Simulación de Algoritmos de Control de Congestión en TCP bajo User Mode Linux

Damián Rattalino[1], Abel Crespo[2]
[1,2]Universidad Nacional de La Pampa - Facultad de Ingeniería
Calle 110 esquina 9, General Pico, La Pampa (CP 6360)
[2]crespoa@ing.unlpam.edu.ar

Resumen: *La experimentación con protocolos y servicios de aplicación bajo TCP/IP requieren de una costosa infraestructura de laboratorio que debe ser actualizada y/o renovada periódicamente, considerando la alta dinámica científica-tecnológica del área. Por lo general, es necesario crear topologías con múltiples redes y/o subredes pobladas de computadoras, y cuando este es el caso, la experiencia de laboratorio forzosamente debe ser grupal. Una alternativa superadora, que permite mejorar la calidad de enseñanza, es a través de la utilización de tecnologías para la virtualización de computadoras y dispositivos de interconexión, posibilitando la construcción de complejos escenarios de red sin otro requerimiento más que el de disponer de una computadora de medianos recursos. En esta dirección, el presente trabajo muestra como experimentar con algoritmos de control de congestión en TCP, utilizando topologías virtuales construidas en User Mode Linux (UML). UML es una tecnología que permite virtualizar computadoras y redes de computadoras bajo GNU/Linux.*

Palabras Claves: *Control de Congestión, TCP/IP, Simulación, User Mode Linux*

Abstract: *Experimenting with protocols and application services under TCP/IP requires a costly laboratory infrastructure which must be updated periodically, considering the high dynamics of the area and the rapid technological obsolescence of involved devices. On one hand, we must create multiple network topologies and subnets composed by tens of computers. On the other, we need to be sure our labs are available to workgroups in order to share configurations and test results. A better alternative, which improves the quality of teaching, is through the use of virtualization technologies for interconnecting computers and devices, enabling the construction of complex network scenarios with no other requirement than a middle cost server. In this direction, this paper shows how to experiment with algorithms in TCP congestion control using virtual topologies built in User Mode Linux (UML). UML is a technology that enables multiple virtual Linux kernel-based systems to run as an application within a normal Linux system.*

Keys Words: *Cogestion Control, TCP/IP, Simulation, User Mode Linux.*

INTRODUCCIÓN

Para el dictado de un curso típico de Redes de Computadoras, se requiere de las bases teóricas del área, para luego entrar a otra, en que el dictado se puede transformar en una conjugación muy estrecha entre teoría y práctica. El límite entre ambos modos de enseñanza suele ser el estudio exhaustivo de las funciones de la capa de enlace de datos respecto del modelo de referencia OSI. Luego, todo lo relativo al estudio de protocolos y servicios de aplicación de red puede abordarse desde la perspectiva simultánea antes mencionada. Mediante esta técnica, el alumno puede asimilar conceptos y reafirmarlos desde la experimentación. Si la experimentación es individual, el proceso de enseñanza mejora notablemente, pues desde la óptica docente, permite unívocamente determinar puntos

de incertidumbre, que deben ser aclarados para que el proceso de aprendizaje converja al entendimiento definitivo. Un laboratorio de experimentación basado en tecnologías para la virtualización de computadoras y dispositivos de interconexión que permita la construcción de complejas topologías de red, es la única vía para lograr la modalidad de enseñanza antes descrita. Ha de notarse que las empresas más renombradas en la producción de dispositivos relacionados a las redes de cualquier envergadura, proveen soluciones propietarias a sus clientes para la virtualización de sus productos. De esta forma, un administrador, antes de realizar modificaciones en la configuración real de una infraestructura de red, lo hace sobre una réplica virtual, y si la nueva configuración responde a sus expectativas, finalmente las aplicará en la infraestructura real.

Si bien existen múltiples opciones de fuentes abiertas (open source) respecto a tecnologías de virtualización para construcción de redes de computadoras, en el ámbito de este trabajo se utilizará User Mode Linux, de ahora en más por su acrónimo UML (Dike, 2006). UML es un núcleo Linux, modificado para ser compilado sobre GNU/Linux. Las computadoras o dispositivos virtuales se soportan sobre un núcleo Linux compilado para la arquitectura UML, que se ejecuta como un proceso de usuario sobre el núcleo Linux convencional en la computadora anfitriona. De esta manera, es posible crear redes virtuales que respondan a topologías arbitrariamente complejas. Absolutamente todo lo relativo al conjunto de protocolos TCP/IP puede ser experimentado con UML, y en esta dirección, este articulo muestra como experimentar con algoritmos de control de congestión.

El control de congestión aplicable a flujos de bytes sobre conexiones TCP es uno de los tópicos que en un curso típico de redes de computadoras se lo aborda por lo general, desde una perspectiva estrictamente teórica, y este artículo contribuye con un novedoso método de experimentación que no requiere de la utilización de complejos simuladores de eventos discretos, tal como NS y OPNET, ni de la variada cantidad de dispositivos necesitados para construir una topología real. A tal fin, se virtualizará con UML una topología de red apta para él estudió de algoritmos para el control de congestión, posteriormente se llevaran a cabo los experimentos considerando las características típicas de una infraestructura de red WAN.

