Universidad Nacional de La Plata
Facultad de Informática

Tesis presentada para obtener el grado de
Magister en Redes de Datos

"Análisis del rendimiento del protocolo TCP en redes de acceso Wireless"

Autor
Ing. Diego R. Rodriguez Herlein

Directores
Ing. Luis A. Marrone
Ing. (Mg) Carlos A. Talay

La Plata, Febrero 2020

Agradecimientos

Quiero agradecer al Ing. Luis Marrone por su apoyo, el tiempo dedicado, los importantes aportes y sugerencias, que contribuyeron no solamente en el desarrollo de esta tesis, sino también en mi actividad como docente investigador. Quiero expresarle mi agradecimiento de forma muy especial por mi formación de postgrado en esta área de conocimiento, y mi admiración por su labor docente.

A Carlos Talay, quien siempre creyó que esto era posible, por su disponibilidad y paciencia que hizo que nuestras acaloradas discusiones se transformaran en aportes valiosos tanto a nivel científico como personal.

A mis compañeros del grupo de investigación del Instituto de Tecnología Aplicada (ITA-UARG), a Claudia González por su apoyo incondicional y, muy especialmente, a los alumnos Luz Almada y Franco Trinidad por la constante predisposición y sus invalorables aportes.

A la Universidad Nacional de La Plata (UNLP), por mi formación profesional y la Universidad Nacional de la Patagonia Austral (UNPA), por darme el espacio para desarrollarme como docente e investigador.

A mis padres, que, con el ejemplo, siempre me enseñaron el camino del esfuerzo y el crecimiento personal. A mi mujer Raquel y a mi hija Karen por ser el constante apoyo en ese camino.

Esta tesis no hubiese sido posible sin la cooperación desinteresada de todas y cada una de las personas que me acompañaron en el recorrido de este trabajo. A todas ellas les quiero hacer llegar mi más profundo agradecimiento.

Río Gallegos

Diciembre de 2019

Resumen

El presente trabajo tiene como objetivo comparar distintas variantes del protocolo TCP en WLAN. Dentro del espectro de variantes, se pone especial énfasis en las que mantienen la filosofía extremo a extremo de TCP, lo que implica que no recibe ningún tipo de información explicita de la red.

Para conseguir este objetivo, se utiliza el simulador de eventos discretos NS-2, con el propósito de realizar modelos simples que permitieran observar las estrategias de los diferentes controles de congestión implementados en las distintas variantes del protocolo TCP. Los modelos utilizados intentan recrear redes de acceso inalámbricas y escenarios donde la ruta de un extremo al otro incluye distintos tipos de redes y enlaces. Estas redes heterogéneas están conformadas por enlaces tanto cableados como inalámbricos. Por lo que resulta relevante analizar el comportamiento de algunas de las variantes del protocolo en estos modelos, concebidas para distintos tipos de escenarios.

En las simulaciones se recrearon eventos frecuentes de los enlaces inalámbricos, como las desconexiones de distintos tiempos de duración y los errores en ráfaga de longitudes diferentes. Además, se definieron algunas métricas que permiten la comparación de los diversos mecanismos de control de congestión de las variantes de TCP ensayadas y se analizó cómo es la competencia por la utilización del ancho de banda (equidad) entre distintas variantes de TCP, cuando comparten un mismo canal de transmisión de datos.

PALABRAS CLAVE: TCP, WLAN, 802.11, Control de Congestión, Rendimiento, Equidad

Abstract

This work aims to compare different TCP variants in WLAN. Among these variants, a special emphasis is put on those that use the end-to-end philosophy, meaning that they don't receive any kind of explicit information from the network.

To accomplish this aim, the discrete event simulator NS-2 is used to implement simple models and study the congestion control strategies implemented by different TCP variants. These models attempt to recreate wireless access networks, in which the path from one end to another includes different kinds of networks and links. Since these heterogeneous networks are conformed by wired and wireless links, it's relevant to analyze the behavior of some TCP variants, conceived for different kind of scenarios.

The simulations recreated certain events, typical of wireless links, such as disconnections of varying duration and burst errors of different length. Some metrics were defined in order to compared the different kinds of congestion control used. Further, the contest for bandwidth usage (fairness) on a transmission channel shared by different TCP variants was analyzed.

KEY WORDS: TCP, WLAN, 802.11, Congestion Control, Performance, Fairness

Contenido

Acrónimos

3G	Third Generation Cellular Networks
3GPP	Third Generation Partnership Project
4G	Fourth Generation Cellular Networks
ACK	Acknowledgment
AIAD	Additive Increase Additive Decrease
AIMD	Additive Increase Multiplicative Decrease
AP	Access Point
ARPA	Advance Research Projects Agency
ARQ	Automatic Repeat reQuest
backbone	Nivel superior en una red jerárquica.
backoff	Retardo de retransmisión, normalmente de valor aleatorio
BDP	bandwidth-delay product
BER	Bit Error Rate
BS	Base Station
BSD	Berkeley Software Distribution
buffer	Área de almacenamiento para manejar datos temporales
BW	Bandwidth (ancho de banda)
CA	Congestion Avoidance
CCK	Complementary Code Keying
CRC	Cyclic Redundancy Check.
CSMA	Carrier Sense Multiple Access
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detect
CTS	Clear To Send
CWND	Congestion Window
CWR	Congestion Window Reduced
delay	retardo, latencia
DF	Don't Fragment flag
DUPACK	Duplicate Acknowledgment
ECE	ECN-Echo
ECN	Explicit Congestion Notification
ELN	Explicit Loss Notification
EOF	End Of File
EWMA	Exponential Weighted Moving Average

FEC	Forward Error Correction
FH	Fix Host
FI	Fairness Index
FIN	Finish Flag
FTP	File Transfer Protocol
GEO	Geostationary Earth Orbit.
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
HEO	Highly Elliptic Orbit
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
EDCA	Enhanced Distributed Channel Access
IEEE	Institute for Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPPM	IP Performance Metrics
IPSEC	IP Security
ISM	Industrial Scientifical Medical
ISN	Initial Sequence Numbers
ISO	International Organization for Standardization
ITU	International Telecommunication Union
LAN	Local Area Network
LEO	Low Earth Orbit
LL	Link Layer
LLC	Logical Link Control
LTE	Line Terminating Equipment
MAC	Medium Access Control
MAN	Metropolitan Area Network
MANET	Mobile Ad hoc Network
MEO	Medium Earth Orbit
MH	Mobile Host
MIAD	Multiplicative Increase Additive Decrease
MIMD	Multiplicative Increase Multiplicative Decrease
MIMO	Multiple Input and Multiple Output
MSL	Maximum Segment Lifetime
MSS	Maximum Segment Size
MTU	Maximum Transfer Unit
NACK	Negative Acknowledgement
NAM	Network AniMator
NAT	Network Address Translator
NAV	Network Allocation Vector
NS-2	Network Simulator version 2
OFDM	Orthogonal Frequency Division Modulation
OMNeT++	Objective Modular Network Testbed in C++
OPNET	Optimized Network Engineering Tool
OSI	Open System Interconnection
OTCL	Object Oriented Tool Command Language
OWD	One-Way Delay
PAN	Personal Area Network
PDR	Packet Delivery Ratio
PDU	Protocol Data Unit
PHY	PHYsical
PLR	Packet Loos Ratio

PMD	Physical Medium Dependent
PMTUD	Path MTU Discovery
PSH	Push Flag
PSTN	Public Switched Telephone Network
QAM	Quadrature Amplitude Modulation
QUIC	Quick UDP Internet Connections
RED	Random Early Detection
RFC	Request for Comments (IETF).
RST	Reset Flag
RTO	Retransmission TimeOut
RTD	Round Trip Delay
RTT	Round-Trip Time
RTTVAR	Round Trip Time Variation
RWND	TCP Receiver Window
SACK	Selective Acknowledgement
SMSS	Send Máximum Segment Size
SRTT	Smoothed RTT estimator
SS	Slow Start
SSTHRESH	Slow Start Threshold
STA	Station
SWS	Silly Window Syndrome
SYN	SYNchronization
TCL	Tool Command Language
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
UMTS	Universal Mobile Telecommunications System
URG	Urgent Pointer Flag, TCP header
VANET	Vehicular Ad-hoc Network
VoIP	Voice over IP
WAN	Wide Area Network
WiFi	Wireless Fidelity
WLAN	Wireless Local Area Networks
WMN	Wireless Mesh Network
WSN	Wireless Sensor Network

Introducción

El presente trabajo está dedicado al análisis del desempeño del protocolo TCP en accesos inalámbricos. Por esta razón, se estudian las deficiencias que este protocolo manifiesta al ser utilizado en WLAN, como así también las sucesivas modificaciones, que dieron origen a variantes del protocolo buscando mejorar su rendimiento.

En distintos estudios, se observó que en las redes Wi-Fi, existe un problema de eficiencia en la transmisión y de equidad en el reparto de ancho de banda, especialmente cuando hay estaciones que transmiten a distintas velocidades o caminos con distintos valores de RTT. Por lo tanto, resulta de gran interés el análisis de estos fenómenos, y el ensayo de distintas propuestas de mecanismos de control que mejoren su desempeño.

TCP ha sido utilizado sobre todo tipo de redes, y para lograr esa flexibilidad, ha tenido que renunciar a conseguir un rendimiento óptimo en redes de muy alta velocidad, en redes con valores altos del producto de ancho de banda y latencia, y en medios con valores de pérdidas altas como los entornos inalámbricos. Durante los últimos años, se han realizado distintas propuestas para mejorar el protocolo y adecuarlo a estos nuevos ambientes. Dentro de este espectro de propuestas de soluciones, este trabajo sólo contempla los esquemas de extremo a extremo y que no reciben ningún tipo de información explicita de la red.

Una de las áreas de mejora, es hacer TCP más adecuado para entornos de alta velocidad donde se destacan Fast TCP, HSTCP, STCP, HTCP, BIC, CUBIC. La siguiente área de mejora consiste en hacer TCP más robusto ante eventos que no impliquen congestión, tales como reordenamiento de paquetes y diferentes retardos

que TCP puede malinterpretar fácilmente. Dentro de este objetivo se destacan TCP-DOOR, TCP-PR, TCP-FR.

Por otro lado, se han realizado propuestas para mejorar el rendimiento de TCP en entornos especiales, principalmente para optimizar el comportamiento en redes inalámbricas. Una recopilación de las modificaciones de TCP propuestas incluye, entre otros, a TCP Vegas y TCP Veno, TCP Westwood y sus mejoras.

Uno de los escenarios más interesantes para su estudio, desde el punto de vista del rendimiento, es el de las redes heterogéneas, lo que implica que la ruta de comunicación de un extremo a otro consistirá en enlaces tanto cableados como inalámbricos. Dentro de estas, las redes de acceso inalámbricas típicas comprenden un host móvil conectado de forma inalámbrica a una estación base, que a su vez está conectada a la red troncal cableada, posiblemente a Internet. Por lo tanto, estudiar el rendimiento de TCP en una red con enlaces tanto cableados como inalámbricos ha generado un gran interés en la investigación en los últimos tiempos.

A pesar de todas las propuestas que se pueden encontrar en la literatura, que dieron lugar a tantas otras variantes del protocolo TCP dedicadas a resolver problemas con un determinado escenario o a obtener el máximo rendimiento posible en un tipo de enlace, estas no han tenido una gran aceptación, debido a que TCP debe funcionar en todo tipo de redes. Esto se debe principalmente a que un dispositivo puede conectarse a múltiples tipos de redes y que una conexión puede atravesar diferentes tipos de enlaces en su camino. Por tanto, a no ser que se trate de un medio muy acotado, se asumirá que no habrá una versión de TCP específica para cada tipo de red.

Por lo anteriormente expuesto, resulta interesante ensayar el comportamiento y el rendimiento de distintas variantes de TCP desarrolladas para distintos tipos de enlaces y escenarios, en una red heterogénea de acceso WLAN.

En cuanto a las simulaciones planteadas en este trabajo, no se tiene en cuenta el efecto de la movilidad, ya que en definitiva se traducirá a errores en la comunicación. Sin embargo, uno de los aspectos más frecuentes en estas redes y que tiene una influencia importante en el rendimiento de TCP son las pérdidas en ráfagas, lo que implica la pérdida de varios segmentos consecutivos pertenecientes a un mismo flujo.

TCP tiene, en principio, dos maneras de detectar la pérdida de un segmento, mediante la recepción de ACK duplicados, o bien mediante la caducidad de un temporizador de

retransmisión. Por lo tanto, en el caso de pérdidas de paquetes en ráfagas, la influencia en el rendimiento de TCP viene dada por el número total de segmentos perdidos de un mismo flujo, que pueden ser de distintas longitudes.

Es extremadamente complejo hacer una comparación exhaustiva de las distintas implementaciones de los esquemas de solución a la pérdida de rendimiento, porque cada una tiene como objetivo resolverla en un entorno con características diferentes. Sin embargo, en el presente trabajo, se considera como condición que debe cumplir la solución: que mantenga la semántica extremo a extremo de TCP.

En la actualidad, la investigación de la capa de transporte se centra básicamente en tres enfoques. Primero, la investigación sobre el TCP en sí aún está activa, con nuevos protocolos de congestión que se proponen para apuntar a la baja latencia y la utilización de ancho de banda completo, como el TCP Bottleneck Bandwidth and Round-Trip (BBR) o TCP Low Latency (LoLa). No obstante, la adopción de estos protocolos a escala todavía está limitada por la necesidad de actualizar el sistema operativo. La segunda alternativa, es el uso de protocolos de transporte implementados en el espacio de usuario, que usan UDP. En este sentido, se destaca Quick UDP Internet Connections (QUIC). Finalmente, la tercera tendencia está relacionada con la adopción de soluciones de múltiples rutas (Multipath) en la capa de transporte, principalmente con Multipath TCP (MPTCP), pero también con extensiones de múltiples rutas para SCTP, QUIC, etc. Si bien lo descripto anteriormente escapa a los alcances del presente trabajo, resulta interesante para ubicar la línea de investigación y, a su vez, la posibilidad de futuras direcciones de trabajo, una vez concluido el presente.

1.Motivación

Las redes inalámbricas han experimentado un importante auge en los últimos años debido a la aparición de dispositivos basados en la serie de normas 802.11x. Accesibles y fáciles de utilizar, las redes inalámbricas brindan flexibilidad y movilidad al usuario, sin tener que sacrificar la conexión a Internet o a la red del lugar de trabajo. El control de congestión “estándar” está diseñado a medida de las redes cableadas, donde los datos normalmente llegan en orden y prácticamente sin errores. De esta manera, se encuentra con desafíos importantes en los enlaces inalámbricos, debido a su naturaleza más impredecible. Estos enlaces poseen más perdidas que los enlaces cableados, dado que las señales que se propagan sufren de atenuación, interferencia y

ruido y los paquetes que se reciben pueden estar dañados y se descartan, produciendo la pérdida de paquetes en tránsito. Esto ocurre entonces, a nivel de enlace y no de red, escenario para el cual están pensadas la mayoría de las técnicas de control de congestión. Debido a las altas tasas de errores de transmisión y, en algunos casos, a la movilidad en redes inalámbricas, el reordenamiento de paquetes es más frecuente. Por lo tanto, el control de congestión, enfrenta nuevos desafíos en el entorno inalámbrico.

Cuando se pierde un paquete, el TCP “estándar” asume que es debido a la congestión en la red y disparara el procedimiento de control de congestión. Sin embargo, en el entorno inalámbrico, esta pérdida puede ser causada también por el reordenamiento de paquetes y las pérdidas en tránsito. De esta forma, TCP tiende a retransmitir paquetes y a reducir la tasa de envío de datos innecesariamente. En consecuencia, los recursos de red disponibles se desperdician y se subutilizan, reduciendo el rendimiento de TCP.

Las redes inalámbricas presentan características de transmisión muy diferentes de las redes tradicionales fijas. Estas características tienen su origen tanto en la propia naturaleza del medio físico, como en los efectos debidos a la movilidad. Es habitual observar largos retardos y pérdida de paquetes que no se deben exclusivamente a la congestión. También las desconexiones frecuentes producen lo que se denomina errores en ráfaga, donde se pierde una secuencia contigua de paquetes.

TCP transporta la mayor parte del tráfico de internet, por lo que el rendimiento de internet depende de cómo funciona TCP. Las características de rendimiento de una versión particular de TCP se definen, en gran medida, por el algoritmo de control de congestión que implementa. El problema que intenta resolver el control de la congestión, es el uso inteligente de los recursos disponibles en las redes. Como resultado, el control de congestión es uno de los temas más ampliamente estudiados en la investigación en Internet realizada en los últimos 20 años.

El creciente uso de equipos móviles generó mucho interés en investigaciones sobre el rendimiento y la mejora del protocolo TCP en entornos inalámbricos. Estos desarrollos se distribuyeron en distintos tipos de enfoques del problema.

2.Objetivos

El presente trabajo de tesis, tiene como objetivo efectuar una contribución para el conocimiento, actualización y avance del estado del arte, en referencia a las soluciones

para mejorar el rendimiento del protocolo TCP en redes con enlaces inalámbricos, considerando en escenarios simples emulando redes de acceso WLAN.

Si bien existen múltiples enfoques para mejorar el rendimiento del protocolo TCP en redes inalámbricas, el presente estudio se acota a las soluciones que mantienen el espíritu extremo a extremo (End-To-End) de TCP y que perciben a la red como una caja negra, es decir, la red no entrega ningún tipo de información explícita del estado de congestión o pérdida a ninguno de los hosts intervinientes. Esta línea de investigación intenta manejar las pérdidas a fin de mejorar la performance del TCP estándar. Se concentra en el análisis de las mejoras propuestas en el control de congestión de la capa de transporte, por ejemplo, los algoritmos Fast Retransmition y Fast Recovery. En general, su implementación requiere modificaciones en el TCP emisor, aunque en algunos casos particulares también puede ser necesario en el TCP receptor (SACK).

3.Desarrollo de Investigación

Para el desarrollo de la tesis, se investigaron como bases del proyecto, por su vinculación directa, los siguientes temas:

- Los algoritmos de control de congestión de TCP y sus variantes.
- El estudio del problema de degradación de rendimiento en las distintas variantes del protocolo TCP, utilizados en enlaces inalámbricos. Soluciones propuestas para mejorar el rendimiento.
- Las redes inalámbricas de área local WLAN, y la familia de estándar 802.11.

Para generar los datos que se utilizaron en este trabajo, se realizaron diferentes ensayos en el simulador de eventos discretos NS-2 (Network Simulator 2). En este simulador, se implementaron distintos escenarios y condiciones, pero en todos los casos, los modelos incluían enlaces cableados e inalámbricos del tipo 802.11 y, de esta manera, representar una red de acceso inalámbrica. De estas simulaciones se extrajeron distintas métricas que permitieron comparar y evaluar el rendimiento de las distintas versiones de TCP.

4.Estructura del Documento

En la primera parte de este trabajo se presenta una revisión general del protocolo TCP y algunos de los mecanismos de control de congestión que mantienen el espíritu extremo a extremo del protocolo TCP. Luego, se revisan diferentes algoritmos de control de congestión que hacen hincapié en los inconvenientes presentados por variados tipos de escenarios. El objeto de esta tarea es plantear los conceptos principales que serán utilizados a lo largo del trabajo, y establecer las líneas de investigación precedentes y actuales sobre este tipo de redes. Posteriormente se analizan las redes inalámbricas y las falencias que presenta el protocolo que desencadenan la merma de su rendimiento. Se presentan las distintas líneas en las soluciones planteadas por los diversos grupos de investigación, a fin de acotar o ubicar el espectro de estudio del presente trabajo.

A continuación, se realiza una breve discusión sobre los principales simuladores de redes y se justifica la elección del NS-2, como simulador Open Source para todos los ensayos del presente trabajo.

Finalmente se detallan las características principales de los escenarios de simulación sobre los que se desarrollarán los diversos estudios del rendimiento de las variantes de TCP utilizando distintas métricas. Se realizaron siete grupos de simulaciones donde se evidencian las estrategias que utilizan los algoritmos de control de congestión en cada una de las variantes del protocolo. Se utilizan métricas tales como la evolución del número de secuencia del segmento TCP, el tiempo total necesario para el envío de una cantidad fija de datos o el throughput instantáneo, entre otras, para poder comparar cualitativa y cuantitativamente estas estrategias. Se estudian comportamientos peculiares en la variante TCP Vegas que plantean la necesidad de ensayos adicionales.

Las simulaciones finales contemplan el análisis de lo que se denomina equidad o justicia, donde, con un modelo levemente complejizado, se estudian los ensayos de dos flujos compitiendo por los recursos de la red en primera instancia y posteriormente una contienda que incluye hasta ocho flujos.

En cuanto a la bibliografía utilizada para la realización de este trabajo, se encuentra al final del documento.

La presente tesis se completa con dos anexos en donde se presentan algunos de los scripts utilizados en las simulaciones y las contribuciones que se realizaron mientras se desarrollaba esta tesis tales como, presentación de trabajos en congresos, artículos originales en revistas, capítulo de libro y actividad de Extensión.

CAPITULO 1: El protocolo TCP

1.1. Introducción

La capa de Transporte es la responsable del envío del mensaje desde el proceso origen al proceso destino, para ello utiliza los servicios de la capa de Red que entrega los datagramas desde el host origen al host destino. La capa de Red utiliza los servicios de la capa de Enlace de Datos que entrega tramas entre dos nodos vecinos. TCP es el protocolo más utilizado en la capa de transporte, soporta la mayoría de las aplicaciones más populares de internet. En la siguiente figura (Figura 1), se observa la ubicación de la capa de transporte en el Stack TCP/IP y el modelo ISO OSI.

