	diff>3000
w=12250 RTT=0,54	4537	w=11250	diff>3000
w=11250 RTT=0,51	3750	w=10250	diff>3000
w=11300 RTT=0,48	3290	w=10300	diff>3000
w=11300 RTT=0,45	2920	w=10300	1000<diff<3000
w=11300 RTT=0,43	2880	w=10300	1000<diff<3000

Tabla 10. Estimativo del retardo de encolamiento de la red, la tasa de transferencia actual y cantidad de paquetes transmitiéndose por la red. Pasados 60 segundos en la simulación propuesta.

w (bytes) RTT(s)	Retardo cola (s)	Estimativo tasa transferencia (bytes/s)	Cantidad transmitiéndose (bytes)
w=16000 RTT=0,56	0,22	28571,42857	6285,714286
w=13500 RTT=0,58	0,24	23275,86207	5586,206897
w=12750 RTT=0,59	0,25	21610,16949	5402,542373
w=12250 RTT=0,54	0,2	22685,18519	4537,037037
w=11250 RTT=0,51	0,17	22058,82353	3750
w=11300 RTT=0,48	0,14	23541,66667	3295,833333
w=11300 RTT=0,45	0,11	25111,11111	2762,222222
w=11300 RTT=0,43	0,09	26279,06977	2365,116279

La tabla 10, muestra el comportamiento de la red al aumento de la congestión de la misma, donde se observa que momentáneamente hay un aumento de paquetes enviados (aproximadamente solo 4s, mientras que esta se estabiliza), en este tiempo la cantidad de paquetes transmitidos por la red decae de 6,2 paquetes y se estabiliza a 2,5 paquetes. En comparación a los primeros 60s se ve una reducción de solo 0,5 paquetes (17%). Luego de este cambio presenta un comportamiento periódico en la red, vale resaltar que en ningún momento se tuvo paquetes perdidos.

3.2. Análisis comparativo

Teniendo en cuenta los resultados obtenidos en las simulaciones mostradas en la sección 4 se puede analizar algunas cosas interesantes.

Según la tabla 1, desde el punto de vista control de flujo la implementación de TCP que permite mantener la máxima integridad de entrega de datos sin gastar recursos extras de la red es TCP VEGAS, debido a que en todo el tiempo de la simulación no registró ninguna pérdida de paquetes, sin dejar de lado que es la implementación que permitió el mayor envío de paquetes en el mismo tiempo de simulación comparado con RENO y SACK. En las simulaciones se puede observar que una de las virtudes de VEGAS es su rápida reacción al detectar congestión en la red, rapidez que es aprovechada para redimensionar la ventana. Con los resultados se pone VEGAS a la cabeza del rendimiento desde el punto de vista de control de flujo.

Descendiendo en la escala de mejor control de flujo, se encuentra la implementación de TCP SACK, la cual para todo el tiempo de simulación solamente registró la pérdida de 35 paquetes, pero en cuanto a la cantidad de paquetes entregados es la que menos entregas logró en el tiempo de simulación, pero estos resultados muestran que SACK tiene un mejor control cuando aparecen congestiones en la red que le quitan acceso exclusivo a la red, capacidades que van por encima de las que tiene TCP RENO.

Por último se encuentra la implementación de TCP RENO, que es la que mayor cantidad de pérdidas presenta entre todas las implementaciones de TCP, pero a pesar de eso muestra que dentro de la entrega de paquetes fue capaz de entregar una cantidad mayor que SACK, pero inferior a VEGAS. Una de las debilidades que salen a la luz de RENO es que en presencia de congestión y pérdida de paquetes adicionales no es capaz de controlar rápidamente el tamaño de su ventana, produciendo mayores pérdidas de paquetes que las otras implementaciones.

Comparando todas las implementaciones de TCP cuando se emplea tráfico Telnet, se observa que el desempeño de todos es similar, entregando al final de la simulación exactamente los mismos resultados en cuanto a pérdida de paquetes y número de paquetes entregados, por eso desde el punto de vista de tráfico intermitente, todas las implementaciones dan el mismo servicio.

6. CONCLUSIONES

En el análisis en tráficos FTP contra TELNET, se encuentra que en TELNET hay gran dependencia de la capacidad de manejo de encolamiento de paquetes de la red. Los procesos matemáticos realizados para tráficos FTP, tendrá para

RENO y SACK, 0% de caer en estado de espera, y 100% de posibilidad de recuperación y rápida retransmisión esto debido a la no perdida de paquetes. Para RENO, se tienen intervalos de insuficiente tiempo para ajuste de una ventana constante.

A nivel probabilístico el funcionamiento de RENO 56% y de SACK 60,72%, de probabilidad de caer en la fase de rápida recuperación y rápida retransmisión, además del 7,74% y 3,08% respectivamente de probabilidad de caer en estado de espera, todo ello permite visualizar que en redes que no son ideales, se tiene más robustez por parte de SACK a tener una capa de transporte que no cumpla con las expectativas de los usuarios.