Este trabajo está organizado del siguiente modo. La siguiente sección tratará sobre aspectos teóricos de algunos de los algoritmos de control de congestión escogidos para la simulación con UML, posteriormente se describirá la topología de experimentación a implementar con UML y las herramientas utilizadas para llevar a cabo el análisis experimental, luego se mostraran los resultados experimentales obtenidos y finalmente se presentaran las conclusiones del artículo.

CONTROL DE CONGESTIÓN

INTRODUCCIÓN

El termino congestión hace referencia a un fenómeno no deseado, generado en una red, o en una parte de su infraestructura, cuando distintas fuentes inyectan un volumen de tráfico superior al que los dispositivos de conmutación de paquetes pueden procesar. En octubre de 1986 se evidenció por primera vez un colapso debido al fenómeno de congestión y a partir de ese evento, la situación se agravaría en la medida que más computadoras se conectaban a Internet.

TCP TAHOE

En 1988 y a raíz de la problemática descrita, se propuso (Jacobson, 1988), un mecanismo de control

de congestión basado en el concepto de ventanas de congestión, cwnd, y en la definición de tres algoritmos: arranque lento (slow start), prevención de congestión (congestion avoindace) y retransmisión rápida (fast retransmit).

A cada extremo en una conexión TCP se le asigna una ventana de congestión, cwnd, indicativa del número de segmentos que puede enviar a la red sin esperar por reconocimientos explícitos desde el receptor (ACK`s). Al inicio, en la fase de operación de arranque lento (slow start), cwnd se configura en uno o dos segmentos de máximo tamaño (MSS) y su valor se incrementará en un segmento por cada reconocimiento recibido desde su par receptor (ACK). Durante la fase operativa de arranque lento, cwnd experimentará un crecimiento exponencial, tal como se ilustra en la Fig. 1. Cuando cwnd alcanza el valor de la variable "umbral de arranque lento" o slow start threhold (ssthresh), el mecanismo de control de congestión conmuta al modo congestion avoindance y cwnd se incrementa linealmente a razón de un segmento por grupo de reconocimientos (ACKs), que se corresponden con el valor actual de la ventana de congestión (Fig. 1).

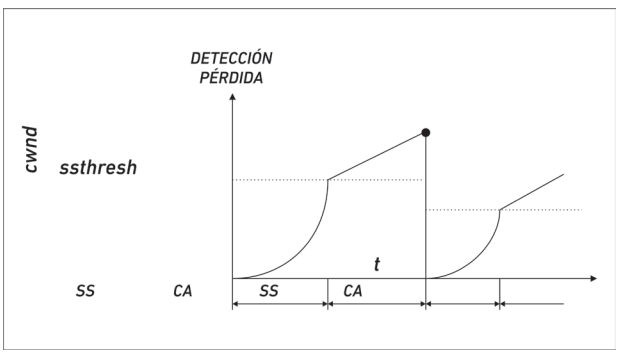

Fig. 1 - TCP Tahoe.

Se abandona el estado de congestion avoindace cuando el emisor TCP detecta una pérdida por expiración de timeout o por la recepción consecutiva de tres reconocimientos duplicados (ACKs). En ambos casos el emisor reconfigura la variable ssthrehold al valor cwnd/2, retransmite el segmento perdido (fast retransmit), y conmuta a la fase de operación slow start (SS) actualizando cwnd=1.

TCP RENO Y TCP NEWRENO

TCP Reno surgió en 1990 como el sucesor de TCP Tahoe y fue implementado por primera vez en el sistema operativo 4.3BSD Unix. Si bien mantuvo las características de su antecesor, incorporó el algoritmo de recuperación rápida o fast recovery (Jacobson, 1990). El algoritmo de recuperación rápida (FR) evita el ingreso a la fase de arranque lento (SS), cuando se detectan tres reconocimientos duplicados (ACKs) en la fase prevención de congestión (CA) y bajo esta circunstancia el emisor ejecuta las siguientes acciones: reconfigura la variable ssthresh al valor cwnd/2, retransmite el segmento perdido y actualiza la ventana de congestión a cwnd=ssthresh+3, finalmente abandona la fase de recuperación rápida para regresar a la fase de prevención de congestión (CA).