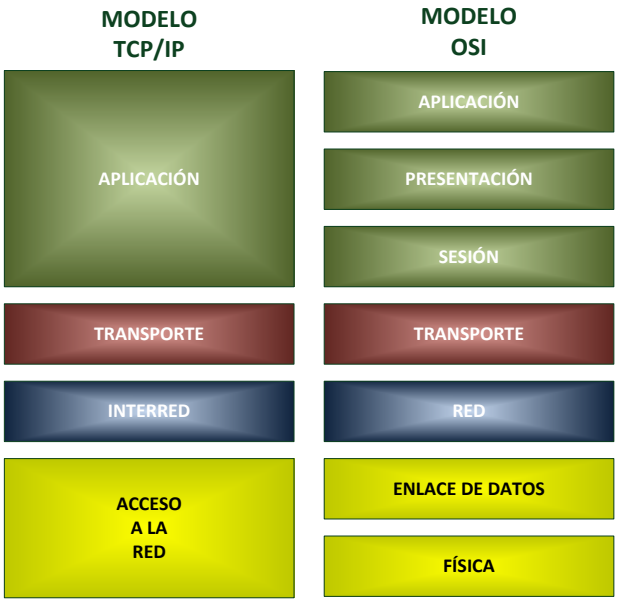

Figura 1: Capa de transporte en el Stack TCP/IP y en el modelo OSI

El problema de comunicación en entornos donde el medio puede perder o alterar los mensajes transmitidos, se puede resolver enviándolos hasta que se reciban correctamente. Este enfoque, denominado *Automatic Repeat Request (ARQ)*, constituye la base de muchos protocolos, incluyendo TCP. En este caso, el receptor envía un paquete de acuse de recibo o ACK para confirmar la correcta recepción de los datos.

En una red que no pierda ni daña paquetes, el rendimiento depende de su nivel de uso. Es decir, el rendimiento, medido como los datos que son enviados a través de la red por unidad de tiempo (throughput), va a depender del tamaño del paquete y del tiempo que requiere desde su envío hasta la llegada del ACK correspondiente. El *Round Trip Time (RTT)* es el tiempo que transcurre desde el instante en el que el emisor transmite el segmento TCP, hasta que recibe el reconocimiento (ACK), que informa que dicho segmento ha llegado correctamente al receptor. Si sólo se envía un único segmento TCP, el tiempo necesario para recibir el ACK correspondiente es RTT, por lo tanto, la tasa es el cociente entre el tamaño, en bytes, del segmento y el valor de RTT correspondiente (MTU/RTT, donde *MTU - Maximum Transmission Unit*), lo que produce una baja utilización de los recursos de la red. Por esta razón, el emisor debe determinar no sólo cuando debe inyectar un nuevo paquete, sino además en que cantidad. Para maximizar la transferencia resulta imprescindible maximizar el número de segmentos a enviar por cada RTT. Sin embargo, si el emisor inyecta paquetes a una tasa que el receptor o la red no pueden manejar, se descarta ese excedente, lo que reduce el rendimiento.

Uno de los problemas más importantes de rendimiento que experimentan los protocolos basados en retransmisiones es cuánto tiempo deben esperar antes de concluir que el paquete se perdió y requiere ser reenviado. Sin embargo, estos valores no se pueden conocer con certeza y todos ellos varían con el tiempo a medida que, por ejemplo, se agrega carga adicional. En este caso, la mejor estrategia es que sea el mismo protocolo quien estime dinámicamente el valor de estos tiempos, mediante un proceso estadístico basado en la medida RTT. Una vez que se estima el valor de RTT, que tendrá variaciones en forma dinámica, se debe definir el valor del tiempo de espera o *Timeout*, para que el protocolo retransmita el paquete. Este valor debe ser mayor que la media para

evitar retransmisiones no deseadas, pero tampoco debería ser demasiado grande porque influiría en el rendimiento.

Uno de los aspectos más importantes de TCP es el control de congestión. Si bien no formó parte de la implementación original, define, en gran medida, su rendimiento. TCP regula la inyección de datos a la red en función del estado de congestión de la misma. Los algoritmos de control de congestión tratan de determinar dinámicamente el ancho de banda y la latencia de la red. A partir de estos datos, modifican la tasa de envío de paquetes y su rendimiento de acuerdo a los cambios de tráfico para evitar el colapso de la subred.

TCP fue desarrollado y posteriormente optimizado para redes cableadas, lo que implica que la perdida de paquetes se debe casi con exclusividad a la congestión en la red, suponiendo una tasa de error de bits en tránsito prácticamente inexistente. Con el crecimiento en la demanda de movilidad, se comenzó a utilizar en enlaces inalámbricos, que poseen características muy diferentes. De esta manera, los retrasos y las pérdidas de paquetes ya no se deben exclusivamente a la congestión, sino también al daño de paquetes en tránsito o a la desconexión temporal de los nodos. En estos entornos, al identificar de manera incorrecta el origen de las perdidas, TCP puede tener un comportamiento muy poco eficiente.

En virtud de ello, a lo largo del tiempo se han realizado modificaciones sobre el protocolo, que tratan de adaptarlo a distintas condiciones de trabajo, acompañando a las innovaciones tecnológicas que se han desarrollado en el área de las telecomunicaciones y que utilizan tanto medios cableados como inalámbricos.

1.2. Transmission Control Protocol

Las especificaciones originales para el protocolo TCP que se definieron en el RFC793 [1], y corrigieron en el RFC1122, fueron publicadas por Internet *Engineering Task Force (IETF)* [2]. Desde ese momento, fueron revisadas y extendidas para incluir, clarificar y mejorar la conducta del control de congestión en los RFC5681, RFC3782, RFC3517, RFC3390 y RFC3168; los tiempos de espera de retransmisión en los RFC6298 [3], RFC5682, RFC4015,

NAT (Network Address Translation) en el RFC5382 [4]; comportamiento del ACK RFC 2883, seguridad en los RFC6056, RFC5927 y RFC5926, administración de la conexión en el RFC5482 [5], directrices de implementación de mecanismo urgente en el RFC6093 [6], entre otras.

Una de las ideas que más influenciaron el diseño de TCP fue el principio END-To-END. Este enuncia que las cuestiones relativas a los protocolos de transporte son responsabilidad de los extremos de la conexión y, por lo tanto, no deben ser delegados a la red. Dentro de estas responsabilidades se encuentran la corrupción de datos y la congestión. Si bien todos los enlaces en Internet tienen sumas de comprobación de capa de enlace para proteger contra la corrupción de datos, TCP agrega su propia suma de comprobación. TCP es esencialmente la única capa que se ocupa de la gestión de la congestión lo que no significa que el backbone de Internet no debe preocuparse por la congestión; sino que los mecanismos de gestión de este backbone no deben reemplazar la gestión de congestión de extremo a extremo [7].

TCP se diseñó específicamente para proporcionar un flujo de bytes confiable de extremo a extremo, a través de una red no confiable que puede tener diferentes topologías, anchos de banda, retardos, tamaños de paquete y demás parámetros.

TCP se caracteriza por ofrecer a las aplicaciones que lo utilizan, los siguientes servicios [8]:

- *Transferencia de datos*: Desde el punto de vista de la aplicación, TCP transfiere un flujo continuo de bytes a través de la red.
- *Confiabilidad*: TCP asigna un número de secuencia a cada byte transmitido, y espera una confirmación positiva (ACK) de la capa TCP del receptor.
- *Control de flujo*: cuando el receptor le envía un ACK al remitente, también le indica el número de bytes que puede recibir (más allá del último segmento TCP recibido) sin provocar el desbordamiento en sus buffers internos.

- *Multiplexación*: Lograda mediante el uso de puertos, permite a varios procesos de la capa de aplicación de un host acceder a la red por una sola interface.
- *Conexiones lógicas*: TCP inicializa y mantiene cierta información de estado para cada flujo de datos. La combinación de este estado, incluyendo sockets, números de secuencia y tamaños de ventana, se denomina conexión lógica.
- *Full dúplex*: TCP proporciona flujos de datos concurrentes en ambas direcciones.

1.2.1. El Segmento TCP

TCP intercambia datos en forma de segmentos [9]. Un segmento está formado por un encabezado fijo de 20 bytes (puede tener una parte opcional), seguido de los datos de aplicación. El tamaño de los segmentos lo define el mismo protocolo, puede acumular datos de varias escrituras para formar un segmento o dividir los datos de una escritura en varios segmentos. Sin embargo, hay dos límites que restringen el tamaño de segmento: cada segmento debe caber en la carga útil de 65,515 bytes del IP. Cada red tiene una unidad máxima de transferencia, abreviado en sus siglas en inglés *MTU (Maximum Transmission Unit)*.

En la mayoría de las redes, el tamaño de la trama para transportar los datos de los protocolos de las capas superiores se encuentra limitado en la capa de Enlace de Datos, lo que, además, limita el número de bytes de carga útil. Cuando dos hosts se comunican a través de la misma red, el tamaño queda definido por el MTU del enlace que los conecta, pero cuando las comunicaciones se realizan a través de múltiples redes (internet), cada uno de los enlaces involucrados puede poseer distintos MTU. Se denomina "*path MTU*" al valor mínimo de los valores de MTU de todos los enlaces de la ruta entre los hosts. El descubrimiento del path MTU (*PMTUD Path MTU Discovery*) se logra en base de mensajes ICMP. La forma en que TCP utiliza PMTUD se describe para TCP/IPv4 en RFC1191 y para TCP/IPv6, RFC1981.

Cuando TCP establece una conexión, utiliza el valor mínimo de MTU de la interfaz de salida, o el valor del tamaño máximo de segmento *MSS* (*Maximum Segment Size*) anunciado por el otro extremo, como base para seleccionar el *Send Maximum Segment Size (SMSS)*. Una vez elegido el SMSS inicial, todos los datagramas IPv4 de la conexión tienen activado el campo de bits de DF (no fragmentar). De esta manera, si en la ruta hay un enlace con MTU menor, el paquete se descarta y se envía *Internet Control Message Protocol (ICMP) Fragmentation Needed*. TCP disminuye el tamaño del segmento y lo retransmite.

En la siguiente figura (Figura 2) se observa el formato del encabezado TCP.

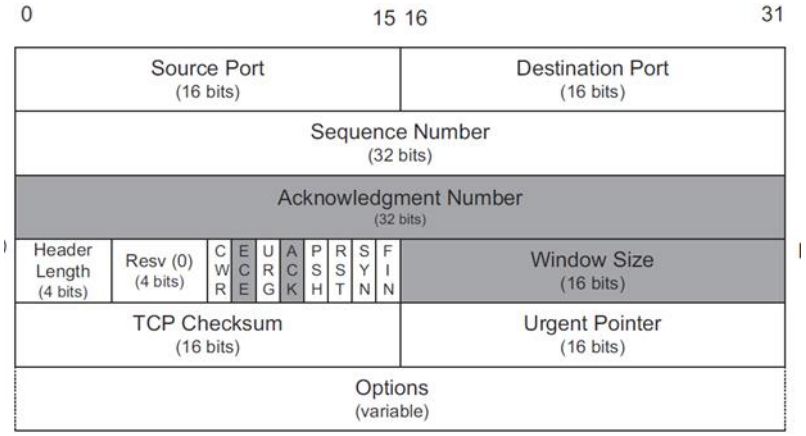

Figura 2: encabezado TCP (TCP Header)

[FUENTE: TCP/IP illustrated, W. Richard Stevens, 2nd ed. Addison-Wesley [9]]

Cada segmento comienza con un encabezado de formato fijo de 20 bytes. El encabezado fijo puede ir seguido de opciones de encabezado. Tras las opciones, si las hay, pueden continuar bytes de datos.

A continuación, se realiza una breve descripción de los elementos que conforman el encabezado TCP de la Figura 2.

Source Port/Destination Port: Puerto origen y puerto destino que, junto con las direcciones IP origen y destino, identifican de manera unívoca cada conexión (denominado socket en la bibliografía TCP).

Sequence Number: Es un número sin signo de 32 bits, e indica el número de secuencia del primer octeto de datos en este segmento, excepto cuando está presente el indicador SYN. Si el indicador SYN está activo, se

trata del número de secuencia inicial (ISN) y el primer octeto de datos es el ISN+1. Antes de que se envíe el segmento SYN para establecer la conexión, se elige un ISN para esa conexión. El ISN debe cambiar con el tiempo y la RFC 793 especifica que el ISN debe verse como un contador de 32 bits que aumenta en 1 cada 4µs. El propósito de esto, es evitar que los números de secuencia de los segmentos de una conexión se superpongan con los números de otra conexión.

Acknowledgement Number: El campo número ACK contiene el siguiente número de secuencia que el receptor del segmento espera recibir, es decir, el siguiente byte esperado. Este campo sólo es válido si el bit ACK está activado.

Header Length: El campo Longitud del encabezado es la longitud total del encabezado en palabras de 32 bits. Esto es necesario porque la longitud del campo Opciones es variable.

Flags: En este campo actualmente se definen ocho bits de control, indicadores, o flags. Las implementaciones más antiguas sólo comprendían seis de estas banderas. Estos dos indicadores adicionales son CWR y ECE se utilizan junto con dos banderas del encabezado IP (ECT y CE) para comunicarle al emisor de la existencia de congestión en la red, evitando las pérdidas de paquetes y sus retransmisiones.

- *CWR – Congestion Window Reduced:* RFC3168, indica que se recibió un segmento con el indicador ECE activado y se notifica al mecanismo de control de congestión.
- *ECE – ECN Eco:* Se recibió un paquete con el indicador Congestion Experienced CE en el encabezado IP durante la transmisión normal. Los bits CWR y ECE definidos en el RFC3128 se utilizan para la notificación de congestión explícita (Explicit Congestion Notification ECN)
- *URG – Urgente:* El bit urgente sirve para indicar que los datos en el campo Puntero Urgente son válidos e indican un desplazamiento en bytes a partir del número actual de secuencia en el que se encuentran datos urgentes.
- *PSH – Push:* Indica datos que se deben transmitir de inmediato. Por este medio se solicita al receptor que entregue los datos a la aplicación y no los almacene en buffer

- *RST – Reset:* Restablecer la conexión se usa para restablecer una conexión debido a una caída de host u otra razón; también sirve para rechazar un segmento no válido o el intento de abrir una conexión.
- *SYN:* se usa para establecer conexiones. Cuando se establece una nueva conexión, el bit SYN se activa. Estos segmentos se denominan segmentos SYN
- *FIN:* se usa para liberar una conexión; especifica que el emisor no tiene más datos que transmitir.

Window Size: El campo tamaño de la ventana se utiliza para realizar el control de flujo, pues cada extremo comunica la cantidad de bytes que está dispuesto a aceptar. Se trata de un campo de 16 bits, que limita el tamaño máximo de la ventana a 65.535 bytes y, por lo tanto, restringe el rendimiento de TCP en algunos entornos.

TCP CheckSum: Es una suma de verificación del encabezado, los datos y el pseudoencabezado. El pseudoencabezado contiene las direcciones IP de 32 bits de las máquinas de origen y de destino, el número de protocolo, y la cuenta de bytes del segmento TCP (incluido el encabezado). La inclusión del pseudoencabezado en el cálculo de la suma de verificación TCP ayuda a detectar paquetes mal entregados. Sin embargo, esto viola la jerarquía de protocolos puesto que las direcciones de IP que contiene pertenecen a la capa de red y no a la capa de transporte.

Urgent Pointer: El campo puntero urgente es válido sólo cuando el bit indicador URG está activado. Este valor es el desplazamiento que hay que agregar al número de secuencia del segmento, para obtener el número de secuencia del último byte de datos urgentes.

Options: [8] El campo opciones ofrece una forma de agregar características extra no cubiertas por el encabezado y puede o no estar presente. En la definición original del protocolo, sólo se definieron las opciones End of Option List (EOL), No Operation (NOP) y Maximum Segment Size (MSS). A continuación, se detallan las opciones actuales de interés, con las RFC de referencia. La lista completa es mantenida por IANA (Internet Assigned Numbers Authority) [9].

- *Maximum Segment Size (*MSS): El campo tamaño máximo del segmento tiene 16 bits y es el tamaño de segmento más grande que un TCP está dispuesto a recibir. El valor de MSS solo contempla los bytes de datos de TCP.
- *Selective Acknowledgement (SACK):* TCP maneja los acuses de recibo ACK en forma acumulativa. Envía el próximo número de secuencia a recibir, confirmando la correcta recepción de todos hasta el número inmediatamente anterior. Esta opción (SACK RFC2018, RFC2883), se negocia en el establecimiento de la conexión. El TCP que recibe datos fuera de secuencia, puede ahora enviar al emisor la opción SACK describiendo los datos recibidos para que realice la retransmisión más eficientemente. La información contenida consiste en una serie de números de secuencia, que representan los bloques de datos que el receptor ha recibido con éxito.
- *Window Scale (WSCALE or WSOPT):* Escala de ventana definida en RFC1323 [10], permite al emisor y al receptor negociar un factor de escala de ventana. El campo tamaño de la ventana o Window Size de la cabecera TCP tiene longitud 16 bits, el tamaño máximo de la ventana sería de 64 Kbytes. Para definir mayores tamaños de ventana, se propuso esta opción de escala.
- *Timestamps:* Permite al emisor adjuntar una marca de tiempo o timestamp en cada segmento que después el receptor copiará en el paquete de confirmación, permitiendo al emisor calcular el RTT por cada ACK recibido.
- *User Timeout (UTO):* La opción de tiempo de espera del usuario (UTO) es una capacidad TCP relativamente nueva descrita en el RFC5482. El valor UTO (llamado USER_TIMEOUT) especifica la cantidad de tiempo que un emisor TCP está dispuesto a esperar por un ACK de datos pendientes antes de concluir que el extremo remoto ha fallado. USER_TIMEOUT ha sido tradicionalmente un parámetro de configuración local para TCP (RFC793). La opción UTO permite que un TCP señale su valor USER_TIMEOUT a su par de conexión.
- *Authentication Option (TCP-AO):* esta opción fue diseñada para mejorar y reemplazar un mecanismo anterior llamado TCP-MD5 (RFC 2385). La opción de autenticación TCP (TCP-AO), definida en RFC 5925, utiliza un algoritmo de cifrado criptográfico, en combinación con una clave secreta conocida por cada extremo, para autenticar cada segmento.

1.2.2. Gestión de las conexiones

TCP debe establecer una conexión entre los dos procesos antes de transferir un dato, lo que lo define como un protocolo orientado a la conexión. En la

siguiente figura (Figura 3) se pueden ver las tres fases de una conexión TCP: el establecimiento de la conexión, la trasferencia de datos y el cierre de la conexión [9].

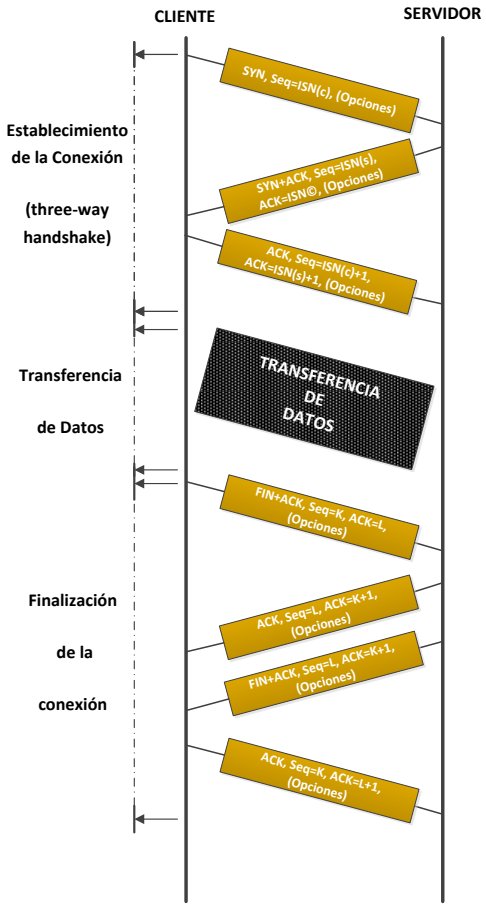

Figura 3: Fases de una conexión TCP

Establecimiento de la Conexión (three-way handshake)

1. El emisor o cliente, envía un segmento SYN, un segmento TCP que tiene activado el bit SYN en el encabezado, especificando el número de puerto del receptor al que desea conectarse y su propio número de secuencia inicial o ISN(c). Luego de esto puede o no enviar una o más opciones.
2. El receptor o servidor responde con su propio segmento SYN que contiene su propio número de secuencia inicial ISN(s). El servidor acusa el recibo del SYN del cliente por ACK ISN (c) + 1
3. El cliente debe confirmar el segmento SYN, enviado por el servidor, mediante un ACK ISN (s) + 1.

La figura anterior (Figura 3) también muestra cómo se cierra una conexión TCP. Cualquiera de los extremos puede iniciar una operación de cierre, que la operación se inicia enviando un segmento TCP con el bit FIN del encabezado activado.

Finalización de la conexión

1. El extremo que desea cerrar la conexión envía un segmento FIN en el que especifica el número de secuencia actual que el receptor espera, K en la Figura 3. El FIN también incluye un ACK para los últimos datos recibidos, L en la Figura 3.
2. El receptor responde con el ACK K+1 para indicar la recepción exitosa del segmento FIN. En este punto, se notifica a la aplicación que el otro extremo de su conexión ha realizado un cierre. El número de secuencia es igual a L.
3. Para completar el cierre, el segmento final contiene un ACK para el último FIN.

Mientras que para establecer una conexión se utilizan tres segmentos, normalmente se requieren cuatro segmentos TCP para liberar una conexión, un FIN y un ACK para cada sentido. El modelo de comunicaciones de datos de TCP es bidireccional, lo que significa que es posible cerrar el flujo de datos en una sola dirección, mientras persiste en el sentido contrario. Sin embargo, es posible que el primer ACK y el segundo FIN estén contenidos en el mismo segmento, reduciendo la cuenta total a tres [11].

1.2.3. Diagrama de Estados de TCP

Las reglas que definen el comportamiento de TCP ante un evento dependen del estado en que se encuentra y de estímulos, tales como segmentos que se envían o reciben y temporizadores que se agotan. Estos estados y sus transiciones se pueden observar en la Figura 4 [9].

Los diferentes estados están representados por óvalos y las transiciones entre ellos por flechas. Algunas transiciones se disparan por la recepción de un segmento con ciertos campos de bits de control activados (SYN, ACK, FIN), mientras que otras producen el envío de un segmento con campos de bits de control activados. También pueden ser activadas por acciones de aplicación o por temporizadores que expiran.

TCP comienza en el estado CLOSED y pasa al estado SYN_SENT si se produce una apertura activa (Cliente) o al estado LISTEN si es una apertura pasiva (Servidor). Las dos transiciones que llevan al estado ESTABLISHED corresponden al establecimiento de una conexión TCP, mientras que las dos transiciones que dejan este estado, corresponden a la finalización de una

conexión TCP. En el estado ESTABLISHED es donde se puede llevar a cabo la trasferencia de datos, en ambos sentidos entre los extremos.