Tanto para RENO como para SACK, existe perdida de paquetes con los que estos se autor regulan, pero al adicionar a esto un cambio en la congestión de la red, SACK mostro más robustez en comparación a RENO, evidenciando que en redes reales, se denota conveniente el trabajo a partir de SACK.

Se manifestó un comportamiento en RENO a nivel de paquetes perdidos igual que en SACK, esto cuando la red no se congestiona, dando lugar el uso de RENO en redes de baja congestión, aunque va en contravía al sentido de ser escalable. Para la elección no es conveniente solamente revisar esta variable, se podría extender el análisis a la demora de procesamiento de cada protocolo o también costos de hardware.

Si la efectividad de los sistemas se mide por el número de paquetes perdidos, se tiene que en VEGAS no hay ningún paquete perdido (vale resaltar que está sujeto a la cantidad de paquetes que puede encolar el sistema), que ayuda a aliviar una posible congestión en la red, por no tener que retransmitir datos, o activar contadores de no envió. En este aspecto, VEGAS es el mejor algoritmo de control de ventana para capa de transporte, a nivel de no congestionar la red, una tasa de envió constante, posibilidad de reducir costos a nivel de software para detección de errores, y a nivel de hardware por tener la posibilidad de bajar costos en memorias (buffer)

En general se logra ver en tráficos TELNET, que los 3 modelos de control de ventana de capa de transporte RENO, SACK y VEGAS necesitan un espacio para el ajuste de su ventana, para los 3 casos se ve que el sistema se vuelve con memoria, y al no tener datos precedentes es incapaz de dar un sistema confiable.

Referencias Bibliográficas

[1] H. Balakrishnan, V. Padmanabhan, S. seshan, R. Katz; A comparison of mechanisms for improving TCP performance over. s.l. : Computer Science Division, Department of EECS, University of California at Berkeley, 1996.

[2] T. Dyer, R. Boppana; A comparison of TCP performance over three routing. s.l. : Computer Science Division, The Univ. of Texas at San Antonio, 2001.

[3] A. Wierman, T. Osogami, J. Olsén; A unified framework for modeling TCP-Vegas, TCP-SACK, and TCP-Reno. s.l. : School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213, 2003.

[4] J. Mo, J. La, V. Anantharam, J. Walrand; Analysis and Comparison of TCP Reno and Vegas, s.l. : INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, Vol. 3. 6346043, 2002.

[5] G. Hasegawa, K. Kurata, M. Murata; Analysis and Improvement of Fairness between TCP Reno and Vegas for deployment of TCP Vegas to the Internet. s.l. : Department of Informatics and Mathematical Science, Osaka University, 2007.

[6] B. Sikdar, S. Kalyanaraman and K. S. Vastola; Analytic models for the latency and steady-state throughput of TCP Tahoe, Reno and SACK. s.l. : Dept. of ECSE, Rensselaer Polytechnic Institute, 2002.

[7] B. Thomas; Comparison of TCP Reno and TCP Vegas via fluid approximation. s.l. : Institut National de Recherche En Informatique Et En Automatique, 1998.

[8] K. Jingmin Song, Q. Zhang, M. Sridharan; Compound TCP: A scalable and TCP-friendly congestion control for high-speed networks. s.l. : Microsoft Corporation One Microsoft way, Redmond, WA, Microsoft Research Asia Beijing, China, 2004.

[9] F. Shaojian, M. Atiquzzaman; Effect of delay spike on SCTP, TCP Reno, and eifel in a wireless mobile environment. s.l. : School of Computer Science University of Oklahoma, 2002.

[10] H. Singh, S. Singh; Energy consumption of TCP Reno, Newreno, and SACK in multi-hop wireless networks. s.l. : Department of Computer Science Portland State University. Portland, OR 97207, 2002.

[11] A. Tang, J. Wang, S. Hegde, S. Low; Equilibrium and fairness of networks shared by TCP Reno and Vegas/Fast. s.l. : California Institute of Technology, Pasadena, CA 91125, USA, 2005.

[12] K. Hari,, H.Randy H; Explicit loss notification and wireless web performance. s.l. : Computer Science Division, Department of EECS University of California at Berkeley, 1998.

[13] K. Kurata, G. Hasegawa, M. Murata; Fairness comparisons between TCP Reno and TCP Vegas for future deployment of TCP Vegas. s.l. : Department of Informatics and Mathematical Science, Osaka University, 2002.

[14] A. Lachlan, L. Tan, T. Cui, M. Zukerman; Fairness comparison of FAST TCP and TCP Vegas. s.l. : ARC Special Research Centre for Ultra-Broadband Information Networks (CUBIN),an affiliated programme of the National ICT Australia. Department of Electrical and Electronic Engineering,The University of Melbourne, Victoria 3010, Australia.

[15] D. Hegde, W. Cheng, J. Steven, H. Sanjay. FAST TCP: Motivation, Architecture, Algorithms, Performance. s.l. : Engineering & Applied Science, Caltech, 2006.

[16] M. Gerla, R. Cigno, S. Mascolo, W. Weng; Generalized window advertising for TCP congestion control. s.l. : Computer Science Department – Boelter Hall – UCLA, 405 Hilgard Ave., Los Angeles, CA, 90024, USA, Motorola, Inc., San Diego, California, USA. 2001.