Una de las vulnerabilidades de TCP Reno se manifiesta cuando en la fase fast recovery (FR) se origina

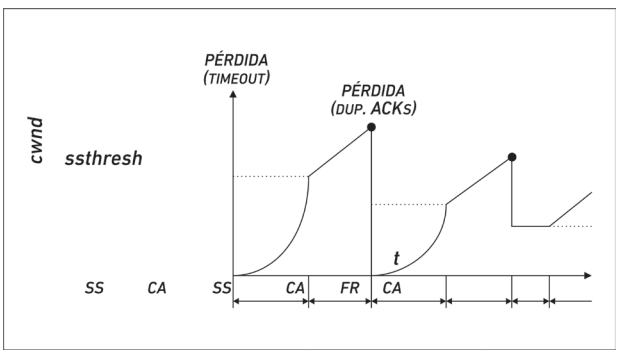

Fig. 2 - Dinámica de cwnd en Reno.

un evento de perdida de múltiples segmentos para el valor actual de cwnd. Bajo esta circunstancia TCP Reno alternará su operación entre las fases CA/FR, y provocará una reducción sucesiva del valor de la variable umbral de arranque lento (ssthresh), y

de la ventana de congestión (cwnd). Como consecuencia se produce una importante disminución en el desempeño (thrughput) de la conexión TCP.

Atentos a la problemática descrita, en (Floyd and Henderson, 1999) y (Floyd et al., 2004) se introdujo un simple refinamiento al algoritmo fast recovery (FR). En la fase congestion avoindance (CA), al recibir tres ACK´s duplicados, ingresa a la fase fast recovery, pero a diferencia de TCP Reno, no abandona la fase fast recovery hasta que se produzca la confirmación (acknowledged) de todos los segmentos de la ventana de congestión (cwnd). Más formalmente, TCP NewReno utiliza una variable de estado adicional para registrar el número de secuencia del último segmento TCP enviado antes de entrar a la fase fast recovery.

TCP NewReno resuelve el problema de bajo desempeño ante eventos de múltiples pérdidas en la ventana de congestión (cwnd) pero como contrapartida emplea un tiempo excesivo en el proceso de recuperación, pues le toma un Round Trip Time (RTT) recuperar cada uno de los segmentos perdidos en la ventana de congestión.

Diversas estrategias se formularon para mejorar el desempeño de TCP NewReno en la fase de recuperación rápida (fast recovery) y una excelente referencia que investiga exhaustivamente numerosas propuestas puede encontrarse en (Afanasyev et al., 2010).

TCP BIC

El desempeño del protocolo TCP basado en TCP Reno y TCP NewReno no es satisfactorio en redes de alta velocidad, con elevados RTT entre los host que se comunican, y el factor limitante es el comportamiento conservativo de ajuste de la ventana de congestión que gobierna la tasa de transmisión del emisor (Ha et al., 2006). Por tal razón surgieron una serie de propuestas basadas en reformular la vía con la cual TCP adapta la ventana de congestión.

Un intento exitoso para crear un control de congestión que pueda escalar en redes con alto valor BDP (bandwidth delay product) fue propuesto en (Xu et al., 2004). BIC (Binary Increase Congestión control) extiende TCP NewReno con una fase adicional denominada de convergencia rápida (Rapid Convergence or RC). La fase RC, tiene como objetivo descubrir mediante una rápida búsqueda binaria, el valor óptimo de la ventana de congestión en función de los recursos actuales de la red. El proceso de búsqueda binaria es como se ilustra en la Fig. 3.

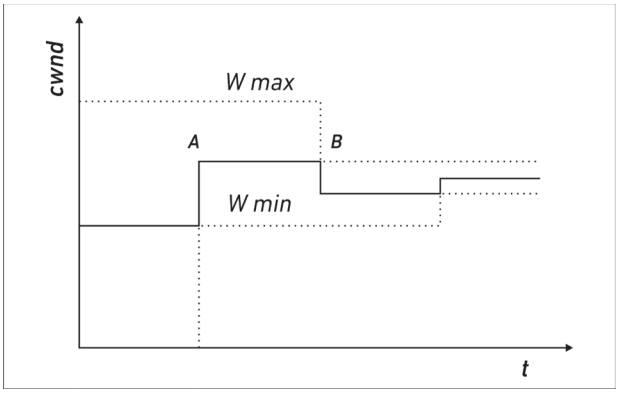

Fig. 3 - Búsqueda binaria de cwnd.

Inicialmente se establecen las variables Wmin=1 y Wmax en un valor arbitrariamente alto. Si la red entrega correctamente todos los segmentos emitidos (el emisor recibe todos los ACKs correspondientes al último RTT), la ventana de congestión se actualiza al valor promedio entre Wmax y Wmin (A en la Fig. 3) y Wmin adquiere el tamaño de la ventana de congestión previa. Cuando se detecta una pérdida, BIC reduce Wmax al valor actual de la ventana de congestión (B en la Fig. 3) y como en TCP NewReno ingresa a la fase de recuperación rápida (FR).