Se definen los estados FIN_WAIT_1, FIN_WAIT_2 y TIME_WAIT como los estados que deben sucederse cuando es la aplicación local la que inicia una solicitud de cierre de la conexión (Active Close). Los estados CLOSE_WAIT y LAST_ACK se corresponden con un cierre pasivo. El cierre simultáneo está representado por el estado CLOSING.

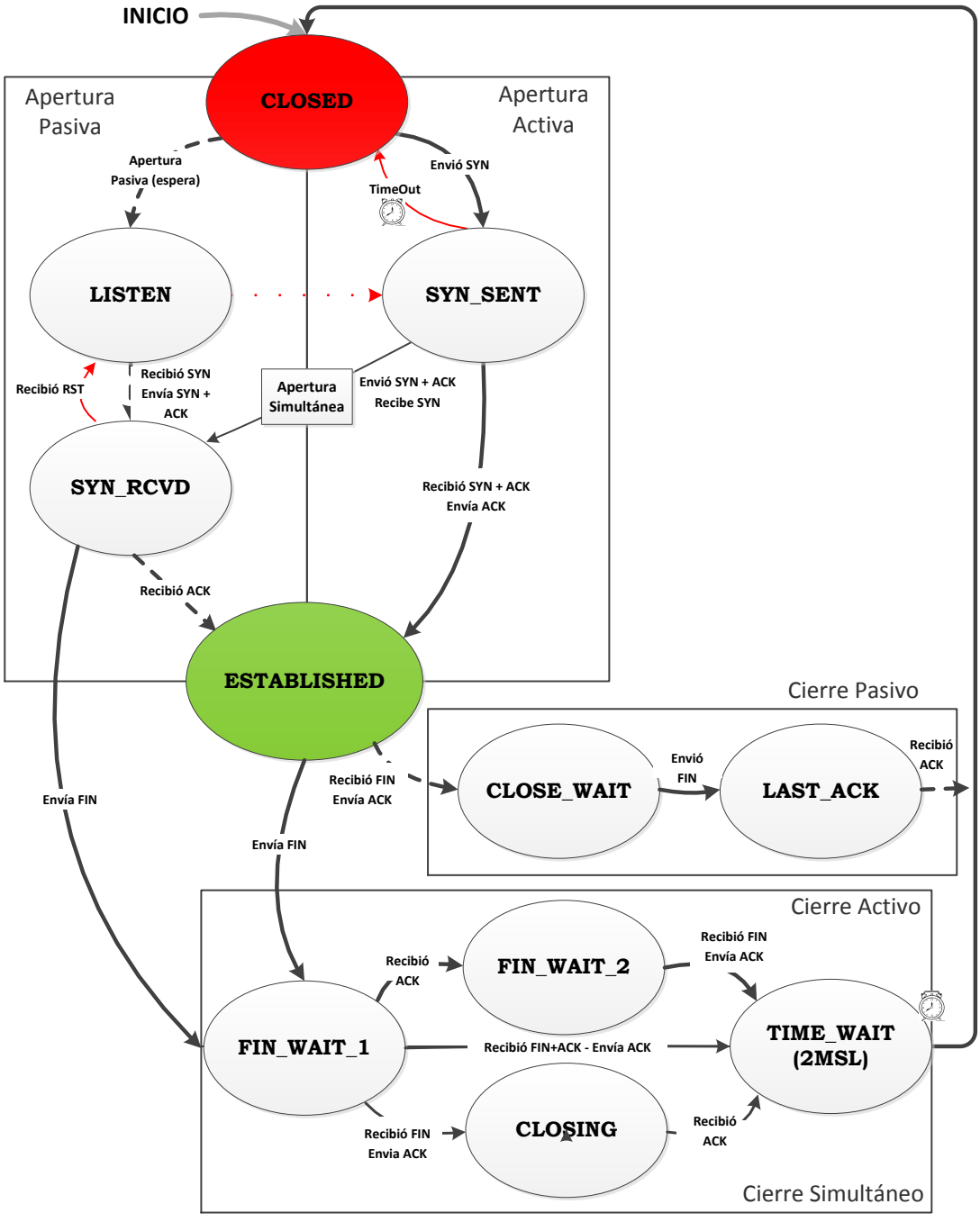

Figura 4: Diagrama de transición de estados TCP (máquina de estado finito) [9]

La transición de estado de *LISTEN* a *SYN_SENT* no es compatible con los sockets de Berkeley. La transición de retorno de *SYN_RCVD* a *LISTEN* sólo es válida si se llegó al estado desde el estado *LISTEN* y no desde el estado *SYN_SENT*. Si TCP está en el estado de *LISTEN* y se recibe un segmento SYN (apertura pasiva), TCP responde enviando un segmento SYN con el ACK y pasa al estado *SYN_RCVD*. Si recibe el segmento con el ACK, se establece la conexión y pasa al estado *ESTABLISHED*. Sin embargo, si recibe un segmento con el bit RST activado (segment reset), TCP regresa al estado *LISTEN* a la espera de una nueva conexión.

En la siguiente tabla (Tabla 1) se observan los estados posibles de TCP.

ESTADO	DESCRIPCION
CLOSED	No hay conexión activa ni pendiente
LISTEN	Servidor espera un pedido de conexión
SYN RCVD	Llego pedido de conexión, a la espera de ACK
SYN SENT	La aplicación local comenzo a abrir una conexión
ESTABLISHED	Conexión extablecida, estado normal de trasnferencia de datos
FIN WAIT 1	La aplicación local informo que no tiene mas datos para enviar
FIN WAIT 2	El otro extremo informo que desea cerrar la conexión
TIME WAIT	Espera a todos los paquetes en transito
CLOSING	Ambos extremos cierran simultaneamente
CLOSE WAIT	El otro extremo inicio un cierre de la conexión
LAST ACK	Espera a todos los paquetes en transito

Tabla 1: Estados posibles en la máquina de estados finitos de administración de conexiones TCP (RFC 793)

En la siguiente figura (Figura 5), el cliente inicia la conexión con el envío de un segmento SYN y pasa al estado *SYN_SENT*, mientras que el servidor se encuentra en el estado *LISTEN* y cuando llega el segmento del cliente, le acusa el recibo del segmento SYN, pero activa el bit RST (reset). Cuando el cliente recibe el RST, pasa al estado *CLOSE*.

El estado *TIMEWAIT* o estado de espera *2MSL* es un estado en el que TCP espera un tiempo igual al doble de Maximum Life Lifetime (MSL). MSL es el tiempo máximo que cualquier segmento puede existir en la red antes de ser descartado. En el RFC793 se especifica que MSL debe ser igual a 2 minutos.

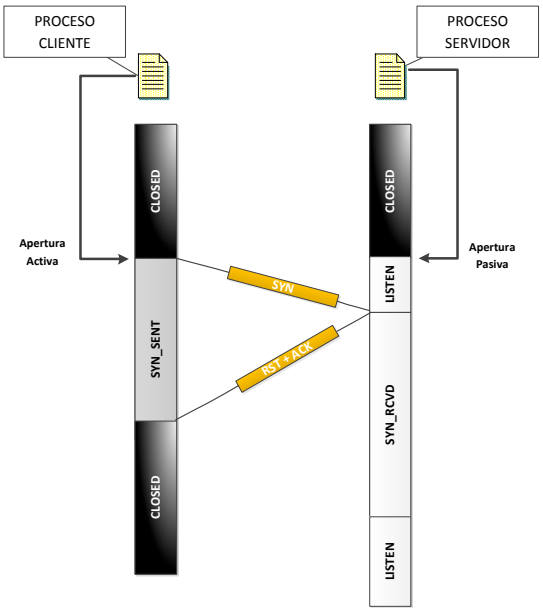

Figura 5: Transición del estado SYN RCVD al estado LISTEN

Una vez establecido el valor de MSL, cuando TCP realiza un cierre activo y envía el ACK final, la conexión debe permanecer en el estado *TIMEWAIT* durante el doble del valor MSL. Esto permite que TCP reenvíe el ACK final en caso de que se pierda. TCP retransmitirá siempre el segmento FIN hasta que reciba un ACK final. Mientras TCP está en este estado, la dirección IP más el puerto del cliente y la dirección IP más el puerto del servidor no pueden ser utilizados. En las aplicaciones interactivas, es habitual que el cliente que realiza el cierre de la conexión en forma activa pase al estado de *TIMEWAIT*, mientras que el servidor que realiza el cierre activo no pase por este estado, dado que en el servidor normalmente se utilizan los puertos bien conocidos. De esta forma, una conexión nueva no podrá confundir algún segmento retrasado de la conexión anterior como perteneciente a la nueva instancia.

Si se reinicia un host que se encontraba en estado *TIMEWAIT*, para evitar que segmentos de la conexión anterior retardados sean considerados como pertenecientes a una nueva conexión, el RFC793 establece que TCP deberá esperar un tempo igual a MSL antes de establecer una nueva conexión después de su inicio, a esto se lo denomina *Tiempo de Silencio* o *Quiet Time*.

TCP envía un segmento FIN y pasa al estado *FIN_WAIT_1*. Cuando recibe el ACK correspondiente pasa al estado *FIN_WAIT_2*. En este punto, debe esperar a que la aplicación del otro extremo cierre su lado de la conexión,

recibiendo el segmento FIN. Cuando recibe el segmento, se produce al cierre pasivo y TCP pasa del estado *FIN_WAIT_2* al estado *TIME_WAIT*.

En la siguiente figura (Figura 6), se observan las tres fases de TCP y sus respectivos estados

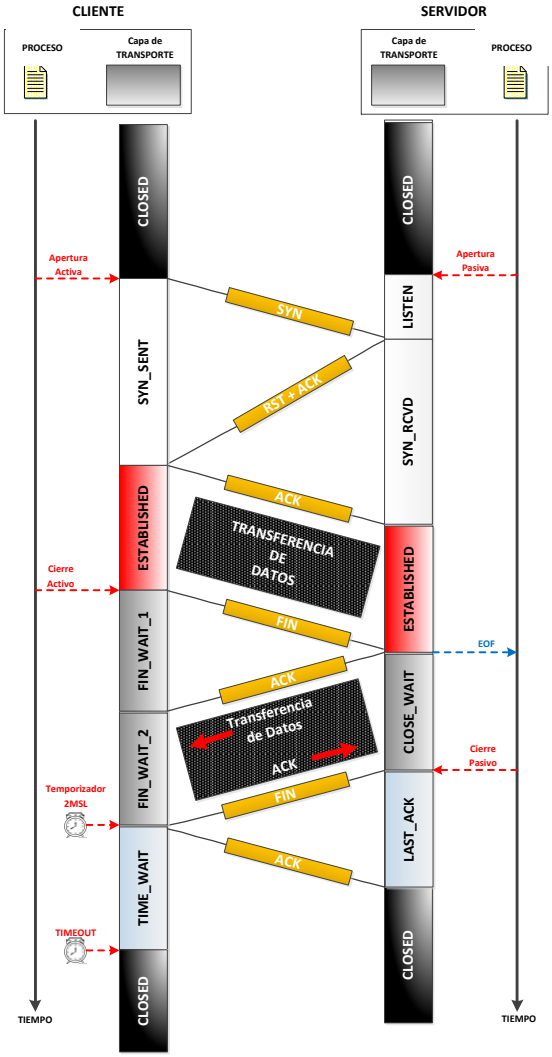

Figura 6: Estados del TCP cliente y servidor

TCP fue diseñado para que cuando se produzca una apertura simultánea, termine en una única conexión. Ambos extremos envían un segmento SYN al mismo tiempo y pasan al estado *SYN_SENT*. Cuando los dos extremos reciben este segmento SYN pasan al estado *SYN_RCVD,* y ambos reenvían el segmento SYN con el acuse de recibo (ACK) del SYN recibido. Cuando los dos extremos reciben los segmentos SYN más los ACK, pasan al estado *ESTABLISHED*.

Cuando las aplicaciones de los dos extremos no tienen más datos para enviar al mismo tiempo, pasan del estado *ESTABLISHED* al *FIN_WAIT_1,* lo que produce que se envíen sendos segmentos FIN. Cuando los extremos reciben el segmento, pasan del estado *FIN_WAIT_1* a *CLOSING* y cada uno manda su ACK final. Al recibir estos ACK, los dos extremos pasan al estado *TIME_WAIT* y se inicia la espera de *2MSL* [9].

1.2.4. Temporizadores y Retransmisiones

Para poder enviar los datos libres de errores, TCP reenvía los datos que supone perdidos. Para decidir qué datos necesita reenviar, TCP depende de los acuses de recibo (ACK) enviados por receptor. Cuando se pierden segmentos de datos o segmentos ACK, TCP reenvía los datos de los cuales no recibió el ACK.

Cuando TCP envía datos, inicia un temporizador y si no recibió el ACK correspondiente cuando este temporizador expira, se produce la retransmisión. A este intervalo se lo llama *Retransmission Timeout (RTO)*. Es necesario definir el valor del temporizador de retransmisión (RTO) para definir cuánto tiempo debe esperar TCP el ACK de los datos enviados. El protocolo debe determinar este valor en forma dinámica, dado que la trasmisión se realiza a través de variedad de entornos que pueden cambiar con el tiempo. La forma en que se ajusta el valor de RTO es fundamental, ya que, si retrasmite un segmento antes, puede estar inyectando innecesariamente un segmento duplicado en la red. A la inversa, si retrasa mucho el envío, la utilización de la red global (y el rendimiento de una sola conexión) disminuye [8].

Dado que el valor de RTT puede variar en el tiempo (cambio de ruta, carga de red, etc.), TCP debe realizar un seguimiento de estos cambios y modificar su tiempo de espera para lograr un buen rendimiento. El desafío de TCP es realizar una estimación del rango de valores de RTT a partir de un conjunto de muestras, que varían en el tiempo. A partir de esta estimación, debe establecer el valor del temporizador RTO. TCP estima un valor de RTT para cada conexión por separado y dispara un temporizador para cada segmento pendiente de confirmación.

La *especificación original de TCP (RFC793)* [1], definía la actualización de la estimación RTT suavizada SRTT (Smoothed RTT estimator), de la siguiente forma (Ecuación 1):

$$\mathrm{SRTT} \leftarrow \alpha(\mathrm{SRTT}) + (1-\alpha)\mathrm{RTT} \qquad \textbf{Ecuación 1}$$

La constante *α*, es el factor de suavizado y su valor oscila entre 0,8 y 0.9. Este tipo de promedio también se conoce como media móvil ponderada exponencialmente (EWMA).

Una vez obtenido el valor de SRTT, el RFC793 recomienda que el valor de RTO se establezca de la siguiente manera (Ecuación 2):

$$\mathrm{RTO} = \min(\mathrm{ubound}, \max(\mathrm{ibound}, (\mathrm{SRRT})\beta)) \qquad \textbf{Ecuación 2}$$

Donde, *β* es un factor de varianza de retardo (el valor recomendado es 1,3 a 2,0), *ubound* es un límite superior (se sugiere 1 minuto), y *ibound* es un límite inferior (el valor sugerido es 1 s).

Sin embargo, cuando el valor de RTT presenta grandes fluctuaciones, ajustar el RTO con la media suavizada (SRTT), en general, produce retransmisiones innecesarias cuando el RTT real es mayor que el esperado. El aumento de valor de la muestra de RTT indica un crecimiento en la carga de la red, por otro lado, las retrasmisiones innecesarias contribuyen a empeorar esta situación.

Para resolver este problema, *el Método Estándar* realiza el seguimiento de la estimación de la varianza en las mediciones de RTT, además de la estimación de su promedio como se establecía en la definición original. Al establecer el RTO basado en una estimación de la media y de la varianza, se logra una mejor respuesta en escenarios con grandes fluctuaciones de los tiempos de ida y vuelta (RTT) [9].

La desviación media es una buena aproximación a la desviación estándar, pero es más fácil y más rápido de calcular. Por lo tanto, se debe estimar la media y la desviación media a partir de cada medición de RTT (M), como se expresa en las siguientes ecuaciones (Ecuación 3, Ecuación 4 y Ecuación 5).

$$\mathrm{srrt} \leftarrow (1-\mathrm{g})(\mathrm{srtt}) + (\mathrm{g})\mathrm{M} \qquad \textbf{Ecuación 3}$$

$$\text{rttvar} \leftarrow (1 - \text{h})(\text{rttvar}) + (\text{h})(|\text{M} - \text{srtt}|) \quad \textbf{Ecuación 4}$$

$$\text{RTO} = \text{srtt} + 4(\text{rttvar}) \quad \textbf{Ecuación 5}$$

Donde M es el último valor medido de RTT.

En este caso, el valor *srtt* reemplaza al valor anterior de *SRTT* y el valor *rttvar*, que proviene de la desviación media, se utiliza en lugar de β para determinar el valor de RTO. La ganancia *g* es el peso dado a una nueva muestra de RTT (*M*) en el valor de srtt y se fija a 1/8. La ganancia *h* es el peso dado a una nueva desviación media de la muestra para la estimación de la desviación rttvar y se fija a 1/4. La ganancia mayor para la desviación hace que el RTO suba más rápido cuando el RTT cambia.

Este método estándar se adoptó como base para el RFC6298 y se le introdujeron ligeras mejoras. La Granularidad del Reloj se utiliza para refinar la forma en que se realizan las actualizaciones del RTO y se le asigna un límite inferior. Se calcula con la siguiente ecuación (Ecuación 6),

$$\text{RTO} = \max\big(\text{srtt} + \max\big(\text{G}, 4(\text{rttvar})\big), 1000\big) \quad \textbf{Ecuación 6}$$

Donde, *G* es la granularidad del temporizador, el valor 1000 ms. representa un límite inferior recomendado.

TCP obtiene el valor de reloj de una variable que se actualiza a medida que avanza el reloj del sistema, no necesariamente uno a uno. La longitud de ese reloj “TICK” se la llama granularidad.

Antes de iniciar una conexión y enviar el primer SYN, TCP no tiene ninguna medida del valor de RTT y ningún valor estimado de RTO. De acuerdo con RFC6298, el ajuste de los *Valores Iniciale*s para el RTO debe ser entre 1 y 3 s, en el caso del segmento inicial SYN. Cuando se recibe la primera medición de RTT (M), los estimadores se inicializan como se observa en las siguientes ecuaciones (Ecuación 7 y Ecuación 8),

$$\text{srtt} \leftarrow \text{M} \quad \textbf{Ecuación 7}$$

$$\text{rttvar} \leftarrow \frac{\text{M}}{2} \quad \textbf{Ecuación 8}$$

Si un temporizador caducó y se reenvía el segmento, cuando el TCP emisor recibe el ACK correspondiente, no puede discernir si corresponde al segmento original o al segmento retransmitido. Para eliminar este problema, el *Algoritmo de Karn* [8] establece que cuando se agota un temporizador y se realiza la retransmisión correspondiente, no se deben actualizar los estimadores RTT ante la llegada del acuse de recibo de los datos retransmitidos.

Al mismo tiempo, la red está indicando que experimenta alguna incapacidad para entregar paquetes. En tal caso, es beneficioso reducir la carga de la red disminuyendo la tasa de retransmisión, al menos hasta que los ACK de los segmentos comiencen a llegar. Este razonamiento es la base para el backoff o retroceso exponencial. TCP aplica un factor de retroceso al RTO, que se duplica cada vez que expira un temporizador de retransmisión, de la siguiente forma (Ecuación 9):

$$\mathrm{RTO} = \gamma \mathrm{RTO} \qquad \textbf{Ecuación 9}$$

Donde, el valor de *γ* es igual a 1 en condiciones normales y se incrementa al doble por cada timeout que se produce, y continúa hasta que se recibe un acuse de recibo (ACK) para un segmento que no fue retransmitido.

La forma en que Linux realiza la estimación de RTT, denominado *Método Linux* [9], es un poco diferente del método estándar. Utiliza una granularidad de reloj más fina (1 ms), junto con la opción Timestamps. Esta combinación de mediciones más frecuentes del RTT y el reloj de grano fino contribuyen a una estimación más precisa del RTT, pero también tiende a minimizar el valor de la desviación media en el tiempo. El valor de RTO se establece de forma análoga a la Ecuación 5.

Linux mantiene las variables *srtt* y *rttvar*, pero además utiliza dos nuevas: *mdev* y *mdev_max*. El valor *mdev* mantiene la estimación de la desviación media usando el algoritmo estándar para rttvar. El valor *mdev_max* mantiene el valor máximo de *mdev* observado sobre la medida de RTT. Además, rttvar se actualiza regularmente para asegurar que tiene, por lo menos, un valor tan grande como *mdev_max*. Por lo tanto, el RTO nunca cae por debajo de 200 ms.

TCP obtiene *M*, y con esta medida actualiza las variables de la conexión a través de las siguientes ecuaciones (Ecuación 10 a Ecuación 14).

$$\text{mdev} = \text{mdev}\left(\frac{3}{4}\right) + |\text{M} - \text{srtt}|\left(\frac{1}{4}\right) \quad \textbf{Ecuación 10}$$

$$\text{mdev\_max} = \max(\text{mdev\_max}, \text{mdev}) \quad \textbf{Ecuación 11}$$

$$\text{srtt} = \text{srtt}\left(\frac{7}{8}\right) + \text{M}\left(\frac{1}{8}\right) \quad \textbf{Ecuación 12}$$

$$\text{rttvar} = \text{mdev\_max} \quad \textbf{Ecuación 13}$$

$$\text{RTO} = \text{srtt} + 4(\text{rttvar}) \quad \textbf{Ecuación 14}$$

El método estándar utiliza un factor 4 en el rttvar (Ecuación 5) lo que produce que el valor de RTO aumente incluso si las mediciones de RTT están disminuyendo. Para solucionar este problema, Linux le da menos peso a la nueva muestra si este valor está por debajo del rango inferior de los valores estimados de RTT (srtt - mdev). La relación completa es la siguiente:

```
if (m < (srtt – mdev))
        mdev = (31/32) * mdev + (1/32) * |srtt – m|
else
        mdev = (3/4) * mdev + (1/4) * |srtt – m|
```

De esta forma, si la nueva muestra indica que la conexión puede estar experimentando una reducción significativa del RTT, la nueva desviación media |*srtt* - *m*| recibe un peso 8 veces menor en comparación con su ponderación normal y, de esta manera, se evita aumentar el valor de RTO (pues aumenta *mdev* y rttvar).