[17] A. Detti, M. Listanti; Impact of segments aggregation on TCP Reno Flows in optical burst switching networks. s.l. : A. Detti is with the Electronic Dept., University of Rome "Tor Vergata", Italy, 2002.

[18] C. Parsa, J. Garcia; Improving TCP congestion control over Internets with heterogeneous transmission media. s.l. : Computer Engineering Department Baskin School of Engineering University of California Santa Cruz, California 95064, 2003.

[19] J. Padhye, V. Firoiu, D. Towsley, J. Kurose; Modeling TCP Reno performance: A simple model and its empirical validation. 1999.

[20] C. Samios, M. Vernon; Modeling the throughput of TCP Vegas. s.l. : Department of Computer Sciences, University of Wisconsin. Madison, Wisconsin 53706, 2003.

[21] A. Shaojian; Modelling TCP Reno with spurious timeouts in wireless mobile environment. s.l. : Telecommunications and Networks Research Lab School of Computer Science University of Oklahoma, Norman, OK 73019-6151, USA, 2001.

[22] J. Mo, R. La, V. Anantharam, J. Walrand; Analysis and comparison of TCP Reno and Vegas. s.l. : Departament of electrical Engineering anc computer sciences University of California at Berkeley, 1999.

[23] D. Wei, P. Cao; NS2 TCPLinux: An NS2 TCP implementation with congestion control algorithms from Linux. s.l.

: Department of Computer Science California Institute of Technology, 2006.

[24] M. Luigi, G. Saverio; Performance evaluation and comparison of westwood+, New Reno, and Vegas TCP congestion Control. s.l. : Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Italy.

[25] L. Sarma, A. Miguel; Performance of TCP over wireless networks with the snoop protocol. s.l. : Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620, U.S.A, 2001.

[26] F. Kevin; Simulation-based comparisons of Tahoe, Reno, and SACK TCP. s.l. : Lawrence Berkeley National Laboratory One Cyclotron Road, Berkeley, CA 94720, 2003.

[27] H. Shimonishi, T. Hama, T. Murase; TCP-Adaptive Reno for improving efficiency-friendliness tradeoffs of TCP congestion control algorithm. s.l. : System Platforms Laboratory, NEC Corporation, 2001.

[28] S. Joel; TCP New Vegas: Improving the performance of TCP Vegas over high latency links. s.l. : Department of Computer Science and Computer Engineering La Trobe University Bundoora VIC 3083, Australia, 2005.

[29] F. Akyildiz, G. Morabito, S. Palazzo; TCP-Peach: A new congestion control scheme for satellite IP networks. s.l. : IEEE/ACM Transactions on Networking,. Vol. 9, 2000.

[30] M. Allman, C. Hayes, H. Druse, S. Ostermann; TCP perfomance over satellite links. s.l. : Ohio University, 1997.

[31] M. Barreto, M. Belino, S. Marcelo; TCP SACK. s.l. : Instituto de Ingeniería Eléctrica – Universidad de la República 2004.

[32] K.. Srijith, L. Jacob, A. Ananda; TCP Vegas-A: Improving the performance of TCP Vegas . s.l. : Communication and Internet research Lab, Department of Computer Science, School of Computing, National University of Singapore, 2005.

[33] C. Casetti, M. Gerla, S. Mascolo, M. Sanadidi, R. Wang; TCP westwood: nd-to-End Congestion Control for Wired/Wireless Networks. s.l. : Kluwer Academic Publishers. Manufactured in The Netherlands, 2002.

[34] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, R. Wang; TCP Westwood: Bandwidth estimation for enhanced eransport over wireless links, s.l. : UCLA Computer Science Department, 2001.

[35] F. Tinnakornsrisuphap; The failure of TCP in high-performance computational grids. s.l. : Research & development in advanced network technology (RADIANT) Los Alamos National Laboratory, 2004.

[36] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi. The macroscopic behavior of the TCP congestion avoidance algorithm. s.l. : Pittburgh Supercomputing Center.

[37] A. Aggarwal, S. Savage, T. Anderson; Understanding the performance of TCP pacing. s.l. : Department of Computer Science and Engineering University of Washington, 2005.

[38] S. Nair; Improving the performance of TCP Vegas and TCP sack: Investigations and solution. s.l. : School of computing National university of Singapore, 2002.

[39] H. Zhang, Z. Bian; Evaluation of different TCP congestion control algorithms using NS-2. 2002.

[40] E. Altman; NS simulator for beginners, Universidad de los Andes, Mérida, 2003.

[41] UC Berkeley, LBL, USC/ISI and Xerox PARC, The ns Manual,the VINT project, 2010.

[42] J. Gonzales; Prácitcas de redes, Protocolo TCP, 2003.

[43] University of California "ns-default. tcl". [En línea], consultado en Marzo 15 del 2011, disponible en: http://read. pudn.com/downloads106/sourcecode/ others/438468/ns-2.31/bluehoc/mod/ns-default.tcl__.htm.