Para incrementar la tasa de convergencia, en ambientes de bajo nivel de pérdidas, BIC reduce el coeficiente de decremento multiplicativo

desde 0.5 a 0.125, para lograr mayor velocidad de respuesta. Las características del algoritmo de búsqueda binaria se traducen en tiempos de convergencia muy bajos que pueden provocar degradación en conexiones TCP. Si la ventana de congestión se incrementa drásticamente, un grupo significativo de segmentos pueden perderse. Por esta razón BIC implementa un algoritmo de arranque lento restringido (Floyd, 2004) que limita el incremento en cada paso de slow start a un valor máximo de 100 segmentos. En la misma dirección, BIC impone límites al crecimiento de la ventana de congestión en la fase de convergencia rápida cuando el rango de búsqueda binaria es muy amplio, y no permite incrementar la ventana de congestión más allá de un valor predefinido Smax.

En el caso opuesto, cuando el rango de búsqueda binaria es pequeño, y por tanto cercano al valor óptimo, BIC define el valor constante Smin como un quantum de incremento de la ventana de congestión. Cuando el valor actual de la ventana de congestión alcanza el valor optimo, o lo excede, BIC conmuta a la fase de arranque lento restringido, no limitado por el valor umbral de ssthresh (slow start threshold). El propósito de esta acción es descubrir un nuevo límite superior para luego reiniciar la fase de búsqueda binaria. La Fig. 4 ilustra la dinámica de la ventana de congestión de una conexión para TCP BIC.

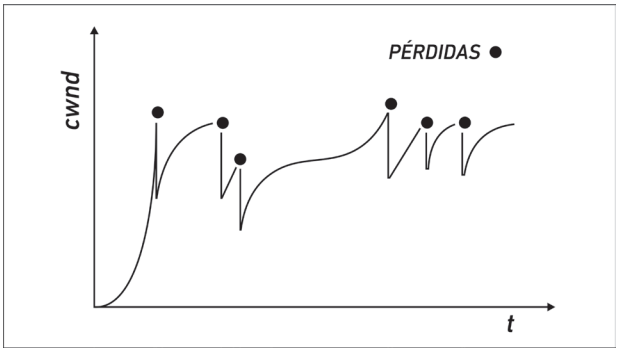

Fig. 4 - Dinámica de cwnd en BIC.

TCP CUBIC

En el artículo (Rhee and Xu, 2005) se comprobó que TCP BIC presenta problemas cuando dos o más flujos de bytes con diferencias significativas en sus RTT compiten por el ancho de banda en un enlace de características restrictivas (bottleneck link). Concretamente se demostró que flujos de bytes con pequeños RTT escalan rápidamente su ventana de congestión al valor óptimo y consumen los recursos de la red, en detrimento de flujos con RTT mayores y como contrapartida propusieron el mecanismo de control de congestión Cubic, que constituye una mejora de TCP BIC e implementa una función para el crecimiento de la ventana de congestión en forma independiente del RTT para cada conexión en particular.

TCP Cubic define una ventana de congestión (w) como una función cubica del tiempo transcurrido desde el último evento de congestión (Δ):

$$w = C\left(\Delta - \sqrt[3]{\frac{\beta * w_{max}}{C}}\right)^3 + w_{max}$$

Donde C es una constante predefinida, β es un coeficiente multiplicativo decreciente en la fase de recuperación rápida, y w_{max} es el tamaño de la ventana de congestión inmediatamente antes de que se detecte una pérdida. La función presenta un rápido crecimiento cuando w es pequeña

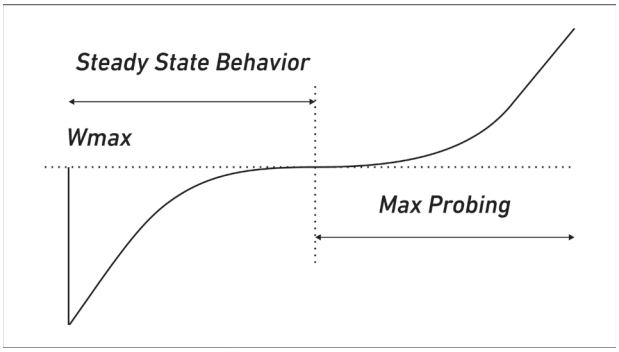

Fig. 5 - Descripción función cubica

respecto de w_{max}, y es muy conservativa cuando w se aproxima al valor de w_{max} tal como se ilustra en la Fig. 5.

La Fig. 6 muestra la dinámica de la ventana de congestión de TCP Cubic. De la observación se puede deducir que, durante el paso inicial "1" se desconoce w_{max} y para descubrir su valor se emplea la sección derecha de la función cubica en la Fig. 5 (Max Probing), esta etapa finaliza al detectarse una perdida. A continuación, en la fase 2, se emplea la sección izquierda de la función cubica, que implica una tasa de crecimiento en la ventana de congestión menor al de la fase 1. La etapa 3 incluye ambas etapas de la función cubica y finaliza cuando se produce una pérdida; y en este caso particular se actualiza el valor umbral de w_{max}.