A medida que una conexión avanza y se transmiten segmentos, TCP ajusta variables de estado de acuerdo a las características de la ruta de red entre el emisor y el receptor. Históricamente, estos valores se pierden cuando se cierra la conexión, sin embargo, las implementaciones de TCP más recientes mantienen muchas de estas métricas incluso después de cerrar las conexiones. Así cuando se establece una nueva conexión, TCP consulta para ver si hay alguna información preexistente de la ruta al host de destino con el

que se va a comunicar. De este modo, los valores iniciales para srtt y rttvar se pueden inicializar a un valor basado en una experiencia anterior relativamente reciente.

Una vez que el TCP emisor estableció un valor de RTO basado en las mediciones de RTT, cada vez que envía un nuevo segmento, ajusta un temporizador de retransmisión con este valor de RTO y el número de secuencia del segmento enviado. Si se recibe el ACK a tiempo, se cancela el temporizador. Si no se pierde ningún dato, ningún temporizador de retransmisión expira. Sin embargo, si no se recibe el ACK dentro del tiempo RTO, TCP realiza una retransmisión basada en temporizador, lo cual considera como un evento importante y reacciona reduciendo rápidamente la velocidad con la que envía los datos a la red [9].

Cuando arriba un segmento al receptor, este genera un acuse de recibo indicando el próximo segmento que está preparado para recibir. Sin embargo, si se pierde un segmento intermedio y arriban los siguientes, para cada uno de ellos generara un acuse, siempre indicando al segmento perdido. Puede darse el caso que el emisor reciba estos ACK duplicados y aún no haya caducado el temporizador de retransmisión. Un ACK duplicado que llega al emisor, es un indicador potencial de que se ha perdido un segmento. TCP puede deducir las pérdidas en base al análisis de los ACK acumulativos recibidos que no avanzan en el número de secuencia, a lo largo del tiempo. Para mejorar el rendimiento, el emisor puede inducir una retransmisión de segmentos basado en la retroalimentación del receptor en lugar de requerir que expire un temporizador de retransmisión. A este procedimiento se lo denomina Fast Retransmit (RFC5681) [9].

Con la implementación de la opción de acuse de recibo selectivo o Selective Acknowledgement (SACK, RFC2018), el receptor es capaz de detallar los números de secuencia de datos que recibió correctamente, más allá del ACK acumulativo del campo Numero de ACK que envía en el encabezado TCP. De esta manera, el TCP emisor puede retransmitir todos los datos faltantes, sin esperar los *timeouts* ni ACK duplicados.

1.2.5. Gestión de Ventanas y Control de Flujo

Cuando un receptor es demasiado lento en relación al emisor, se utiliza el control de flujo que obliga al emisor a disminuir la tasa de envío de paquetes. El control de flujo se basa en la técnica de ventanas deslizantes (Sliding Window). Su tamaño no es fijo, sino que se le permite variar en el tiempo. Para implementar esta técnica de control de flujo, el receptor le debe informar al emisor el tamaño de la ventana a utilizar (Window Advertisement, Window Update). Esta actualización la utiliza el emisor para ajustar la tasa de envío y el tamaño de la ventana. Este enfoque funciona bien para el receptor. Sin embargo, los nodos intermedios pueden tener mayores limitaciones de memoria, de procesamiento, ancho de banda de los enlaces, etc. De esta forma, puede suceder que la ventana del emisor no sea lo suficientemente pequeña para algún elemento intermedio en el camino desde el emisor al receptor. Esto lleva a una forma especial de control de flujo, denominada control de congestión, que implica que el emisor disminuya la tasa de envío, para no saturar la red entre él y el receptor. Sin embargo, mientras que en control de flujo hay una indicación explícita del receptor de reducir la tasa del envío, en el caso de control de congestión la señalización del problema es implícita, es decir que el emisor debe reducir su tasa de emisión deduciendo el estado de los nodos intermedios de la red [11].

TCP realiza control del flujo y de congestión para realizar un uso eficiente de ancho de banda de la red y de los recursos. De esta forma garantiza que los datos se transmitan a la velocidad compatible con las capacidades tanto del receptor, como de los enlaces intermedios del camino extremo a extremo. El control de flujo es una función orientada al host que intenta evitar que un emisor rápido sobrecargue o sature un receptor lento. Para administrar la cantidad de datos que se envían a la vez, TCP utiliza el denominado algoritmo de ventana deslizante. El tamaño de la ventana establece el número máximo de segmentos TCP que pueden ser transmitidos sin haber sido aún reconocidos. El Throughput queda determinado por la evolución dinámica del tamaño de esta ventana W en bytes y el valor de RTT de la siguiente forma (Ecuación 15):

$$\text{Throughput} = \frac{\text{W}}{\text{RTT}}$$ **Ecuación 15**

El principal objetivo del control de flujo del lado del receptor es proporcionar información sobre el espacio disponible en el buffer de recepción, mientras que el principal objetivo del control de flujo del lado del emisor es limitar el flujo de datos en respuesta a esta información del receptor.

En cada segmento TCP de datos, el encabezado incluye un número de secuencia válido, un número de ACK y un campo Window Size, que contiene el anuncio de ventana o Ventana de Recepción (rwnd). Este valor trasmitido en el campo Window Size, representa la cantidad de datos que el remitente del segmento tiene disponible para almacenar.

El campo *Tamaño de Ventana* en cada cabecera TCP indica la cantidad de espacio vacío, en bytes, que queda en el buffer de recepción. El campo es de 16 bits en TCP, pero con la opción *Escala de Ventana* se pueden usar valores mayores que 65.535. El número de secuencia más grande que el remitente de un segmento está dispuesto a aceptar en la dirección inversa es igual a la suma de los campos Número de ACK y Tamaño de ventana en el encabezado TCP. Para cada conexión activa, cada extremo TCP mantiene una ventana de envío y una ventana de recepción. En la siguiente figura (Figura 7) se observa una ventana de envío de TCP [9].

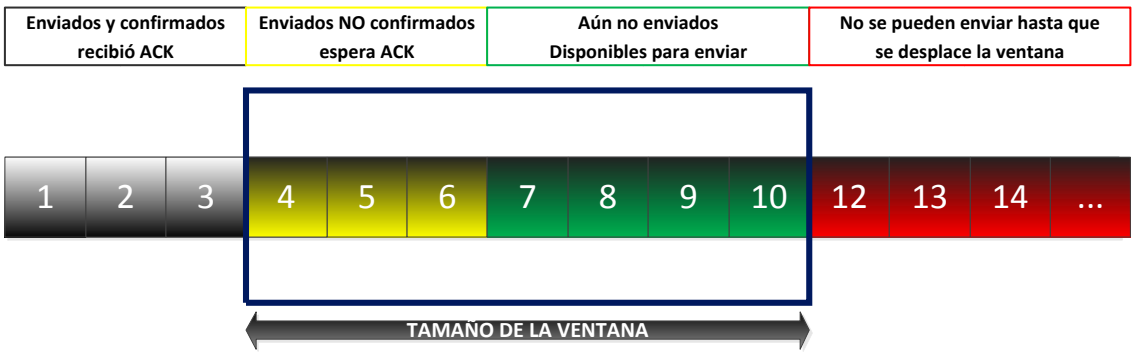

Figura 7: Ventana deslizante (emisor) [11]

La ventana deslizante del lado del emisor mantiene los números de secuencia de los segmentos con acuse de recibo, los que fueron enviados y se espera ACK y los que aún no se enviaron. En la Figura 7 se numeran los bytes de 1 a 14, mientras que el tamaño de ventana ofrecida por el receptor abarca los

bytes de 4 a 10, lo que significa que el receptor ya envió los acuses de recibido (ACK) de todos los bytes hasta el 3 (último valor de ACK, 4) y anunció un tamaño de ventana de 7 bytes. TCP emisor calcula su ventana utilizable, que es la cantidad de datos que puede enviar de inmediato, esto es la ventana ofrecida menos la cantidad de datos ya enviados, pero aún no reconocidos. Debido a que cada segmento TCP contiene un número ACK y un anuncio de ventana, el emisor TCP ajusta la estructura de la ventana basada en ambos valores cada vez que llega un segmento.

En la mayoría de los casos, el protocolo TCP no envía un ACK por cada segmento entrante, sino que lo retrasa intencionalmente aprovechando que el campo ACK es acumulativo (Delayed Acknowledgement). Obviamente, un TCP no puede retrasar los ACK indefinidamente porque el emisor podría concluir que se han perdido datos e iniciar una retransmisión innecesaria. La RFC1122 establece que el protocolo TCP debe implementar un retraso ACK inferior a 500 ms. Al retrasar los ACK se logra que menos paquetes se inyecten en la red. Cuando se utilizan paquetes más pequeños, como en las aplicaciones interactivas, se puede producir un rendimiento deficiente. Al intercambiarse estos pequeños segmentos, se envían pocos bytes de datos en comparación con el número de bytes de los encabezados, lo que degrada el rendimiento. Como TCP realiza el control de flujo mediante ventanas y no utiliza segmentos de tamaño fijo, puede ser víctima de una condición conocida como *Síndrome de la Ventana Tonta* o *SWS (Silly Window Syndrome)* [8], que se refiere a la situación cuando se intercambian segmentos mucho más pequeños, que lo permitido por el MSS a través de la conexión TCP. Si el receptor puede anunciar un tamaño de ventanas pequeño, el emisor sólo puede transmitir segmentos pequeños, lo que empeora el rendimiento de la conexión. La transferencia de segmentos pequeños utiliza ancho de banda adicional debido a la sobrecarga (*Overhead*) de cabecera de protocolo. Estos pequeños paquetes (llamados *Tinygrams*) tienen una sobrecarga relativamente alta para la red, pues se inyectan muchos bytes a la red para enviar pocos bytes de datos [9]. Para aumentar la eficiencia de la transmisión de datos, TCP evita el envío y la recepción de segmentos pequeños utilizando el algoritmo Nagle [8] (John Nagle) y el algoritmo especificado por RFC1122 para evitar la condición

de SWS. El algoritmo de Nagle permite que pequeñas cantidades de datos se acumulen en el buffer y no se envíen hasta que se reciba un ACK para los datos enviados previamente. El algoritmo de Nagle establece que, cuando en una conexión TCP se tienen datos pendientes que aún no se han reconocido (ACK), los segmentos pequeños (de menor tamaño que SMSS) no se pueden enviar hasta que todos los datos pendientes tengan su acuse de recibo. En forma análoga, el algoritmo para evitar la condición de SWS previene los anuncios de ventanas pequeñas en el caso en que un proceso lea los datos desde el buffer lentamente. De esta manera, establece que el receptor TCP espere hasta que el espacio disponible alcance el 50% del tamaño del buffer o un MSS, lo que sea menor. El campo de ventana en el encabezado TCP consta de 16 bits, por lo que el tamaño máximo *rwnd* es 65.535 bytes. En consecuencia, el emisor TCP puede tener sólo 65.535 bytes de datos en tránsito a la vez, lo que limita el rendimiento máximo alcanzable. Para aumentar el rendimiento en redes de larga distancia y alta velocidad la opción de escala de ventana TCP permite al receptor TCP anunciar un tamaño de ventana mayor. El factor de escala de ventana es simplemente un multiplicador de potencia de dos para ser aplicado a la ventana anunciada.

En la siguiente tabla (Tabla 2) se puede observar algunas de las funciones e implementaciones de TCP y el RFC en el cual se definió.

FUNCION	IMPLEMENTACION	RFC
Transferencia de datos	Establecimiento y teminación de conexión	793, 1122
	Opción MSS	879
	Path MTU Discovery	1191
Multiplexación	Número de puertos	793, 1122, 1700
Control de Flujo	Ventana de Recepción	793, 1122
	Síndrome de Ventana Tonta	813, 1122
	Algoritmo de Nagle	896, 1122
	Opción Window Scale	1323
Control de Errores	Checksum	793, 1071
	Números de secuencia	793
	Protección contra números de secuencia	1323
	ACK acumulados y selectivos	1122, 2018, 2883
	Retransmisiones y temporizador de retransmisiones	1122, 2988
Control de Comgestión	Algoritmo de Karn	2988
	Ventana Inicial	3390
	Slow Start	2581
	Congestion avoidance	2581
	Fast retransmit y fast recovery	2581, 3782, 3517
	ECN	3168

Tabla 2: Resumen de funciones TCP y RFC

CAPITULO 2 - Control de Congestión

La congestión es uno de los principales problemas que se afronta en la transmisión de datos, e implica que se saturan los recursos de la red, degradándose su utilización. Por esta razón, uno de los aspectos más importantes, desde el punto de vista del rendimiento del protocolo TCP, es el control de la congestión. Este protocolo utiliza varios mecanismos para alcanzar un alto rendimiento y evitar un colapso. Los algoritmos de control de la congestión permiten adaptar la tasa de envío en forma dinámica, evitando saturar la capacidad de la red y las potenciales situaciones de congestión.

Actualmente no existe ningún enfoque de control de congestión para TCP que pueda aplicarse universalmente a todos los entornos de red. Una de las principales causas es la amplia variedad de entornos y los diferentes puntos de vista con respecto a qué parámetros deben optimizarse.

Los enfoques de investigación han cambiado con el desarrollo de Internet, desde el problema básico de eliminar el fenómeno del colapso de la congestión hasta los problemas de utilizar los recursos de red disponibles de manera eficiente en diferentes tipos de entornos. La primera propuesta que intenta resolver estos problemas es Tahoe, que introduce la técnica básica de probar gradualmente los recursos de red y confiar en la pérdida de paquetes para detectar que se ha alcanzado el límite de la red.

Algunas soluciones al problema de eficiencia incluyen algoritmos que refinan el principio de control de la congestión al hacer suposiciones más optimistas

sobre la red; o bien, refinar el protocolo TCP para incluir capacidades de informes extendidos del receptor, lo que permite al emisor estimar el estado de la red de manera más precisa. Otro enfoque es introducir métricas alternativas para la estimación del estado de la red, recurriendo a métodos proactivos, basados en la estimación de la longitud de la cola observando de retardo.

Otro grupo de propuestas de control de la congestión, son los que se centran en entornos donde los paquetes se reordenan con frecuencia y llegan fuera de orden al receptor. Estas propuestas muestran que, en tales entornos, la eficiencia puede mejorarse significativamente retrasando las acciones de control, o deshaciendo las acciones aplicadas previamente si se detecta el reordenamiento.

Otro grupo, apunta a resolver el problema de la baja utilización de los canales de redes con valores altos del producto de velocidad y latencia por flujos TCP. Las primeras propuestas que abordan este problema introdujeron políticas simples, pero altamente optimistas para sondear las redes en busca de los recursos disponibles. Desafortunadamente, tales técnicas llevaron a la aparición de una serie de nuevos problemas, incluida la falta de equidad. Las propuestas posteriores emplearon técnicas más inteligentes para hacer que el control de la congestión sea agresivo sólo cuando la red se considera libre de congestión y conservadora durante un estado de congestión. Dos propuestas usan la pérdida de paquetes para establecer el límite de recursos de red. Otro grupo de propuestas sondean los recursos de la red, basándose en técnicas secundarias de estimación del estado de la red basadas en el retraso. Desafortunadamente, existen desventajas para ambos enfoques, y no existe un consenso actual en la comunidad de investigación sobre qué enfoque es superior. No es sorprendente que en la Internet actual coexistan diferentes variantes de TCP, como C-TCP, que se implementa en el mundo de Windows, y CUBIC utilizada en entornos Linux.

2.1. Congestión

Cada nodo intermedio en una red de datos, tiene una cola de paquetes asociada a cada uno de los enlaces a los cuales está conectado. Si la tasa a la que se reciben los paquetes supera a la tasa a la que los puede transmitir, el

tamaño de la cola crece y, por lo tanto, también crece el retardo. Dado que el tamaño de las colas es finito, y si comienza a crecer, en algún momento se producirá su desbordamiento. A esta situación, en donde el nodo no puede manejar la tasa de tráfico entrante y comienza a descartar paquetes se la llama congestión. Por lo tanto, el problema de la congestión comienza cuando el número de paquetes en la red se aproxima al límite de su capacidad de gestión.

La congestión, por lo tanto, tiene dos efectos. En primer lugar, cuando la congestión empieza a producirse, el tiempo de transmisión a través de la red aumenta. En segundo lugar, conforme la congestión se hace más severa, los nodos de la red descartan paquetes. El objetivo del control de congestión es mantener el número de paquetes en la red por debajo de ese nivel, sin descuidar el rendimiento.

En la siguiente figura (Figura 8) [11] se observa el resultado de aumentar la carga en la red sin implementar un control de congestión.

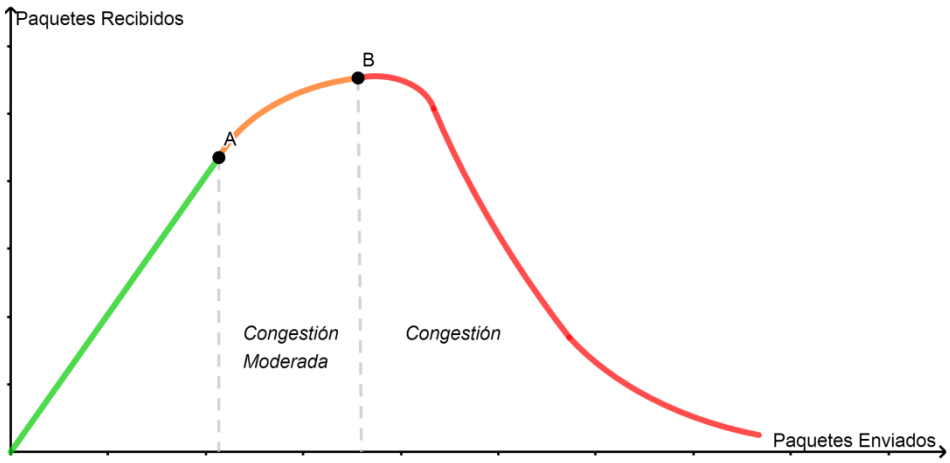

Figura 8: Paquetes Enviados vs. Paquetes Recibidos

Para baja carga, cuando la cantidad de paquetes inyectados en la red es pequeña, el rendimiento, y por tanto la utilización de la red, aumenta conforme lo hace la carga. A medida que ésta continúa creciendo, llega un momento (punto A en la Figura 8) a partir del cual, el rendimiento de la red crece a una tasa menor de lo que lo hace la carga. Este hecho se debe a que la red entra en un estado de congestión moderada, en el cual la red sigue dando curso al tráfico, aunque con un incremento significativo del retardo.

A medida que la carga de la red continúa aumentando, y se inyectan más paquetes, el tamaño de las colas de los distintos nodos sigue creciendo. Eventualmente, llega un momento (punto B en la Figura 8) a partir del cual el rendimiento real decae al aumentar la carga de entrada. Cuando los buffers de un nodo se llenan, éstos deben descartar paquetes. Por tanto, los emisores deben retransmitir los paquetes descartados, además de inyectar los nuevos. Esto sólo consigue empeorar la situación: conforme se retransmiten más y más paquetes, la carga del sistema aumenta y se saturarán más buffers. Mientras el sistema trata desesperadamente de eliminar el exceso de paquetes, los usuarios continúan enviando paquetes al sistema. En estas circunstancias, la capacidad efectiva del sistema puede caer hasta cero [11].

Para mitigar esta situación, cada TCP implementa diversos procedimientos de control de congestión, lo que implica el uso inteligente y eficiente de los recursos de las redes de conmutación de paquetes y así evitar el colapso de congestión. Cada TCP debe mantener el número de paquetes en tránsito en la red por debajo del nivel para el que decaen dramáticamente las prestaciones. El objetivo es lograr que la red sea utilizada eficientemente y, al mismo tiempo, en forma equitativa.

El control de congestión es un conjunto de comportamientos, determinados por algoritmos, que cada TCP implementa para evitar que la red se vea sobrecargada por un tráfico inyectado demasiado grande. El reto es determinar exactamente cuándo TCP debe reducir la tasa de envío de paquetes y cuándo puede aumentarla nuevamente.

Cuando está presente la congestión o está a punto de producirse, es deseable que los TCP emisores disminuyan la tasa de inyección de paquetes y, cuando disminuye la congestión, detectar y utilizar en forma adecuada el ancho de banda disponible. Esto no resulta sencillo, pues no hay ninguna notificación explícita de los nodos intermedios que pueda ayudar a tomar estas decisiones, es decir no hay señalización expresa de congestión.

Para que TCP pueda reaccionar a la congestión primero debe conocer de su existencia. TCP supone que hay congestión cuando se pierde un paquete, pues en las redes cableadas las pérdidas de paquetes se deben casi exclusivamente a los eventos de congestión, resultando casi nula la pérdida en

tránsito. Si bien este ha sido el enfoque desde principios de los 80, con el uso generalizado de las redes inalámbricas, los errores de transmisión y recepción se convierten en una causa importante de pérdida de paquetes. Determinar si la pérdida se debe a congestión o errores de transmisión no es trivial, y ha sido tema constante de investigación.

Cuando TCP detecta que se perdió un paquete, además de retransmitirlo, interpreta que existe congestión y debe disminuir su tasa de envío. De esta manera, se debe lograr que el TCP emisor envíe los datos a una tasa lo más cercana a la que el TCP receptor o, la red, pueden manejar, lo que sea menor. Para determinar dinámicamente la capacidad del receptor, se utiliza el campo *Tamaño de Ventana* del encabezado TCP. Análogamente, se denomina *Ventana de Congestión (cwnd)* al valor utilizado para mantener la estimación de la capacidad disponible de la red. Entonces la ventana utilizable por el TCP emisor se define como el valor mínimo entre la ventana anunciada por el receptor (rwnd) y la ventana de congestión (cwnd), como se describe en la siguiente Ecuación 16:

$$W=min(rwnd,cwnd) \qquad \textbf{Ecuación 16}$$

La anterior ecuación implica que el TCP emisor no puede tener más de W paquetes o bytes sin acuse de recibo, pendientes en la red. Sin embargo, la estimación del valor de W no es trivial pues, rwnd y cwnd cambian dinámicamente debido a las condiciones del receptor y la red. Además, el valor de cwnd generalmente no está determinado por señales explícitas, por lo que debe establecerse de manera empírica y actualizarse dinámicamente en el TCP emisor [9].