Por último, TCP Cubic es el mecanismo de control de congestión más utilizado desde que fue incluido por defecto en las distintas versiones de núcleos Linux desde 2006.

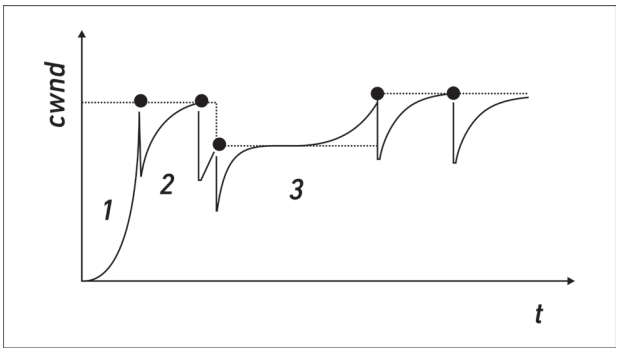

Fig. 6 - Dinámica de cwnd en TCP Cubic.

ANÁLISIS EXPERIMENTAL

Es importante destacar que el objetivo de la experimentación no está dirigido a realizar valoraciones sobre medidas de desempeño entre distintos algoritmos de control de congestión. El propósito central del artículo es convalidar un novedoso e inédito método, para experimentar con algoritmos de control de congestión, basado en la construcción de topologías virtuales utilizando User Mode Linux.

El método propuesto es de fácil e inmediata implementación, y es posible simular el comportamiento de enlaces WAN con sus características inherentes (retardos, jitter, tasa de pérdidas y restricciones de ancho de banda).

Permite ahorrar tiempo en la obtención de resultados y es apto para obtener medidas de desempeño para cualquiera de los algoritmos de control de congestión incluidos como módulos en las distintas versiones de núcleos Linux.

A continuación, en la próxima sección, se presentará el laboratorio de experimentación a implementar bajo User Mode Linux, se describirán los elementos constitutivos de la topología seleccionada y se expondrán los objetivos finales de la experimentación. Posteriormente se abordaran los principales aspectos de configuración del núcleo Linux en la computadora anfitriona, y del núcleo Linux en los dispositivos y computadoras virtualizados bajo UML.

TOPOLOGÍA UML

La Fig. 7 ilustra la topología a implementar en UML y aunque existe una extensa variedad de literatura sobre como ejecutar el escenario de red virtual, la referencia obligada es la del autor de la tecnología (Dike, 2006).

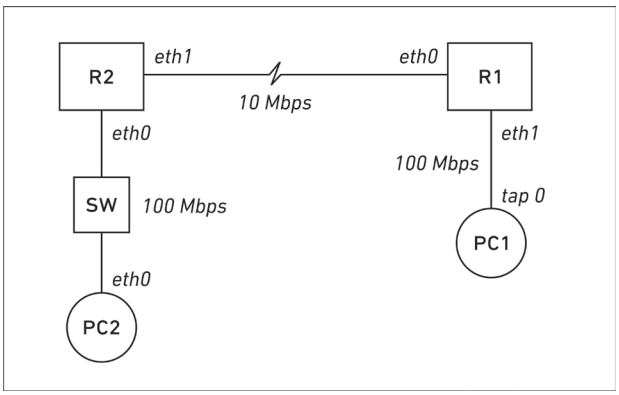

Fig. 7 - Topología virtual en UML.

Los dispositivos de conmutación de paquetes R1 y R2 (routers), la computadora PC2 y el dispositivo de conmutación de tramas, SW, constituyen componentes virtuales UML, soportados por el sistema operativo Linux en el host anfitrión PC1. La vinculación entre los dispositivos virtuales y el host anfitrión (PC1) se define durante la creación de la interfaz de red eth1 en R1, que al mismo tiempo genera una interfaz TAP en el host anfitrión PC1.

Una interfaz TAP es un enlace virtual punto a punto entre dispositivos Ethernet, que en vez de recibir tramas desde el medio físico, lo hace desde un programa en espacio de usuario; y en vez de transmitir tramas al medio físico, las envía al mismo programa en espacio de usuario. Es importante destacar que el núcleo Linux del host anfitrión debe tener habilitado el modulo TUN/TAP Support en la sección de configuración, soporte de dispositivos de red (Network Device Support).

Antes de llevar a cabo el trabajo experimental se deben realizar una serie de tareas que incluyen, la configuración de la capacidad de los enlaces tal como lo expresa la Fig. 7, la utilización de herramientas de software para simular el RTT de una red de área amplia (WAN), la selección de un generador de tráfico TCP para establecer una conexión entre los hosts extremos de la topología, y repetir la experiencia con los algoritmos de control de congestión TCP NewReno, TCP BIC y TCP Cubic.

CONSIDERACIONES DE LA CONFIGURACIÓN

Tal como se lo anticipó en la primera sección de este artículo, User Mode Linux requiere de un núcleo Linux, configurado y compilado bajo herramientas GNU, para la arquitectura UML. Este núcleo, servirá de soporte para computadoras y dispositivos virtuales y se ejecutará como un programa en el plano de usuario del sistema operativo de la computadora anfitriona que posee su propio núcleo Linux.

Para experimentar con algoritmos de control de congestión se necesitará acceder a un conjunto de variables en el núcleo Linux, y la vía para realizarlo es a través de tcpprobe, un módulo que posibilita el acceso a las variables de estado en una conexión TCP. El mecanismo opera insertando una sonda de prueba no invasiva (kprobe) en el buffer de recepción tcp_recv. Mediante este procedimiento se puede capturar en forma dinámica todas las variables necesarias de una conexión TCP, para cualquier algoritmo de control de congestión.

La tecnología de virtualización UML inhabilita el módulo tcpprobe y esto obliga a que el emisor TCP en la conexión entre los hosts extremos de la Fig. 7 sea obligatoriamente PC1, que se soporta sobre el núcleo Linux del host anfitrión. En síntesis, el núcleo Linux del host anfitrión (PC1) debe ser compilado con el módulo tcp_probe habilitado.

Para formar tráfico (traffic shaping) en el ambiente virtual, y limitar el ancho de banda de acuerdo a los valores descritos en la Fig. 7, es necesario habilitar en ambos núcleos Linux, en la sección de configuración “QoS and/or fair queueing” el módulo TBF (Token Bucket Filter). TBF es una sencilla disciplina de filas, que no distingue entre clases de tráfico (clasless) y que consiste de un buffer al que ingresan piezas virtuales de información o tokens, a una tasa constante o token rate. Tal como se ilustra en la Fig. 8, la salida de un token desde el bucket habilita la partida de un segmento TCP. Fijando la tasa de tokens se controla el ancho de banda de cada uno de los enlaces descritos en la Fig. 7.

Para emular el comportamiento de un enlace WAN en el ambiente virtual, es necesario habilitar en el núcleo Linux (soporte de los dispositivos virtuales), la opción Network Emulator o NetEm (Hemminger, 2005). Para configurar NetEm se necesita del comando, tc (traffic control), que es parte del paquete de software iproute2. Aunque NetEm posibilita una gran variedad de opciones

respecto de su funcionalidad, en el marco de este artículo sólo se lo utilizará para generar retardos y deviaciones del retardo (jitter), para replicar en el ambiente virtualizado las condiciones de un enlace WAN cuyas características topológicas son similares a las de la Fig. 7 y sus características temporales son: RTT promedio 50ms con ± 6 ms de jitter.

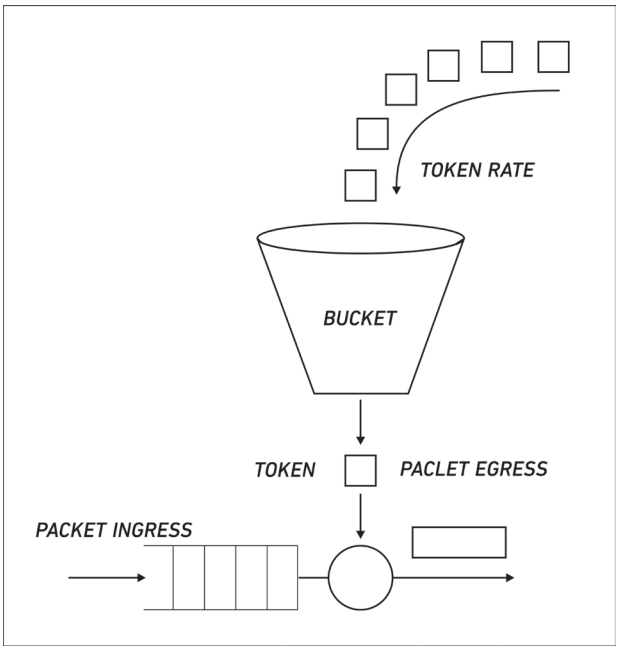

Fig. 6 - Dinámica de cwnd en TCP Cubic.

El objetivo perseguido es replicar en el ambiente virtual las condiciones de una red de área amplia (WAN). No se inducirán pérdidas mediante NetTem, pues se pretende que los algoritmos de control de congestión funcionen en base a la limitación originada por la restricción del ancho de banda del enlace WAN de 10 Mbps, considerando que el medio físico utilizado es fibra óptica y el nivel de pérdidas de segmentos es despreciable.

Un aspecto a tener en cuenta para lograr granularidad en los retardos introducidos mediante NetEm es relativo a la opción de configuración CONFIG_HZ en el núcleo Linux de los dispositivos virtuales. Esta opción determina la frecuencia con que el núcleo Linux atiende a los eventos generados por procesos en el plano de usuario. El valor por defecto de CONFIG_HZ en núcleos compilados bajo GNU para la arquitectura UML es 100 Hz, y sólo se puede modificar mediante la aplicación del código referenciado en (Dike, 2008). Si se busca granularidad en el orden de 1 ms, el valor de CONFIG_HZ debe ser modificado a 1000 Hz.