Es deseable que el valor de W sea el óptimo, instante a instante, para cada una de las condiciones de la red y del receptor. Se puede demostrar que el rendimiento máximo de un flujo TCP depende directamente del tamaño de la ventana de congestión e inversamente del tiempo de ida y vuelta (RTT) de la ruta. Sin embargo, si la ventana de congestión es demasiado grande, crece la probabilidad de pérdida de paquetes, dado que la red y el receptor tienen recursos finitos, lo que limita la cantidad de tráfico que pueden manejar. Para minimizar las pérdidas de paquetes, se requiere minimizar la ventana W. Por lo

tanto, el problema se reduce a encontrar un valor óptimo de la ventana de congestión que proporcione un buen rendimiento, pero no sobrecargue ni a la red y ni al host receptor.

Además, TCP debe ser capaz de recuperarse de las pérdidas de paquetes, lo que significa que cuanto más corto sea el intervalo entre la transmisión de paquetes y la detección de pérdidas, más rápido puede recuperarse. Sin embargo, este intervalo no puede ser demasiado corto, o de lo contrario, el TCP emisor puede detectar una pérdida prematuramente y retransmitir el paquete innecesariamente. Esta sobrerreacción simplemente sobrecarga los recursos de la red y puede incrementar el estado de congestión. Por lo tanto, cuándo y cómo un remitente detecta las pérdidas de paquetes es uno de los problemas complejos que debe articular TCP.

Los algoritmos de control de congestión deberían proporcionar un uso eficiente de los recursos de la red, responder rápidamente a sus cambios y ser equitativos con otros flujos presentes en la red. Esto significa que cada flujo comparta una porción justa de los recursos.

El concepto de equidad se puede dividir, en general, en tres categorías:

- Equidad IntraProtocolo (Intra-Protocol Fairness) característica de la distribución de recursos entre los flujos que ejecutan el mismo algoritmo de control de la congestión en el mismo entorno de red;
- Equidad Inter-Protocolo (Inter-Protocol Fairness) característica de imparcialidad de la distribución entre flujos que ejecutan diferentes algoritmos en el mismo entorno;
- Equidad RTT (RTT-Fairness) característica de la distribución de recursos entre flujos que comparten el mismo vínculo de cuello de botella pero que tienen diferentes valores de RTT.

La equidad es una medida de cómo un flujo TCP afecta al resto de los flujos, y cómo ese flujo es afectado por el resto en términos de Throughput. Las medidas de equidad, en general, no tienen en cuenta las características de ruta de la red. Sin embargo, esto no implica que no se vea afectada y que no se puedan utilizar para medir el efecto de las diferentes particularidades de ruta sobre la equidad entre distintos flujos.

2.2. Control de Congestión TCP

La técnica de control de congestión basada en ventanas utilizada por TCP intentará regular la tasa de envío de datos ajustando el tamaño de la ventana para evitar la congestión de la red y, al mismo tiempo, proporcionar una parte justa del ancho de banda de la red a todas las conexiones. Aunque la Ecuación 16 establece la tasa de envío como el valor mínimo entre la ventana anunciada por el receptor y la ventana de congestión, en la práctica se considera solamente el tamaño de la ventana de congestión como factor que limita la velocidad. Esto supone que, en la mayoría de los casos, la tasa de procesamiento de los hosts finales es varios órdenes de magnitud superior a la tasa de transferencia de datos que la red.

Una de las técnicas de control de congestión de TCP utiliza los algoritmos Slow Start, Congestion Avoidance, Fast Retransmit y Fast Recovery. Las primeras dos fases del mecanismo de control de la congestión (SS y CA) regulan la cantidad de paquetes inyectados en la conexión y son responsables de la detección de la congestión, mientras que los otros dos algoritmos reaccionan para superar la congestión y proporcionar un mecanismo que acelera la recuperación de la conexión [12].

En la siguiente figura (Figura 9), se puede observar el funcionamiento de los distintos algoritmos a través de la evolución del tamaño de la ventana de congestión en función del tiempo.

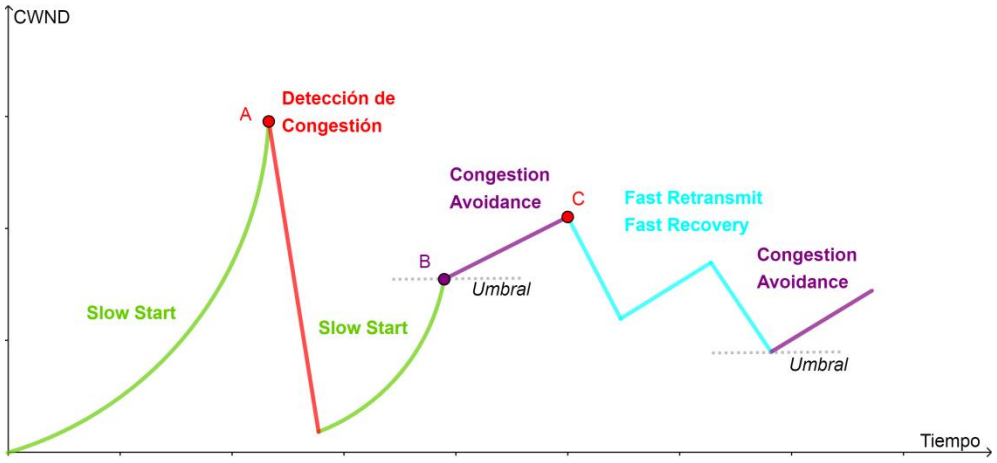

Figura 9: Tamaño de la Ventana de Congestión vs. Tiempo

2.2.1. Slow Start

Es uno de los algoritmos que TCP utiliza para controlar la congestión dentro de la red, también conocido como fase de crecimiento exponencial. Este algoritmo se dispara al comienzo de todas las conexiones TCP y su objetivo es encontrar el máximo ancho de banda disponible. Al incrementar en forma exponencial, el tamaño de la ventana de congestión intenta encontrar lo más rápido posible la capacidad de un canal de transmisión.

Durante esta fase, el algoritmo lo que hace es incrementar el tamaño de la ventana en el mismo número de segmentos con acuse de recibo, por cada ACK recibido. Este proceso duplica la tasa de envío por cada RTT. Esto significa que la velocidad de transmisión del emisor TCP depende completamente de los acuses de recibo (ACK) devueltos por el receptor TCP.

La fase de Slow Start puede representar un fragmento del control de la congestión, pero puede afectar considerablemente el comportamiento y el rendimiento de la red como un todo.

Como se observa en la Figura 9, cuando se inicia la conexión, cwnd se ajusta en 1 luego, el emisor envía un segmento y, con cada nuevo ACK recibido con éxito, para cada RTT, cwnd aumenta su tamaño en forma exponencial. Actualmente se observan algunas implementaciones que ajustan el tamaño inicial en 2 o 3 MSS.

Esto continúa hasta que no se recibe un ACK de algún segmento (punto A en la Figura 9) o bien, hasta que el tamaño de la ventana alcanza un umbral predeterminado (punto B en la Figura 9). Si se pierde un segmento, TCP asume que es debido a la congestión en la red y toma medidas para reducir la carga inyectada. Por el contrario, si se ha alcanzado un umbral preestablecido, denominado *umbral de Slow Start (ssthresh)*, TCP abandona la fase de Slow Start y entra en una fase de crecimiento lineal (a partir del punto B, Figura 9) del tamaño de la ventana de congestión llamada Congestion Avoidance. En este punto, el tamaño de la ventana se incrementa de a un segmento.

```
//**_____**//
// **Algoritmo Slow Start** //
cwnd = 1; /** valor inicial **/
       For (cada acuse de recibo ACK)
```

```
                cwnd++;
        Until (evento de congestión; or cwnd > ssthresh;)
//**_____________________________________________________________**//
```

2.2.2. Congestion Avoidance

Si la ventana del receptor es lo suficientemente grande, el crecimiento cuasi exponencial de Slow Start hará crecer la ventana de congestión al punto que se comenzará a descartar paquetes. Para encontrar la capacidad disponible de la red, TCP pasa al algoritmo de Congestion Avoidance. Cuando el tamaño de la ventana alcanza el valor umbral denominado ssthresh (umbral de Slow Start), el emisor TCP cambia al algoritmo Congestion Avoidance para regir el crecimiento de la ventana (a partir del punto B, Figura 9). Este algoritmo reduce la tasa de aumento del tamaño de la ventana, desde el crecimiento exponencial (Slow Start) hasta el crecimiento lineal (Congestion Avoidance). En esta fase, en lugar de duplicar el valor de cwnd cada RTT, TCP adopta un enfoque más conservador y aumenta el valor de cwnd en un solo segmento cada RTT (RFC5681).

Esta fase continúa hasta que se detecta un evento de congestión, por lo cual el TCP emisor debe reducir su tasa de envío de paquetes. Si el evento se detecta por la caducidad de un temporizador: se reduce el valor de cwnd a 1 y se define el valor de ssthresh a la mitad del valor alcanzado por cwnd o al menos dos segmentos. Sin embargo, si el evento se detecta por la recepción de ACK duplicados, (punto C en la Figura 9), se puede deducir que la congestión no es tan severa pues la red continúa entregando segmentos y, por lo tanto, el comportamiento de TCP debe ser menos drástico: divide a la mitad el valor de cwnd y le suma 3 de los ACK duplicados recibidos luego, establece el valor de ssthresh a la mitad del valor de cwnd y se invoca a los algoritmos de Fast Retransmit y Fast Recovery.

```
//**_____________________________________________________________  **//
// **Algoritmo Congestion Avoidance** //
/** finalizado slow start y cwnd > ssthresh **/
        For (cada acuse de recibo ACK)
                cwnd = cwnd + (1=cwnd);
        Until (caducidad de un temporizador; or 3 DupACK;)
//**_____________________________________________________________  **//
```

2.2.3. Fast Retransmit

Si se recibe un segmento fuera de orden, el receptor genera lo que se denomina un ACK duplicado. Dado que no es posible saber si el acuse de recibo duplicado fue causado por un segmento perdido o simplemente por el reordenamiento de segmentos, el emisor espera tres acuses de recibo duplicados (3 DupACK) antes de retransmitir el segmento. Si este límite hubiera sido menor, esto aumentaría la probabilidad de que los segmentos reordenados crearan duplicados y se transmitieran innecesariamente.

La ventaja de este mecanismo es que TCP no tiene que esperar a que expire el temporizador de retransmisión. Simplemente asume que tres acuses de recibo duplicados son un buen indicador de un segmento perdido. El efecto de esto es reducir el tiempo para producir la retransmisión del segmento. De esta manera, el objetivo es evitar que el TCP emisor no tenga que esperar el temporizador de retransmisión para reenviar el segmento perdido. Se describió en las modificaciones al algoritmo de Congestion Avoidance propuestas en el RFC2001 [13].

Si la congestión se indicó mediante ACK duplicados, el remitente de TCP entra en el modo de Fast Retransmit para retransmitir lo que parece ser un paquete perdido sin esperar a que expire el temporizador de retransmisión. Este proceso acelerará la recuperación de las pérdidas del segmento y, TCP intenta mantener la velocidad de flujo existente sin volver a Slow Start. Luego, el emisor establece ssthresh en la mitad de la ventana de congestión actual y ajusta el valor de la ventana de congestión en la nueva ssthresh más la cantidad de confirmaciones duplicadas recibidas y entra en la fase de Fast Recovery.

2.2.4. Fast Recovery

TCP realiza una Fast Retransmit y entra en el estado de Fast Recovery. Fast Recovery (RFC5681) [14] es un mecanismo que reemplaza a Slow Start cuando se dispara Fast Retransmit. Si bien la recepción de los ACK duplicados indica que se ha perdido un segmento, también indica que los paquetes siguen fluyendo entre emisor y receptor, y puede suponer que se ha descartado un

único paquete y que la red no está completamente congestionada. Por lo tanto, el remitente TCP no necesita volver al modo Slow Start, sino a la fase de Congestion Avoidance con la mitad de la tasa anterior. Durante esta fase, el valor de cwnd se incrementa en 1 por cada ACK duplicado recibido. Eventualmente, cuando llega el ACK del segmento faltante, TCP entra en el estado Congestion Avoidance, después de reducir cwnd. Si se produce un evento de timeout, Fast Recovery cambia al estado de Slow Start después de realizar las mismas acciones: el valor de cwnd se establece en 1 y el valor de ssthresh se establece en la mitad del valor de cwnd cuando ocurrió el evento de pérdida. Si no hubo evento de timeout, después de la fase Fast Retransmit se pasa a la fase de Congestion Avoidance [15].

Fast Recovery permite un mayor rendimiento bajo congestión, especialmente cuando se utilizan grandes ventanas de congestión. Recibir tres acuses de recibo duplicados le dice a TCP más que la expiración del temporizador de retransmisión. Al utilizar este enfoque, evitando la fase de Slow Start, el TCP no reduce la velocidad de transferencia y mejora el rendimiento [12].

```
//**_____________________________________________________________**//
//** Algortimo Fast Recovery **//
/** después de la fase de Fast Transmit **/
/** NO entrar en la fase de Slow Start **/
ssthresh = cwnd / 2;
cwnd = ssthresh + 3;
       For (cada ACK duplicado recibido)
              cwnd++;
              enviar un segmento si cwnd lo permite;
       For (cada ACK recibido)
              If (ACK recibido parcial?)
                     Quedarse en Fast Recovery;
                     Retransmitir el próximo segmento perdido (1 por RTT);
              If (ACK recibido completo?)
                     cwnd = ssthresh;
                     Regresar a Congestion Avoidance;
       For (caducidad de un temporizador)
              ssthresh = cwnd / 2;
cwnd = 1;
Invocar al Algoritmo Slow Start;
//**_____________________________________________________________**//
```

En la siguiente figura (Figura 10) se observa la máquina de estados finitos del control de congestión de TCP [7], con sus distintas fases y transiciones.

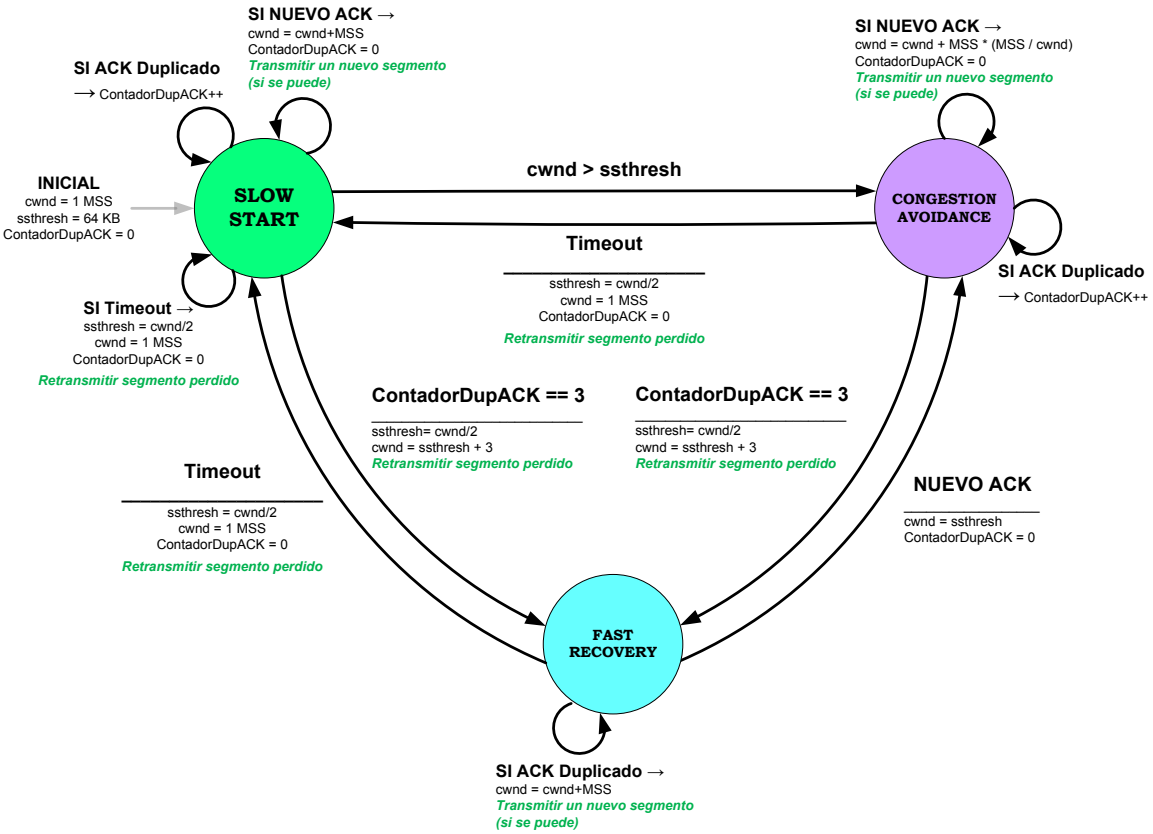

Figura 10: Control de Congestión. Máquina de Estados [7]

Cuando el TCP emisor se encuentra en cualquiera de las fases de Slow Start o Congestion Avoidance y recibe tres ACK duplicados dispara los algoritmos de Fast Retransmit y Fast Recovery (Figura 10). A continuación, se detallan las acciones y transiciones realizadas por el TCP,

- Cuando se recibe el tercer ACK duplicado, se configura ssthresh en la mitad de la cantidad de datos pendientes en la red ($\frac{cwnd}{2}$), pero al menos 2 veces MSS.
- Se retransmite el segmento perdido y se establece el valor de cwnd en ssthresh más tres MSS. Incrementa artificialmente la ventana de congestión en la cantidad de tres segmentos que han salido de la red y que el receptor ha confirmado.
- Por cada ACK duplicado adicional recibido, se incrementa el tamaño de cwnd. Esto incrementa artificialmente el tamaño de la ventana de congestión para reflejar el segmento adicional que ha abandonado la red.

- Se transmite un segmento, si lo permite el nuevo valor de cwnd y la ventana anunciada del receptor.
- Cuando se recibe el próximo ACK que reconoce nuevos datos, se configura el valor de cwnd igual al valor de ssthresh. Este ACK debe confirmar todos los segmentos intermedios enviados entre el segmento perdido y la recepción del tercer ACK duplicado.
- Si el ACK que reconoce datos nuevos no llega; se produce la caducidad del temporizador de retransmisión (timeout), y el emisor pasa del modo de Fast Recovery a la fase de Slow Start.

Aunque lo que constituye el TCP "estándar" está en debate, los algoritmos descriptos constituyen los procedimientos primarios identificados con la operación TCP estándar. Los algoritmos Slow Start y de Congestion Avoidance se implementan generalmente juntos, y el comportamiento general de referencia se da en RFC5681. Esta especificación no requiere el uso de estos algoritmos exactos, pero se impone un requisito que cualquier implementación TCP no sea más agresiva de lo que permitirían estos algoritmos [16].

2.3. Cronología del protocolo TCP

La primera idea de TCP fue dada en 1947 por Khan y Cerf. El avance de la tecnología de red ha dado lugar a muchos problemas y nuevos escenarios, tales como, la subutilización del ancho de banda, la congestión de la red, la retransmisión evitable, la pérdida sin congestión, la entrega fuera de orden y equidad. Debido a esto, el comportamiento del TCP fue revisado por los expertos y se introdujeron muchas variantes de TCP.

Durante las últimas décadas [17], se ha propuesto una gran variedad de modificaciones de TCP, que van desde cambios menores, hasta enfoques de diseño completamente nuevos. Como regla general, estas modificaciones incluyen algoritmos desarrollados específicamente para mejorar el rendimiento de TCP en diferentes escenarios y entornos de red. Sin embargo, alguna de estas modificaciones no son estándares y pueden producir consecuencias no deseadas, especialmente aquellas relacionadas con la estabilidad y la equidad en Internet.

Sin embargo, el mecanismo de control de la congestión se convierte en el factor decisivo para mejorar la eficiencia o el rendimiento de TCP. El algoritmo de control de congestión TCP es el factor clave y desempeña un papel crítico en el nivel de rendimiento en las redes.

TCP se adoptó oficialmente como un estándar RFC793 para tratar el control del flujo, basada en ventanas deslizantes, y la corrección de errores, pero no presentaba ningún esquema de control de congestión. Después de observar una serie de colapsos de congestión en 1980, Jacobson introdujo varios mecanismos innovadores de control de la congestión en TCP en 1988. Esta versión TCP, llamada TCP Tahoe, representa la segunda generación de versiones de TCP y es la primera variante que incorpora un control de congestión que incluye los algoritmos de Slow Start, aumento aditivo y disminución multiplicativa (AIMD Slow Start, additive increase and multiplicative decrease), y Fast Retransmit. Dos años más tarde, el algoritmo de Fast Recovery se agregó a Tahoe para formar una nueva versión TCP llamada TCP Reno. Reno es la tercera versión de la primera serie desarrollada, y está estandarizada en RFC2011 [18].

En las siguientes figuras (Figura 11 y Figura 12) se observa el desarrollo cronológico de algunas versiones del protocolo TCP.