Una cuestión relativa a los dispositivos de conmutación de tramas (switchs) utilizados para construir topologías virtuales, es que UML puede emplear diversas herramientas de software para implementarlos, entre ellas, uml_switch y vde_switch. En el trabajo experimental se utilizó vde_switch al comprobar que no presentaba pérdidas de tramas en condiciones de tráfico intenso. Por el contrario la utilidad uml_switch ante idénticas circunstancias presentó pérdidas de tramas a nivel de capa de enlace de datos y este factor fue un elemento limitante en el comportamiento de los algoritmos de control de congestión experimentados.

Para generar tráfico entre ambos extremos de la topología de la Fig. 7 se utilizó la herramienta de software Iperf, con PC1 como emisor y PC2 como el receptor de segmentos TCP.

En la próxima sección se presentaran las generalidades relativas al hardware y al sistema operativo utilizado en la computadora anfitriona. Posteriormente se describirán las características de software de las computadoras y dispositivos virtuales; y finalmente se concluirá la sección mostrando los resultados de las simulaciones efectuadas para los algoritmos de control de congestión considerados.

RESULTADOS EXPERIMENTALES

Las Fig. 9, Fig. 10 y. Fig 11 ilustran los resultados de la simulación para los algoritmos experimentados bajo UML (New Reno, TCP BIC, y TCP Cubic).

Las pruebas experimentales para los algoritmos seleccionados, se realizaron en una computadora

(host anfitrión) con las siguientes características; procesador AMD (FX) - 8128 de 8 núcleos con 8 GByte de memoria RAM, ejecutando un sistema operativo GNU/Linux Ubuntu v6 basado en la versión del núcleo Linux 2.6.32.28.

Las computadoras y dispositivos virtuales UML, se ejecutaron sobre un sistema operativo GNU/Linux Slackware versión 12 soportado por la versión de núcleo Linux 3.2.59, y se asignó 1 GB de memoria para cada dispositivo virtual.

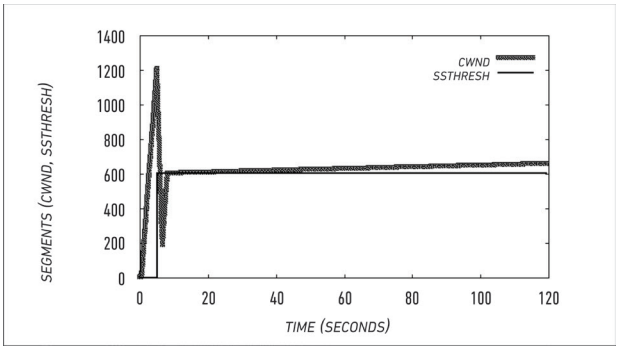

Fig. 9 - Simulación New Reno.

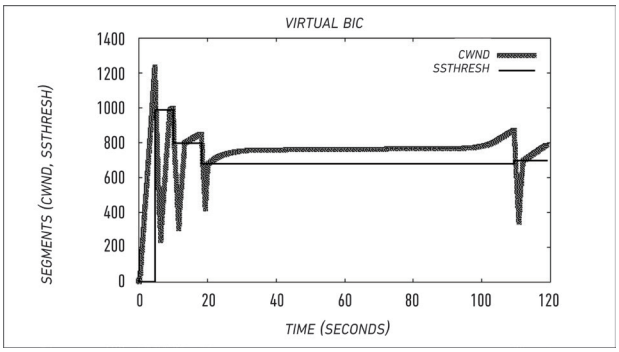

Fig. 10 - Simulación TCP BIC.

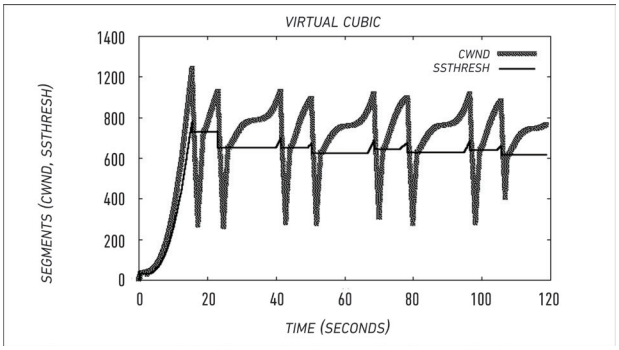

Fig. 11 - Simulación TCP Cubic.

Las simulaciones se realizaron, en un todo de acuerdo con la topología ilustrada en la Fig. 7, considerando un RTT de 50 ms, un jitter de ± 6 ms, sin introducir pérdidas ni duplicaciones de segmentos. Los segmentos generados por Iperf son de máxima longitud (MSS) para un MTU de Ethernet de 1500 bytes.

La Fig. 9 muestra los resultados de la simulación para el algoritmo de control de congestión TCP New Reno.