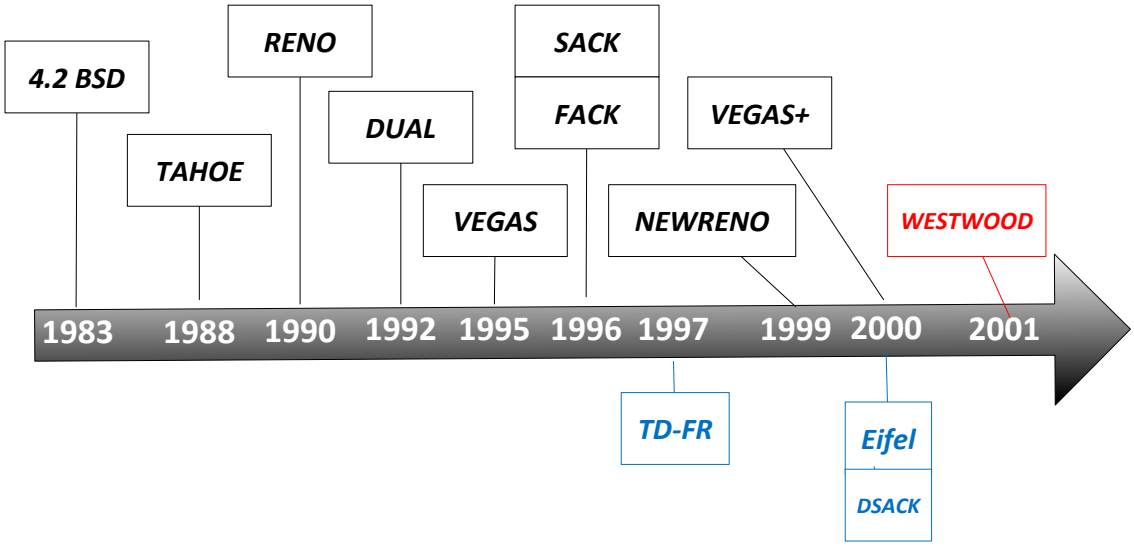

Figura 11: Cronología de versiones de TCP (1983 – 2001)

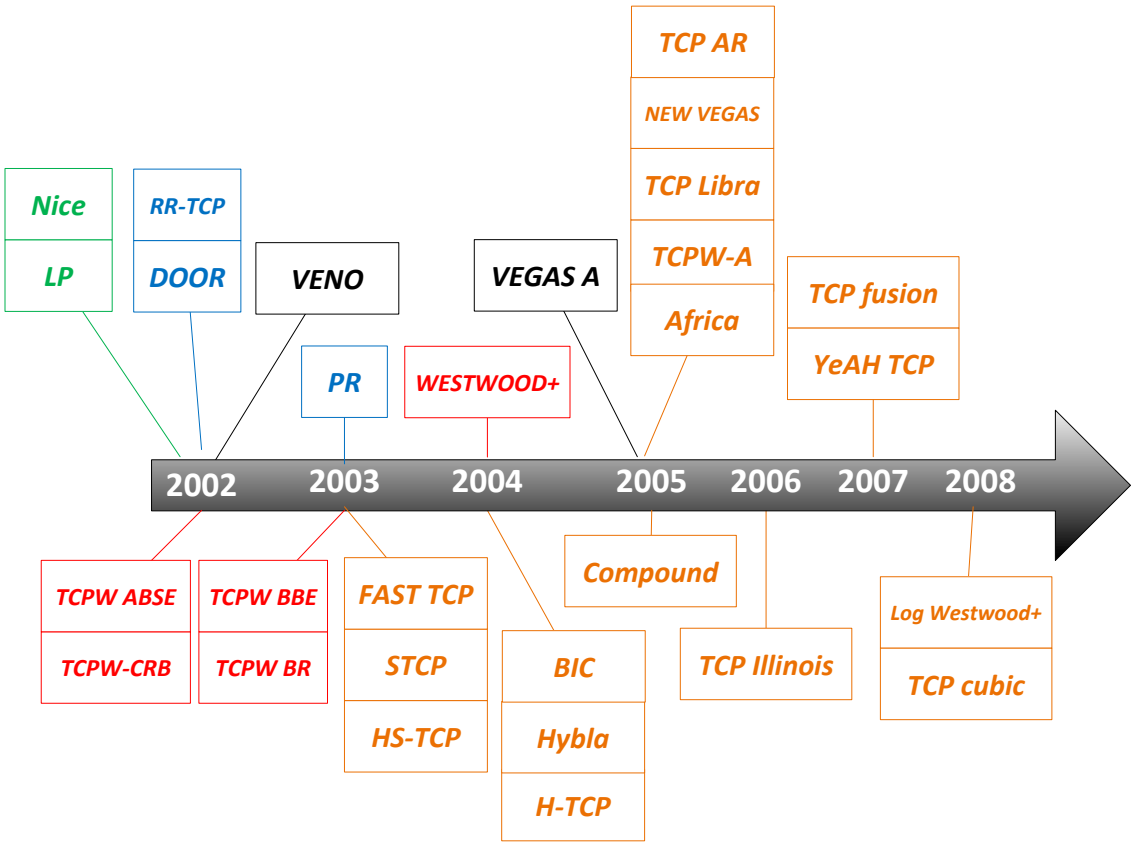

Figura 12: Cronología de versiones de TCP (2001 – 2008)

La técnica de control de congestión basada en ventanas empleada por TCP [19], intentará ajustar la tasa de flujo de datos mediante el tamaño de la ventana de congestión para evitar la congestión de la red y, al mismo tiempo, obtener una parte justa del ancho de banda en todas las conexiones posibles.

Anteriormente, el tráfico de Internet estaba controlado por el estándar TCP, también conocido como TCP Reno. Actualmente, el tráfico de Internet está controlado por múltiples mecanismos de control de la congestión donde se destacan el Compound TCP y TCP CUBIC. TCP Compound es el mecanismo de control de congestión predeterminados en los sistemas operativos Microsoft, aunque Windows XP utilizaba New Reno. TCP CUBIC es el mecanismo de control de congestión predeterminado en los sistemas operativos Linux, Android y Free-BSD.

TCP Reno se puede considerar como un esquema de control de la congestión reactiva. Utiliza la pérdida de paquetes como indicador de congestión. Esta estrategia, produce oscilaciones que pueden degradar el rendimiento. Para aliviar el problema de degradación del rendimiento, muchos investigadores

intentaron refinar los algoritmos de Fast Retransmit y Fast Recovery. Las nuevas propuestas incluyen TCP New Reno, TCP FACK, TCP SACK, D-SACK, FR-TCP, RR-TCP. Todos estos algoritmos proporcionan una mejora en el rendimiento de una conexión después de que se detecta una pérdida de paquetes.

Otro enfoque, se presenta cuando el emisor estima dinámicamente el ancho de banda de red disponible midiendo y promediando en retardo de extremo a extremo. Esta técnica es utilizada por variantes como TCP Vegas y emplea diferentes enfoques de Slow Start, Congestion Avoidance, Fast Retransmit y Fast Recovery para combatir el problema de oscilación inherente de TCP Reno y mitigar el problema de degradación del rendimiento.

TCP Westwood es un enfoque híbrido entre los dos anteriores y logra mejor rendimiento que TCP Reno incluso, logra equidad en determinados escenarios.

El emisor utiliza los ACK devueltos como un "reloj" para determinar cuándo enviar nuevos paquetes a la red. La pérdida de paquetes no es un problema real para una conexión TCP, simplemente indica la aparición de congestión. El verdadero problema es la pérdida de ese “reloj” que trae como consecuencia que caiga el uso de la red. Si se degrada el uso de la red, el TCP emisor tardará varios RTT en restablecer el uso eficiente de los recursos, lo que reduce el rendimiento de TCP. Además, cuando se pierden múltiples paquetes en la misma ventana de datos, en el emisor TCP se puede producir una degradación sustancial del rendimiento.

En cualquier caso, después que se restablece la ventana de congestión, la conexión requiere varios RTT antes que se restaure la utilización de la capacidad de la red. Este problema se agrava cuando ocurren pérdidas aleatorias o esporádicas. Las pérdidas aleatorias se definen como pérdidas no causadas por congestión y, son habituales en los canales inalámbricos. En este caso, una ráfaga de segmentos perdidos es interpretada erróneamente por un emisor TCP como una indicación de congestión y por ello, es tratada mediante la reducción de la ventana de congestión. Tal acción, claramente, no alivia la condición de pérdida aleatoria y simplemente da como resultado la degradación del rendimiento. De esta forma, los algoritmos que suponen que la pérdida de paquetes, causada por errores de bits, es muy pequeña y que la

misma es señal de congestión en algún punto de la red, fuerzan a que TCP disminuya la tasa de envío de datos, incluso cuando no haya congestión presente. De esta forma, además de la baja utilización de la capacidad de la red, se pueden necesitar muchos ciclos RTT de tiempo para que el valor de cwnd pueda crecer a un valor que implique un uso eficiente de la red.

La incapacidad de TCP para distinguir entre reordenamiento y la pérdida de paquetes hace que el protocolo tenga un rendimiento bajo en las rutas que entregan los paquetes fuera de orden. Las pérdidas detectadas pueden hacer que TCP reduzca su tasa de envío. Confundir reordenamiento con pérdida de paquetes es de incumbencia del mecanismo Fast Retransmit, que concluye arbitrariamente que un paquete debe haberse perdido si aún no ha arribado al extremo del receptor después de que tres paquetes enviados más tarde hayan llegado. En una ruta cuya tasa de reordenamiento de paquetes comienza a ser importante, esta elección de tres ACK duplicados puede ser demasiado agresiva para concluir una pérdida debido a lo cual, esperar más tiempo antes de concluir la pérdida podría revelar que el paquete no se perdió en absoluto, sino que sólo se retrasó en el camino. Ante el fenómeno del reordenamiento, el remitente responde con una retransmisión rápida, aunque no se haya producido ninguna pérdida real. Estas repetidas retransmisiones rápidas mantienen un tamaño pequeño de la ventana del emisor y degradan severamente el rendimiento.

En la siguiente figura (Figura 13) [20] se observan algunas variantes del protocolo TCP con distintas implementaciones, que intentan mitigar el efecto de varios problemas de control de la congestión en tres categorías principales:

- REACTIVO: se basa en la pérdida de paquetes como indicador de congestión.
- PROACTIVO: se basa en la estimación del retardo extremo a extremo como indicador del estado de congestión.
- HIBRIDO: se basa en la perdida de paquetes con estimación del ancho de banda.

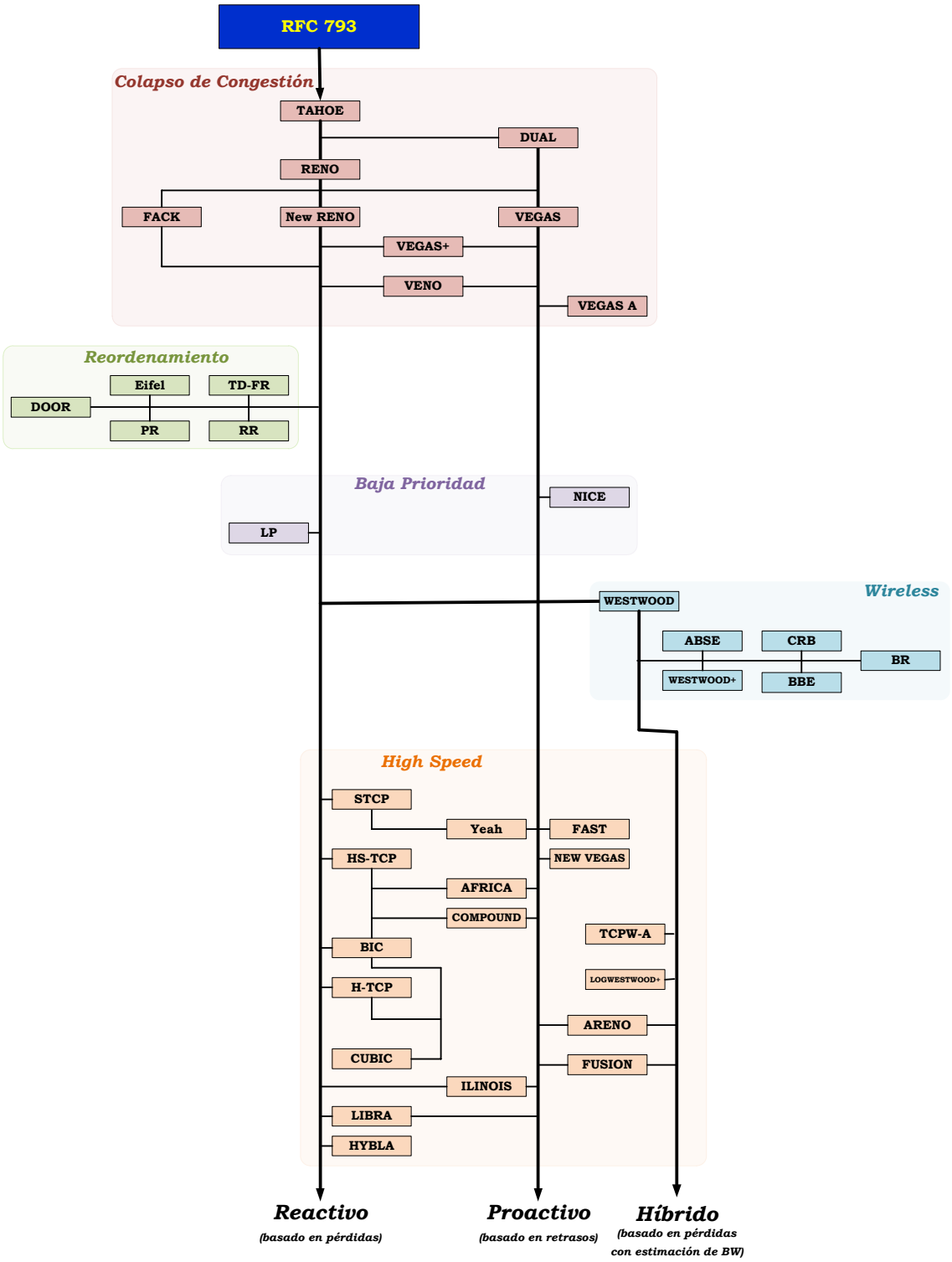

Figura 13: Gráfico evolutivo de las modificaciones del control de congestión TCP

[FUENTE: An initiative for a classified bibliography on TCP / IP congestion control, S. Lar and X. Liao [20]]

En la actualidad [21], la investigación sobre el TCP en sí está aún activa, con nuevas variantes que apuntan a la baja latencia y a la completa utilización de ancho de banda, como el T*CP Bottleneck Bandwidth and Round-Trip (BBR)* o

TCP Low Latency (LoLa). Sin embargo, existen otras líneas de investigación que han tomado fuerza, como es el uso de protocolos de transporte implementados en el espacio de usuario que utilizan UDP, entre los que se destaca *Quick UDP Internet Connections (QUIC)*. Otra de las tendencias está relacionada con la adopción de soluciones de múltiples rutas (Multipath) en la capa de transporte, principalmente con Multipath TCP (MPTCP), pero también con extensiones de múltiples rutas para SCTP, QUIC.

2.4. Variantes de los algoritmos de control de congestión

A continuación [22], se trata de recopilar, clasificar y analizar algunas de las variantes de los algoritmos de control de congestión que optimizan varios parámetros de transferencia de datos TCP. Logran esto, sin depender de ninguna notificación explícita de la red, es decir, preservan el principio host-to-host de TCP y la red percibida como una caja negra [23]. Tratan de resolver el problema en los extremos de la conexión (emisor y receptor) sin esperar ningún tipo de ayuda de la red. La principal ventaja de este enfoque es que mantiene el diseño de la capa de transporte extremo a extremo y no agrega overhead en la red [24].

Las variantes seleccionadas fueron desarrolladas para mejorar el rendimiento en escenarios determinados, o bien, fueron presentadas para resolver alguno de los problemas observados en los distintos entornos [25]. De esta manera, se han agrupado de acuerdo al escenario o característica que intentan dar cuenta. Se exponen distintas situaciones pero sin pormenorizar en el problema que plantean la de las redes inalámbricas que será desarrollado en el capítulo siguiente (*3.6* Variantes TCP End-To-End para Redes Inalámbricas).

La idea es probar variantes que fueron desarrolladas para un escenario determinado en un escenario con acceso WLAN y comprobar su comportamiento, tratando de determinar si mejora en un escenario y empeora en Inalámbrico [26].

2.4.1. Colapso por congestión

La primera modificación de TCP fue la incorporación del control de congestión a raíz del colapso por congestión. A partir de esta variante se desarrollaron las subsiguientes.

Cuando hay muchos usuarios y demandas de recursos en la red, la tasa de envío de todos los emisores TCP que comparten la misma red, puede crecer fácilmente y superar su capacidad. Es bien sabido que si la carga ofrecida en un sistema de distribución no controlada (por ejemplo, el tráfico en una autopista) excede la capacidad total del sistema, la carga efectiva transportada pasará a cero (colapsos) a medida que aumenta la carga. En TCP el origen de este efecto fue conocido como colapso por congestión [27].

En la siguiente figura (Figura 14) se observa la línea de tiempo en el desarrollo de las versiones de TCP antes y después del colapso por congestión.

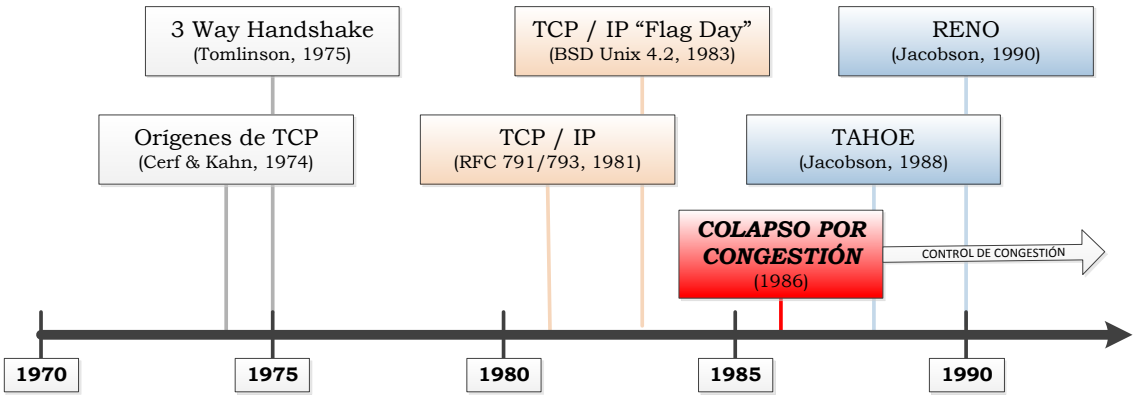

Figura 14: Cronología de TCP y el colapso por congestión

TCP Tahoe

La solución [28] se basa en la especificación original de TCP, en RFC793 [1], e incluye los algoritmos Slow Start y Congestion Avoidance. Estos proporcionan dos mecanismos distribuidos de Host-To-Host, ligeramente diferentes, que le permiten a un emisor TCP, detectar los recursos de red disponibles y ajustar la velocidad de transmisión del flujo TCP a los límites detectados. Suponiendo que la probabilidad de corrupción de paquetes aleatoria durante la transmisión sea muy baja, el host emisor puede deducir todas las pérdidas de paquetes detectadas como indicadores de congestión. Además, la recepción de

cualquier paquete ACK es una indicación de que la red puede aceptar y entregar al menos un paquete nuevo (es decir, el paquete del cual se confirmó la recepción con el ACK ha dejado la red y uno nuevo puede entrar). Por lo tanto, el remitente, razonablemente seguro de que no causará congestión, puede enviar al menos la cantidad de datos que acaban de ser reconocidos. Este equilibrio entre paquetes entrada-salida se denomina *principio de conservación de paquetes* y es un elemento central, tanto de Slow Start como de Congestion Avoidance.

Cuando se establece una nueva conexión, TCP desconoce la capacidad disponible de la red y, por lo tanto, el tamaño óptimo de cwnd. Para obtener su valor ideal, la estrategia es comenzar a enviar datos a tasas de cada vez más altas hasta que se produzcan pérdidas de paquetes. Después de establecer la conexión, el TCP emisor ajusta el valor de cwnd = 1 SMSS. La primera fase está dominada por el algoritmo *Slow Start*, que incrementa el tamaño de la ventana de congestión, *cwnd = min (N, SMSS)* para cada ACK recibido, donde N es el número de bytes aún no reconocidos. Este algoritmo, especificado en el RFC5681, al recibir un paquete de reconocimiento (ACK), puede enviar aproximadamente el doble de la cantidad de datos que fueron reconocidos por ese ACK (política de aumento multiplicativo). No obstante, si se detecta una pérdida de paquetes, la ventana de congestión se restablece al valor inicial (por ejemplo, uno) para garantizar la liberación de recursos de red.

El crecimiento exponencial de la ventana de congestión que provoca el algoritmo Slow Start puede hacer crecer la ventana de congestión a valores donde se inyectan grandes ráfagas de datos en la red. Si la capacidad de la red disponible fuese utilizada en su totalidad con grandes ráfagas de tráfico, las conexiones que comparten la misma ruta sufrirían pérdidas significativas de segmentos, lo que conduciría a la inestabilidad de la red.

Para abordar el problema de tratar de encontrar capacidad adicional disponible de la red, pero sin hacerlo de modo agresivo, TCP implementa el algoritmo Congestion Avoidance. Cuando cwnd alcanza el valor del umbral, TCP ejecuta este algoritmo con el fin de obtener esa capacidad adicional, que aumenta el tamaño de la ventana en aproximadamente un segmento cada vez que se

recibe un ACK. El tamaño de la ventana de congestión cwnd se actualiza de la siguiente manera (Ecuación 17) para cada ACK no duplicado recibido:

$$\text{cwnd} = \text{cwnd} + \text{SMSS} * \frac{\text{SMSS}}{\text{cwnd}} \qquad \textbf{Ecuación 17}$$

Debido a que el valor de cwnd crece ligeramente con cada llegada de un ACK, y este valor está en el denominador de la expresión, la tasa de crecimiento global de cwnd es aproximadamente lineal.

La implementación de TCP Tahoe incluye los algoritmos Slow Start y Congestion Avoidance como fases operativas distintas. Esto combina el descubrimiento rápido de recursos de red y la eficiencia a largo plazo.

Para los fines de conmutación de fase, se introduce un parámetro denominado *ssthresh*, es decir, *umbral de Slow Start (Slow Start threshold)*. Este umbral es el límite en el tamaño de cwnd que determina qué algoritmo está en funcionamiento: Cuando cwnd<ssthresh, TCP utiliza Slow Start, y cuando cwnd>ssthresh, TCP utiliza Congestion Avoidance.

El valor de ssthresh no es fijo, sino que varía en el tiempo y determina el tamaño máximo de la ventana de congestión en la fase de Slow Start. Además, la detección de cualquier pérdida de paquetes ajusta el umbral a la mitad de la ventana de congestión actual.

El valor inicial de ssthresh se puede establecer en un valor alto para explorar la capacidad disponible de la red y, y en cuanto se produce una retransmisión causada por haberse extinguido un temporizador de retransmisión, ssthresh se actualiza (Ecuación 18) de la siguiente manera:

$$\text{ssthresh} = \max\left(\frac{\text{ValorMasAltoCWND}}{2}, 2 * \text{SMSS}\right) \qquad \textbf{Ecuación 18}$$

Si se produce una retransmisión, TCP asume que el tamaño de la ventana es demasiado grande para lo que puede manejar la red. De esta manera, se debe reducir el valor del tamaño de la ventana. La Ecuación 18 define que el umbral ssthresh sea el mayor valor entre la mitad del valor más alto que alcanzó cwnd y dos veces SMSS, es decir, que sea la mitad de la ventana de congestión actual, pero nunca inferior al doble del SMSS.

Esto se conoce como el ciclo de fase Slow Start-Congestion Avoidance que se puede observar en la siguiente figura (Figura 15).