A diferencia de lo analizado en la segunda sección del artículo, el valor de inicial de la variable umbral de arranque lento es elevado (ssthresh) y la ventana de congestión crece exponencialmente en la fase slow start hasta que se detecta la pérdida de un segmento por la recepción de tres ACKs duplicados, a continuación se actualiza el valor de variable ssthresh, TCP NewReno reingresa a la fase de arranque lento (slow start) y aproximadamente a los 10 s el crecimiento de la ventana de congestión (cwnd) alcanza el valor de ssthresh y conmuta a la fase de prevención de congestión (CA). En esta fase de operación cwnd se incrementa de acuerdo a lo expresado en la segunda sección de este artículo y en ausencia de perdidas permanece en CA.

La Fig. 10 ilustra el resultado de la simulación para TCP BIC y se puede observar que el algoritmo tiene una fase de descubrimiento rápido al inicio, para detectar la capacidad de la red para la conexión. La variable ssthresh se corresponde con el parámetro W_min estudiado y descrito en la segunda sección de este artículo, y su dinámica es de rápida convergencia, mostrando su naturaleza de búsqueda binaria.

El comportamiento dinámico del algoritmo de control de congestión TCP CUBIC se muestra en la Fig. 11, la simulación muestra una etapa inicial de descubrimiento del límite del ancho de banda de la red, al que se aproxima utilizando la parte derecha de la función cubica que sustenta el funcionamiento del algoritmo. Una vez que se produce una condición de pérdida, el algoritmo disminuye su ventana de congestión (cwnd) y se aproxima a ssthresh utilizando

la parte izquierda de la función cubica, tal como fue tratado en la segunda sección de este artículo.

CONCLUSIONES

A pesar de la extensa variedad de artículos relativos al tópico control de congestión, no existen al respecto antecedentes de implementaciones ejecutadas íntegramente en ambientes virtualizados. Ello significa que el método de experimentación basado en User Mode Linux, para la simulación de algoritmos de control de congestión es inédito, y de gran utilidad para el estudio, análisis, y comparación de cualquiera de los algoritmos implementados en las distintas versiones de núcleos Linux.

El método propuesto es de fácil e inmediata implementación, y puede simular el comportamiento de enlaces WAN con sus características inherentes (retardos, desviaciones del retardo (jitter), tasa de pérdidas de segmentos y restricciones del ancho de banda). Permite ahorrar tiempo en la obtención de resultados, es apto para obtener medidas de desempeño y no requiere de la utilización de complejos simuladores de eventos discretos, ni de la variada cantidad de dispositivos necesitados para construir una topología real; solo se necesita disponer de una computadora de medianos recursos para obtener idénticos resultados. Los resultados obtenidos vía simulación bajo UML para los mecanismos de control de congestión TCP NewReno, TCP BIC y TCP CUBIC, se corresponden exactamente con los principios teóricos que rigen la dinámica de los algoritmos tratados en este artículo.

AGRADECIMIENTOS

Un agradecimiento especial al Ingeniero Ricardo Furch, y al A.P. Santiago Nicolau, por la lectura y las sugerencias oportunas que contribuyeron a enriquecer el presente artículo.

REFERENCIAS

Dike, "User Mode Linux", First Edition, Prentice Hall, USA, 121-166, (2006).

Jacobson, "Congestion Avoidance and Control", Computer Communication Review; 18 (4), 314-329, (1988).

Jacobson, "Modified TCP Congestion Avoidance Algorithm", end2end-interest mailing list, (1990).

Floyd and Henderson, "The New Reno Modification to TCP's Fast Recovery Algorithm"; RFC 2582, (1999).

Floyd, Henderson, and Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 3782; (2004).

Afanasyev, Tilley, Reiher, and Kleinrock, "Host-to-Host Congestion Control for TCP", IEEE Communications Surveys & Tutorials; 12 (3), 304-342, (2010).

Ha, Kim, Le, Rhee, and Xu, "A Step Toward Realistic Evaluation of High-Speed TCP Protocols", Proceedings of the Fourth International Workshop on Protocols for Fast Long-Distance Networks; (2006).

Xu, Harfoush, and Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks", Proceedings of IEEE INFOCOMM; 4, 2514-2524, (2004).

Floyd, "Limited Slow-Start for TCP with Large Congestion Windows", RFC 3742; (2004).

Rhee, and Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant", Proceedings of the Third International Workshop on Protocols for Fast Long-Distance Networks; (2005).

Hemminger, "Network Emulation with NetEm", Proceedings of the 6th Australia's National Linux Conference; (2005).

Dike, "UML: Add Back CONFIG_HZ", http://ehc.ac/p/mrvopensource/linux-ppc-2.6/ci/7281ff-952c7b3cbefb14847460e5fe73a2d240e4/, (2008).

Fecha de Presentación: 15/05/15
Fecha de Aprobación: 08/06/15