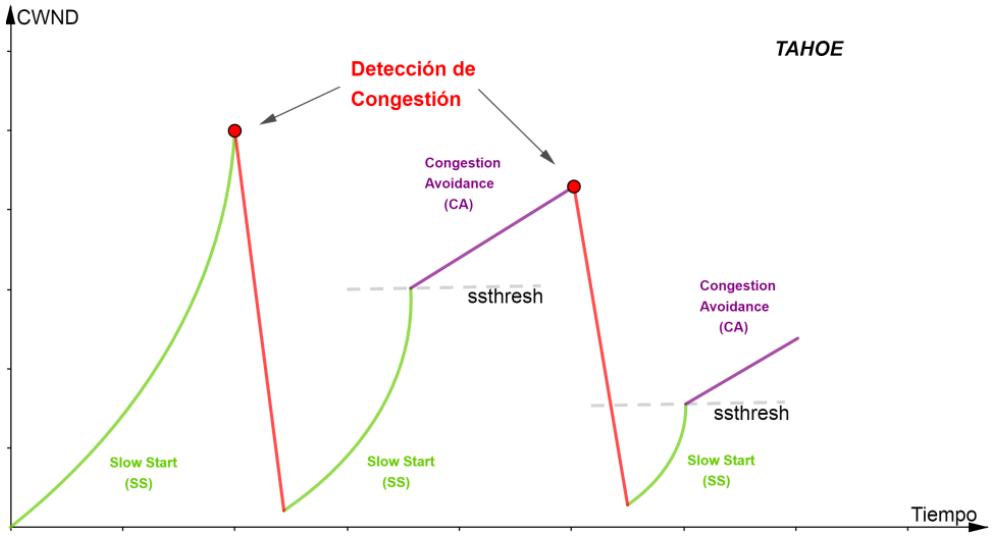

Figura 15: Evolución de la ventana de congestión TCP Tahoe

TCP Tahoe incluyó la estrategia de Fast Retransmit, lo que implica reenviar el segmento luego de recibir tres ACK duplicados. Sin embargo, como presupone que los paquetes se perdieron, establece el umbral ssthresh=cwnd/2 y la ventana de congestión cwnd=1, reiniciando la fase Slow Start.

```
//** _____________________________________________________________________ **//
// **TCP T A H O E** //
/** Valores iniciales **/
cwnd = 1;
ssthresh = 65536;
FOR (Recepción de ACK)
IF (cwnd < ssthresh)
               /*Algortimo Slow Start*/
               cwnd = cwnd + 1;
ELSE
               /*Algoritmo Congestion Avoidance*/
               cwnd = cwnd + (1=cwnd);
FOR (caducidad de un temporizador OR 3 DupACK)
/* reducción multiplicativa */
ssthresh = cwnd /2;
cwnd = 1;
       Retransmitir el segmento;
       Invocar al algoritmo Slow Start;
//** _____________________________________________________________________ **//
```

TCP Reno

TCP Tahoe, inicia las conexiones en Slow Start y si ocurre la retransmisión de un paquete, sin importar en qué fase se encuentre, siempre termina invocando al algoritmo de Slow Start.

Tahoe se implementó mediante la reducción del valor de cwnd a su valor inicial (1 SMSS) sobre cualquier pérdida, lo que fuerza a reiniciar el algoritmo Slow Start hasta que el valor cwnd sea mayor que el de ssthresh. La reducción de la ventana de congestión a un segmento, como reacción a una pérdida puede, en algunos casos, conducir a una degradación significativa del rendimiento. Además, requiere de varios ciclos de RTT para recuperar la tasa de envío anterior a la retransmisión del paquete perdido.

Para resolver este problema se revisó la composición original de Slow Start y Congestion Avoidance [29], [30] introduciendo la idea de diferenciar entre eventos de mayor y menor congestión. La detección de pérdida a través de la caducidad de un temporizador de retransmisión, indica que, durante cierto intervalo de tiempo, se ha impedido la entrega de cualquier paquete de datos en la red. Por lo tanto, el emisor debe aplicar la política conservadora de restablecer la ventana de congestión a un valor mínimo.

Se puede inferir un estado completamente diferente ante una pérdida detectada por ACK duplicados. La llegada de estos ACK duplicados indica que los paquetes subsiguientes no han llegado, pero, sin embargo, la recepción de los mismos indica la entrega satisfactoria de un paquete de datos. El remitente, además de detectar el paquete perdido, también está observando la capacidad de la red para entregar algunos datos. Por lo tanto, el estado de la red puede considerarse ligeramente congestionado, y la reacción al evento de pérdida puede ser más optimista. En TCP Reno, la reacción optimista es utilizar el algoritmo Fast Recovery.

Fast Recovery reduce a la mitad el flujo hacia la red, al mismo tiempo que disminuye a la mitad la ventana de congestión y sondea la red aumentando el tamaño de cwnd, hasta que se recupera el error. El emisor permanece en Fast Recovery hasta que recibe un reconocimiento no duplicado. El algoritmo Fast Recovery permite el crecimiento temporal de la ventana de congestión de a 1

SMSS por cada ACK recibido durante la recuperación y, de esta manera, permite que se envíe un nuevo paquete, hasta que se reciba un ACK no duplicado. De esta forma, después de la reducción del tamaño de la ventana de congestión a la mitad, el algoritmo no sólo retransmite el paquete de datos no reconocido más antiguo (Fast Retransmit), sino que también hace crecer la ventana de congestión en el número de paquetes duplicados.

En la etapa final, cuando se recibe un ACK no duplicado, se reanuda la fase de Congestion Avoidance con el valor de cwnd en la mitad de la ventana de congestión original. En este punto, esta reducción de la ventana de congestión al valor justo después de entrar en la recuperación es una manera simple y confiable para asegurar el estado de salida de Fast Recovery.

En la siguiente figura (Figura 16) se observa la variación del tamaño de la ventana de congestión para TCP Reno, superpuesta a la del TCP Tahoe.

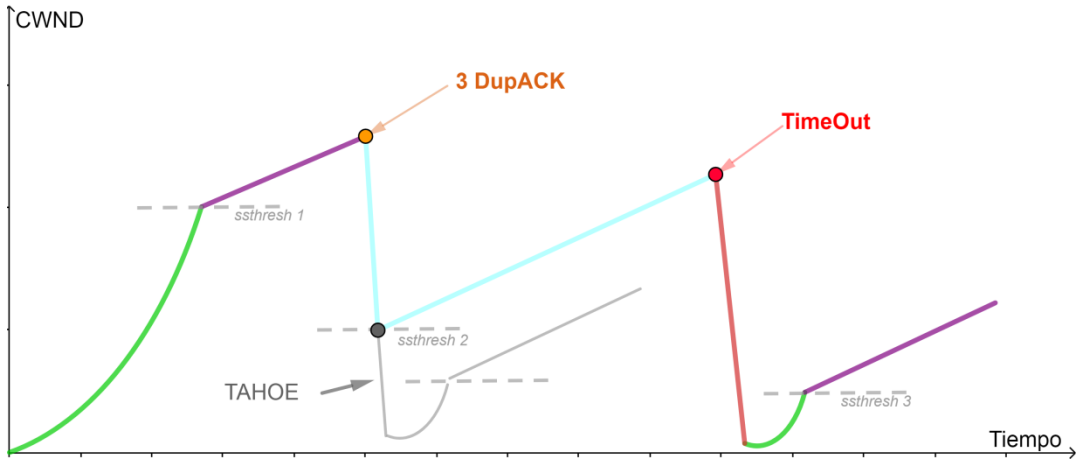

Figura 16: Evolución de la ventana de congestión en TCP Reno comparada con TCP Tahoe

Como se observa en la Figura 16, cuando la detección de la pérdida es por la recepción de tres ACK duplicados, Tahoe reduce drásticamente la tasa de envío, mientras que la reducción de la tasa de Reno es significativamente menor. En el caso que la pérdida se detecte por un timeout, la respuesta es idéntica a la respuesta de TCP Tahoe. De esta manera, como se observa en la Figura 15, TCP Tahoe trata de la misma manera la recepción de tres ACK duplicados y la expiración de un temporizador de retransmisión (timeout).

Si se compara con la dinámica de TCP Tahoe (Figura 15), la efectividad global en el estado estacionario se mejora considerablemente reemplazando las fases

de Slow Start después de cada detección de pérdida por Fast Retransmit. De hecho, la recuperación de una sola pérdida ocurriría dentro de un RTT. Sin embargo, la eficiencia se mejora no sólo reduciendo el período de recuperación, sino también permitiendo transferencias de datos durante esta etapa.

Debido a su simplicidad y características de rendimiento, Reno es generalmente el estándar de control de congestión para TCP. Los algoritmos de Fast Retransmit y Fast Recovery de TCP Reno permiten que la conexión pueda recuperarse rápidamente de pérdidas aisladas de paquetes. Pero cuando se pierden varios paquetes de una misma ventana de datos, TCP Reno puede degradar severamente su rendimiento. Hay una amplia gama de entornos de red en los que Reno no tiene un buen rendimiento, en donde se presentan pérdidas consecutivas de paquetes, pérdidas aleatorias de paquetes y/o reordenamiento de paquetes. También tiene problemas con la equidad si los flujos tienen valores de RTT diferentes, y no utiliza eficientemente los canales de red de alto BDP (producto de ancho de banda y latencia).

Los algoritmos de Fast Retransmit y Fast Recovery de TCP Reno permiten que la conexión pueda recuperarse rápidamente de pérdidas aisladas de paquetes. Pero cuando se pierden varios paquetes de una misma ventana de datos, TCP Reno puede degradar severamente su rendimiento.

```
//** _____________________________________________________________________ **//
// **TCP R E N O** //
/** Valores iniciales **/
0 < cwnd ≤ min(4*SMSS, max(2*SMSS, 4380 bytes));
ssthresh = max(cwnd/2, 2*SMSS);
FOR (Recepción de ACK)
IF (cwnd < ssthresh)
               /*Algortimo Slow Start*/
               cwnd = cwnd + 1;
ELSE
               /*Algoritmo Congestion Avoidance*/
               cwnd = cwnd + (1=cwnd);
FOR (3 DupACK)
/* Fast Retransmit – Fast Recovery */
ssthresh = cwnd / 2;
cwnd = ssthresh + 3;
For (cada ACK duplicado recibido)
cwnd++;
enviar un segmento si cwnd lo permite;
       For (cada ACK recibido)
```

```
                If (ACK recibido parcial?)
                        Quedarse en Fast Recovery;
                        Retransmitir el próximo segmento perdido (1 por RTT);
                If (ACK recibido completo?)
                        cwnd = ssthresh;
                        Regresar a Congestion Avoidance;
FOR (caducidad de un temporizador)
        ssthresh = cwnd / 2;
cwnd = 1;
Invocar al Algoritmo Slow Start;
//** _____________________________________________________________________________  **//
```

TCP New Reno

Una de las vulnerabilidades del algoritmo de Fast Recovery de TCP Reno se manifiesta cuando se producen múltiples pérdidas de paquetes como parte de un único evento de congestión, lo que reduce significativamente el rendimiento de Reno en entornos de alta carga. La reacción optimista deseada de Fast Recovery, es decir, la reducción de la mitad de la ventana de congestión, se transforma en una disminución exponencial. Es decir, la primera pérdida provoca la entrada en la fase de Fast Recovery y la reducción a la mitad de la ventana de congestión. La recepción de cualquier ACK no duplicado, antes de la recuperación de todos los paquetes perdidos, terminaría la recuperación rápida. Sin embargo, las detecciones de pérdidas posteriores hacen que la ventana de congestión disminuya aún más, utilizando los mismos mecanismos de entrada y salida del estado de recuperación.

En un sentido, esta reacción exponencial a múltiples pérdidas se espera del algoritmo de control de congestión, cuyo propósito es reducir el consumo de recursos de la red en situaciones complejas de congestión. Pero esta expectativa descansa en el supuesto de que los estados de congestión, como se deduce de cada pérdida detectada, son independientes. Sin embargo, todas las pérdidas de paquetes de una ventana tienen una alta probabilidad de ser causadas por un evento de congestión único. Por lo tanto, la segunda, la tercera y subsiguientes pérdidas deben tratarse sólo como solicitudes de retransmisión de datos y no como indicadores de congestión.

La reducción de la ventana de congestión no garantiza la liberación instantánea de los recursos de red. Todos los paquetes enviados antes de la reducción de la ventana siguen en tránsito. Por lo tanto, no se debería aplicar ninguna

política adicional de reducción antes que el nuevo tamaño sea efectivo. Esto puede definirse como una reducción de la ventana de congestión, solo una vez por retardo de propagación.

Para resolver este problema se ha desarrollado una modificación llamada New Reno [31], definida en el RFC3782 [32] que introduce un simple cambio al algoritmo de Fast Recovery. Resuelve la ambigüedad de los eventos de congestión al restringir la salida de la fase Fast Recovery hasta que se reconozcan todos los paquetes de datos de la ventana inicial de congestión. Para esto, el algoritmo New Reno añade una variable de estado especial para recordar el número de secuencia más alto del último paquete de datos enviado antes de entrar en el estado de Fast Recovery. Este valor ayuda a distinguir entre ACK de datos parciales y nuevos. Esto permite que un TCP continúe enviando un segmento por cada ACK que recibe mientras se recupera y reduce la ocurrencia de tiempos de espera de retransmisión, especialmente cuando se eliminan múltiples paquetes en una sola ventana de datos. La recepción de un nuevo ACK de datos significa que todos los paquetes enviados antes de la detección de errores fueron entregados con éxito y cualquier nueva pérdida reflejaría un nuevo evento de congestión. Un ACK parcial confirma la recuperación de sólo el primer error e indica más pérdidas en la ventana original de paquetes.

Sin embargo, en algunos casos en particular, cuando caduca un temporizador de retransmisión durante la fase de Fast Retransmit, se puede producir una reducción innecesaria de la ventana de congestión. La solución es recordar el número de secuencia más alto transmitido hasta ahora después de cada timeout, e ignorar todos los ACK duplicados que reconocen números de secuencia inferiores. Esta solución resuelve de manera optimista la ambigüedad de los ACK duplicados que pueden indicar paquetes perdidos o duplicados.

New Reno modifica sólo el algoritmo de Fast Recovery mejorando su respuesta en caso de múltiples pérdidas. El problema con New Reno es que la restricción de retransmitir a lo sumo un paquete por RTT, produce un retraso sustancial en la retransmisión de los paquetes descartados posteriores en la ventana. Por lo tanto, el ancho de banda disponible no se utiliza de manera efectiva.

Opción TCP SACK y TCP FACK

TCP SACK (Selective Acknowledgement) no es un algoritmo de control de congestión como tal, sino más bien es una opción extendida de TCP New Reno. Partiendo de la mejora que supone el uso del Selective Acknowledgement (SACK), surgió TCP FACK (Forward Acknowledgement) que funciona alrededor de los problemas de detección de múltiples paquetes perdidos y retransmisión por RTT.

La especificación TCP define que los mensajes de realimentación del receptor al emisor deben ser en forma de ACK acumulativos, es decir, reconocimientos del último paquete de datos entregado en orden. Así, la duración de recuperación es directamente proporcional al número de paquetes perdidos y el valor de RTT.

Si un receptor puede proporcionar información sobre varias pérdidas de paquetes dentro de un único mensaje, el emisor podría implementar un algoritmo simple para resolver el problema de recuperación. Además, el problema de Reno puede resolverse restringiendo la reducción de la ventana de congestión a no más de una vez por periodo RTT, en lugar de implementar el algoritmo New Reno. Como solución, en el RFC2018 [33], se propuso extender el protocolo TCP normalizando la opción de reconocimiento selectivo (SACK). Esta opción proporciona la capacidad para que el receptor informe bloques de paquetes de datos entregados con éxito. De esta forma, permite que el receptor reconozca explícitamente los datos que llegaron fuera de orden al emisor.

Utilizando esta información, los emisores TCP pueden calcular fácilmente los bloques de paquetes perdidos (espacios en números de secuencia) y retransmitirlos rápidamente. Lamentablemente, el mecanismo SACK tiene serias limitaciones en su forma actual. La especificación TCP restringe la longitud del campo de opción a 40 bytes. Un cálculo simple revela que la opción SACK puede contener, como máximo, cuatro bloques de paquetes de datos recibidos en orden. Aunque es poco probable que se supere en las redes cableadas (ya que durante los eventos de congestión normalmente se sueltan paquetes consecutivos), las pérdidas aleatorias en redes inalámbricas pueden mostrar patrones que se aproximan al peor de los casos. Esta observación

muestra que SACK no es una solución universal para el problema de pérdida múltiple.

Poco tiempo después de haberse desarrollado TCP SACK, nació TCP FACK como algoritmo de control de congestión, haciendo un punto de unión entre Reno y SACK. FACK [34] hace referencia a Forward Acknowledgement. Por una parte, utiliza ACK selectivos, indicando especialmente los paquetes perdidos. La diferencia es que Reno y New Reno utilizan la información que proporcionan los segmentos SACK para ver cuántos paquetes hay que reenviar y cuáles han llegado correctamente.

TCP FACK se concibió como un esfuerzo por minimizar la pausa de transmisión después de pérdidas de paquetes. Utiliza la opción SACK para estimar mejor la cantidad de datos en tránsito. Hace una distinción entre el intervalo de ajuste, que es el período cuando se modifica el tamaño de cwnd, y el intervalo de recuperación, que es cuando se retransmiten los segmentos. De esta forma intenta desacoplar los algoritmos de control de congestión de los algoritmos de los de recuperación de datos.

La parte de control de flujo del algoritmo FACK utiliza SACK para indicar pérdidas. Proporciona, también, un medio para la retransmisión oportuna de los paquetes de datos perdidos. El emisor FACK debe retener información sobre los datos retransmitidos y debe incluir, al menos, la hora de la última retransmisión.

La parte de control de velocidad utiliza la información adicional proporcionada por la opción SACK para mantener una medida explícita de la cantidad total de datos pendientes en la red. En contraste, los algoritmos de Fast Recovery de Reno y New Reno intentan estimarlo asumiendo que cada DupACK recibido representa un segmento que ha abandonado la red. El objetivo de FACK es realizar un control preciso de la congestión durante la recuperación. Al controlar con precisión los datos pendientes en la red, FACK puede mejorar el rendimiento de la conexión durante la fase de recuperación de datos. FACK mantiene tres variables de estado especiales para lograr la estimación: El número de secuencia más alto de todos los paquetes de datos enviados, el número de secuencia más alto de todos los paquetes de datos reconocidos y el número de paquetes retransmitidos.

La relación simple entre estas tres variables proporciona una estimación fiable de paquetes de datos pendientes en la red. Esta expresión captura el tamaño de paquetes en vuelo, incluidas las retransmisiones, y garantiza que si se inyecta otro paquete la red será capaz de manejarlo. Esta estimación puede ser utilizada para definir si los datos pueden ser enviados cuando el número calculado de paquetes de datos pendientes está por debajo de un límite permitido por el tamaño de la ventana de congestión (cwnd).

TCP Vegas

Las soluciones al colapso de congestión propuestas por TCP Tahoe, Reno, New Reno y sus modificaciones, tienen el inconveniente de forzar a la red a oscilar entre fases periódicas, que induce cambios significativos en la tasa de envío, el tiempo de ida y vuelta (RTT) y la utilización de los buffers de la red. Es decir, la red está oscilando en los alrededores de un estado estacionario, que encuentra el límite de los recursos de la red al forzar la pérdida de paquetes. Estos enfoques de los controles de congestión comparten el mismo método reactivo de adaptación de la tasa. Cada uno de ellos detecta que la red está congestionada sólo si se pierden algunos paquetes. De esta manera, provocan pérdidas de paquetes porque sus algoritmos aumentan las tasas de transmisión hasta el punto de congestión de la red, para descubrir los recursos disponibles.

Otro enfoque diferente es utilizar una estimación del retardo de cola para cuantificar el nivel de congestión antes de que ocurra un evento de congestión real. Este método proactivo intenta mitigar los patrones oscilatorios en la dinámica de la red y generar reacciones más suaves a los acontecimientos detectados.

En este sentido, se propuso el algoritmo de Vegas [35] para reemplazar el algoritmo reactivo de Congestion Avoidance. Vegas utiliza la estimación de la trasferencia de la red y la compara con la trasferencia que realmente puede realizar, en contraste con el método reactivo que incrementa la tasa de envío hasta que se produzca la pérdida de segmentos para encontrar el límite de la capacidad de la red.

Esta variante, considera el valor de RTT mínimo observado por el emisor como una buena medida de que la ruta está en un estado sin congestión. Supone que un aumento del RTT sólo puede ocurrir debido al aumento de la utilización del buffer (mayor longitud en las colas intermedias). De esta manera, la diferencia entre el valor RTT medido y el valor RTT mínimo puede ser vista como un indicador del nivel de congestión en la trayectoria.

TCP Vegas utiliza la diferencia entre el throughput Esperado y el throughput Medido como la medida de la congestión. TCP Vegas registra el menor tiempo de ida y vuelta como RTTmin y se calcula el ancho de banda disponible como (Ecuación 19):

$$\text{ThroughputEsperado} = \frac{\text{TamañodelaVentana}}{\text{RTTmin}} \qquad \textbf{Ecuación 19}$$

Donde, *Tamaño de la ventana* es el tamaño actual de la ventana de congestión.

Durante la transmisión de los datos, se registra el tiempo de ida y vuelta (RTT) de los paquetes y calcula el throughput medido como (Ecuación 20):

$$\text{ThroughputMedido} = \frac{\text{TamañodelaVentana}}{\text{RTTmedido}} \qquad \textbf{Ecuación 20}$$

Donde, *Tamaño de la ventana* es el tamaño actual de la ventana de congestión.

Por lo tanto, la diferencia entre el Throughput Esperado y el Throughput Medido se usará para ajustar el tamaño de la ventana (Ecuación 21):

$$\Delta = \text{ThroughputEsperado} - \text{ThroughputMedido} \qquad \textbf{Ecuación 21}$$

Vegas intenta cuantificar el número de paquetes en cola que se encuentra en el cuello de botella, como una función del Throughput Esperado y el Medido como se puede observar en la siguiente figura (Figura 17).

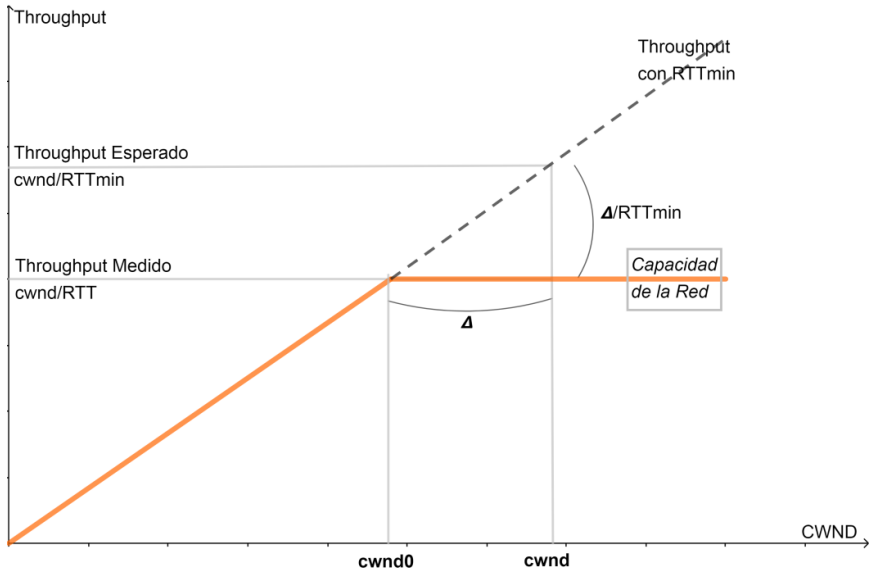

Figura 17: Tamaño del buffer utilizado, como función del throughput esperado y el medido

El Throughput Esperado (línea punteada en la Figura 17) es una tasa teórica de un flujo TCP en un estado de red sin congestión. Si no hay congestión, el RTT medido se mantiene constante y con el mismo valor de RTTmin, por lo que a medida que crece la ventana de congestión, crece el throughput. El Throughput Medido (línea sólida en la Figura 17) se puede expresar como la relación entre la ventana de congestión actual y el valor RTT actual. Sin embargo, debido a la capacidad finita de la trayectoria, siempre se puede encontrar un punto cwnd0 en la Figura 17, donde el throughput medido es numéricamente igual al throughput esperado y, por lo tanto, fallaran todos los intentos de enviar a un ritmo más rápido. El número de paquetes en cola durante el último RTT es la diferencia Δ (Ecuación 22) entre la ventana de congestión actual y el punto de inflexión de la línea sólida de la Figura 17, es decir,

$$\Delta = \mathrm{cwnd} - \mathrm{cwnd0} \qquad \textbf{Ecuación 22}$$

De esta manera, el exceso de paquetes de datos es la única causa del aumento del valor de RTT. Así, Δ puede expresarse como una función del tamaño de la ventana de congestión, RTT y RTT_{min}:

$$\Delta = \mathrm{cwndx}\frac{\mathrm{RTT} - \mathrm{RTTmin}}{\mathrm{RTT}} \qquad \textbf{Ecuación 23}$$

Vegas incorpora esta medida Δ en la fase de Congestion Avoidance para controlar el tamaño de la ventana de congestión. Se definen como los umbrales los valores α y β (0 ≤ α < β). Si Δ<α, el tamaño de la ventana se incrementa durante el próximo RTT; Si Δ > β, entonces el tamaño de la ventana se reduce durante el próximo RTT. De lo contrario, si está entre α y β, se considera que el sistema está en estado estacionario y no se aplican modificaciones a la ventana de congestión.

En la siguiente figura (Figura 18) [36], se observa el comportamiento de la ventana de congestión de Vegas.

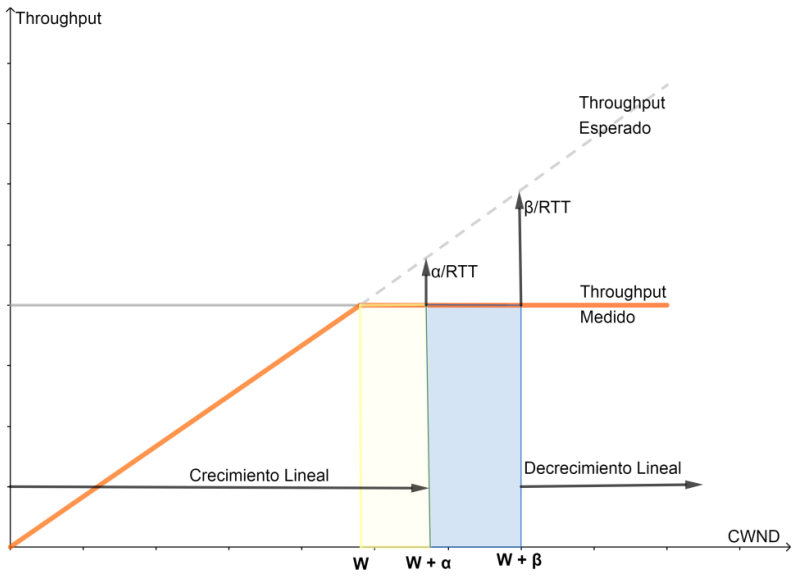

Figura 18: Control del tamaño ventana de Congestión de TCP Vegas

Como se observa en la Figura 18, cuando el tamaño de la ventana de congestión (cwnd) es mayor que W, la longitud de las colas comienza a crecer. TCP Vegas aumentará el tamaño de la ventana en uno durante el siguiente RTT si $cwnd<w+\alpha$, o bien, reducirá el tamaño en uno si $cwnd>w+\beta$. De lo contrario, si $w+\alpha<cwnd<w+\beta$ deja el tamaño de la ventana sin cambios [37].

El objetivo de TCP Vegas es mantener una cierta cantidad de paquetes o bytes en las colas de la red.

Este mecanismo utilizado en TCP Vegas para estimar el ancho de banda disponible, no causa deliberadamente ninguna pérdida de paquetes. Por lo tanto, se elimina el comportamiento oscilatorio y se logra un mejor rendimiento. El mecanismo de retransmisión utilizado por TCP Vegas es más eficiente en

comparación con TCP Reno, ya que retransmite el paquete correspondiente tan pronto como recibe un único ACK duplicado. TCP Vegas, en comparación con TCP Reno, es más preciso y menos agresivo, por lo que no reduce su cwnd innecesariamente.

TCP Vegas tiene la propiedad de estabilizar la velocidad en estado estacionario, lo que puede mejorar significativamente el rendimiento general de un flujo TCP. Desafortunadamente, a pesar de esta y otras ventajas, presenta una incapacidad para obtener una parte justa del ancho de banda, al competir con flujos más agresivos, como TCP Reno [38]. También subestima los recursos de red disponibles en algunos entornos, por ejemplo, en el caso de caminos múltiples, y los nuevos flujos obtienen una mayor participación debido a estimaciones inexactas del valor del RTT mínimo. En determinadas circunstancias, reduce la ventana de congestión innecesariamente por ejemplo cuando hay congestión significativa en el sentido inverso del flujo TCP y los paquetes (ACK) que regresan se retrasan.

El mecanismo proactivo de Congestion Avoidance de Vegas no puede competir efectivamente con el mecanismo reactivo que induce la utilización del buffer de la red y su desbordamiento [39]. Las reacciones opuestas de los dos algoritmos de control de congestión diferentes mantienen el nivel de almacenamiento intermedio fijo en la red, lo que lleva a que el algoritmo proactivo sea completamente rezagado en el aprovechamiento justo de los recursos de la red. TCP Vegas+ se propuso como una forma de proporcionar una forma de implementación incremental de Vegas. Para este propósito, Vegas+ [40] toma prestados tanto los enfoques reactivos como los proactivos en la fase de Congestion Avoidance. Vegas+ inicialmente asume un ambiente de red amigable con Vegas y emplea la estimación de la longitud de las colas para controlar la ventana de congestión. En el momento en que una exploración interna detecta un entorno hostil a Vegas, la fase de Congestion Avoidance se reduce al algoritmo de Reno. La heurística de detección de Vegas+ se basa en una estimación de tendencia del RTT, es decir, si la ventana de congestión es estable, el RTT también debería serlo.

Además de la incapacidad para competir eficazmente con los flujos del tipo de Reno, TCP Vegas tiene una serie de otros problemas: bajo ciertas

circunstancias, Vegas puede reducir indebidamente la tasa de flujo a casi cero. Esto sucede porque la suposición de que el RTT cambiará sólo debido al almacenamiento en buffer no es del todo verdad. De hecho, si el RTT aumenta debido a un cambio de ruta, el algoritmo tomará una decisión incorrecta, llevando a la reducción de la tasa de envío. Otra hipótesis incorrecta es que todos los flujos que compiten a lo largo de la misma ruta observarán el mismo RTT mínimo. TCP VegasA (Vegas con adaptación) [41] extiende el control de congestión Vegas original con un mecanismo adaptable. Los coeficientes de umbral α y β del algoritmo, se ajustan dependiendo de la dinámica de estado estacionario. Es decir, si VegasA detecta un aumento en el ancho de banda real mientras el sistema está en un estado estable, asume un cambio de trayecto y desplaza los límites de control hacia arriba (α = α + 1 y β = β + 1). Los límites se desplazan hacia abajo si se detecta alguna anomalía de red; por ejemplo, si la estimación muestra estado libre de congestión y, sin embargo, el Throughput medido ha disminuido [42].

TCP Veno

La idea clave es utilizar la técnica de estimación de longitud de las colas de Vegas para realizar la detección temprana del estado de congestión. A diferencia de Vegas, esta estimación solo se utiliza para ajustar el coeficiente de crecimiento y disminución del tamaño de la ventana de congestión del algoritmo de control de congestión de Reno, y por lo tanto no hereda los problemas de Vegas.

El algoritmo de *Veno (VEgas y reNO)* [43] define dos modificaciones. En primer lugar, limita el aumento de la ventana de congestión durante la fase de Congestion Avoidance: si la estimación de Vegas indica un estado de congestión en la red, el emisor empieza a probar los recursos de la red de forma muy conservadora (por ejemplo, aumentando uno por cada dos RTT). En segundo lugar, la reducción de la ventana de congestión al entrar en Fast Recovery se modifica para reducir a la mitad el valor de cwnd sólo si la estimación también indica congestión. De lo contrario, si sólo se detecta una pérdida, se reducirá al 80% de su tamaño actual.

La eficiencia del algoritmo Veno es ligeramente mejor en comparación con Reno. El flujo de Veno tiende a permanecer más tiempo en el estado de Congestion Avoidance con mayores tamaños de ventana de congestión. Sin embargo, el precio para esto es el retardo adicional para descubrir recursos de red.

2.4.2. Reordenamiento de paquetes

La suposición de que la red, en general, no reordena los paquetes en tránsito [44], ha permitido a los algoritmos crear un mecanismo de detección de pérdida simple sin necesidad de modificar la especificación TCP existente. La ausencia de reordenamiento garantiza que una entrega fuera de orden se produce sólo si se ha perdido algún paquete. Por lo tanto, si el remitente ve varios ACK que llevan los mismos números de secuencia (ACK duplicados), puede estar seguro de que la red no ha podido entregar algunos datos y puede actuar en consecuencia, por ejemplo, retransmitir paquetes perdidos, inferir un estado de congestión y reducir la tasa de envío. Para solucionar el problema de una detección de pérdida falsa, como regla general se establece un valor umbral para el número mínimo de ACK duplicados requeridos para activar una detección de pérdida de paquetes, por ejemplo, tres. Sin embargo, existe un claro conflicto con este enfoque: la detección de pérdidas se retrasará innecesariamente si la red no reordena los paquetes. Al mismo tiempo, el remitente reaccionará exageradamente, por ejemplo, retransmitir datos o reducir la velocidad de transmisión innecesariamente, si la red de hecho reordena los paquetes.

El reordenamiento de paquetes puede observarse por varias causas. Puede ser un comportamiento erróneo de software o hardware, errores en configuraciones, mal funcionamiento, pero también, en algunas redes, como un efecto secundario de un proceso de entrega normal, por ejemplo, si un router trabaja con servicios diferenciados y reprograma internamente paquetes en su cola. Además, si la red proporciona cierto nivel de garantías de entrega, por ejemplo, en redes inalámbricas, la capa subyacente (capa física o de enlace) puede retransmitir parte de los datos sin la petición de TCP.

La presencia de un paquete fuera de orden pone de manifiesto el potencial problema de la pérdida de rendimiento del flujo TCP, cuando el control de congestión utiliza la presencia de ACK duplicados para detectar pérdidas de paquetes. En ausencia de congestión o de pérdidas de paquetes, cada evento de reordenamiento de paquetes dispara al menos un ACK duplicado, lo cual será considerado como una indicación de congestión y el emisor TCP reducirá la tasa de envío de datos, aun cuando la red puede manejar la tasa actual.

TD-FR Time Delayed Fast Recovery propone [45] que, si el receptor no responde inmediatamente a un evento de paquete fuera de orden, y retrasa el envío del ACK duplicado, la mayoría de los eventos de reordenamiento de paquetes se ocultarán al TCP emisor. Sin embargo, la ventaja de este método es a su vez su desventaja. Este retraso adicional agregado para evitar la sobrerreacción del protocolo, aumenta el tiempo necesario para detectar las pérdidas reales de paquetes. Si este retraso es muy grande, la detección rápida de las pérdidas se vuelve más lenta que la detección basada en temporizadores.

El *algoritmo Eifel (Eifel Response Algorithm)* definido en RFC4015 [46], trata de distinguir entre al evento de reordenamiento y las pérdidas de paquetes. No lo hace con la recepción del primer ACK duplicado, sino que posterga la decisión hasta que se recibe el primer ACK no duplicado. Es decir, si el TCP emisor recibe tres ACK duplicados dispara el algoritmo de Fast Recovery. Sólo cuando recibe el primer ACK no duplicado, Eifel revisa este paquete y toma la decisión de seguir en Fast Recovery o abortar y reponer el valor original de la ventana de congestión. Las acciones del algoritmo de Eifel no afectan el normal funcionamiento de los algoritmos de control de congestión cuando no hay reordenamiento de paquetes. Sin embargo, cuando algunos paquetes se reordenan, la tasa de envío original del flujo se restablece rápidamente. Para tomar la decisión correcta, el algoritmo de Eifel debe resolver la ambigüedad cuando se recibe el primer ACK no duplicado si corresponde al paquete retransmitido, o bien, si llegó el paquete desordenado. Para ello, se puede utilizar la opción de timestamp y, que el TCP emisor mantenga la hora de la primera retransmisión para cada paquete reenviado. Si la opción de timestamp del ACK no duplicado recibido tiene una fecha anterior a la nueva variable de

estado, el TCP emisor puede estar seguro que no hubo eventos de pérdidas de paquetes y puede volver a la tasa de transmisión original antes del evento.

TCP DOOR

En los escenarios que se caracterizan por cambios frecuentes de ruta, se degrada fuertemente el rendimiento de TCP. Durante el cambio de ruta, se pueden perder paquetes generando que los algoritmos de control de congestión erróneamente, disminuyan drásticamente la tasa de envío de datos. Si se puede determinar el intervalo de tiempo durante el cual se realiza el cambio de ruta, se podría eliminar la penalidad en el throughput del flujo TCP, deshabilitando temporariamente los mecanismos de control de congestión durante este intervalo. Este es el concepto que subyace en *TCP DOOR (Detection of Out-of-Order and Response)* [47]. El enfoque no depende de información de las capas inferiores, ni de la red, solo participan los hosts extremos de la conexión, en el procedimiento para determinar el estado de la red. Durante el evento de cambio de ruta, es muy probable que el orden de los paquetes cambie. De esta manera, el problema de identificar el cambio de ruta, se reemplaza por el problema de detectar la entrega de paquetes fuera de orden. Para lograrlo, en forma similar al algoritmo de Eifel, cada paquete de datos y ACK deberá tener información adicional. Cuando se detecta un reordenamiento de paquetes, se deben deshabilitar los mecanismos de control de congestión para disminuir los efectos de la transición o cambio de ruta. Si el control de congestión redujo recientemente la tasa de envío del emisor debido a la detección de una pérdida de paquetes, deberá retrotraer los valores a los del estado inmediatamente anterior del tamaño de la ventana y los temporizadores de retransmisión. A esta acción se la denomina *Instant Recovery*.

TCP PR

TCP PR (Persistent Reordering) [48] supone que el reordenamiento de paquetes es un evento relativamente común y, por lo tanto, los ACK duplicados no pueden considerarse indicadores confiables de pérdida de paquetes o congestión. En cambio, se enfoca en utilizar el tiempo de espera de

retransmisión como un indicador confiable de pérdida y congestión en una amplia gama de entornos de red.

TCP PR mantiene una marca de tiempo para cada paquete de datos transmitido. Se detecta una pérdida cada vez que la marca de tiempo de un paquete de datos es anterior al valor estimado de RTT Máximo (M). El concepto de RTT máximo es similar al RTO, pero difiere en la implementación. En lugar de recalcular RTO una vez por RTT, la estimación máxima M se reajusta en cada llegada de ACK según la siguiente Ecuación 24:

$$\mathrm{M} = \beta.\max\{\alpha^{1/\mathrm{cwmd}}.\mathrm{M}, \mathrm{RTT}\} \qquad \textbf{Ecuación 24}$$

Donde, α y β son constantes (0 <α <1, β> 1).

Sin embargo, TCP PR enfrenta el problema de la reacción exagerada a pérdidas múltiples del mismo evento de congestión, similar a Reno.

RR TCP

La especificación de la extensión SACK para TCP no define las acciones particulares que debe tomar un receptor, si recibe un paquete de datos que ya ha sido entregado. La especificación *DSACK (Duplicate Selective Acknowledgement)* complementa el estándar y proporciona una forma compatible con versiones anteriores para informar tales duplicados. DSACK requiere que el receptor informe cada recepción de un paquete duplicado al emisor.

La opción SACK en sí misma puede proporcionar mucha información sobre los patrones de entrega de paquetes. Por ejemplo, la ocurrencia de un evento de reordenamiento puede detectarse si el emisor recibe un paquete ACK selectivo seguido de un ACK acumulativo. Además, en este caso también se puede calcular cuánto tiempo se retrasó un paquete. Sin embargo, si hubo una retransmisión, se desconoce qué evento fue el causante de haber recuperado la pérdida de paquete previamente informada por el SACK.

RR TCP (Reordering Robust) [49] utiliza DSACK para resolver la ambigüedad de la retransmisión. De esta manera, el emisor conoce la secuencia de transmisión exacta, el orden de trasmisión y retransmisión de los paquetes

perdidos, y puede calcular la tasa de reordenamiento. No se puede calcular si se pierde el primer o segundo ACK.

En base a lo anterior, RR TCP puede calcular cuánto tiempo, en general, se retrasan los paquetes, y puede ajustar el umbral de ACK duplicados, denominado dupthresh y normalmente tiene un valor de 3. Este umbral es el que dispara la fase de Fast Recovery. Esto, en contraste con Eifel o DOOR, protegerá proactivamente al emisor de una reacción exagerada si los paquetes se han reordenado y no perdidos.

RR TCP incluye un concepto de bucle de control para encontrar el valor de dupthresh óptimo para una ruta determinada utilizando una función de costo combinado, incluidos tiempos de espera falsos y retransmisiones rápidas.

2.4.3. Tráficos de baja prioridad

Las aplicaciones tienen diferentes requisitos de transferencia de datos. Algunas tienen requisitos estrictos para la demora, mientras que otras no tienen requisitos particulares y son muy tolerantes con las condiciones de la red. En general, si se puede priorizar el tráfico del primer grupo de aplicaciones, se puede aumentar la calidad general del servicio percibido por el usuario en la red (quality of service o QoS).

La idea central entre las soluciones propuestas es hacer cumplir y garantizar, a través de políticas especiales de control de congestión, una distribución "injusta" de la red compartida entre flujos de alta y baja prioridad. Esta idea puede contradecir el requisito básico de equidad para el control de congestión TCP: un nuevo control de congestión no debería ser más agresivo que los algoritmos de control de congestión TCP “estándar” (Reno, New Reno y SACK). Sin embargo, si se restringe el alcance de QoS basado en TCP solo a un servicio de baja prioridad, entonces definitivamente se cumplirá con el requisito de imparcialidad. Es decir, el problema se limitaría a encontrar políticas de control de congestión que garanticen la liberación de los recursos cuando haya flujos TCP de alta prioridad.

TCP Nice [50] propone un mecanismo simple y distribuido Host-To-Host para minimizar la interferencia entre flujos de alta prioridad (primer plano) y de baja

prioridad (segundo plano). Las políticas de control de congestión de Nice se ajustan para reaccionar de manera muy conservadora a todos los cambios detectados en el estado de la red. En cierto sentido, Nice considera que todos los flujos TCP llevan datos de alta prioridad e intenta consumir los recursos de la red solo si nadie más los usa. Su diseño se basa en el algoritmo de Vegas pues incorpora un mecanismo de detección de congestión proactiva que permite redistribuir los recursos de red entre los flujos TCP competidores, sin inducir ninguna pérdida de paquetes. Además, debido a su naturaleza proactiva, un flujo TCP que ejecuta el algoritmo de control de congestión de Vegas tiene problemas para capturar su recurso compartido de red mientras compite con un flujo TCP reactivo.

TCP LP (Low Priority) [51] se basa en New Reno y tiene como objetivo proporcionar un servicio de transferencia de datos de baja prioridad para aplicaciones en segundo plano. En TCP LP, el cálculo del retardo de cola se refina progresivamente mediante el uso de estimaciones más precisas, haciendo uso de la opción timestamp y aplicando heurística para estimar el retraso de propagación unidireccional. Aunque esto puede complicar los cálculos de retardo de la cola, los valores resultantes son mucho más inmunes a la congestión en el canal inverso, por lo que el nivel de detecciones de congestión falsas se reduce sustancialmente. El proceso de detección de congestión consta de dos partes: TCP LP mantiene retrasos mínimos y máximos durante la vida útil de la conexión y, por cada RTT, compara la estimación actual de retardo unidireccional con un umbral predefinido.

La característica única del algoritmo TCP LP es su reacción a la detección temprana de la congestión. Tras la detección de un primer evento, reduce la ventana de congestión a la mitad del valor actual e inicia el temporizador de inferencia. Si el emisor desencadena otro evento de detección de congestión temprana antes de que transcurra el tiempo del temporizador, TCP LP infiere la presencia del flujo de alta prioridad y la ventana de congestión se reduce al valor mínimo. En otros casos, LP reanuda las acciones de la fase de Congestion Avoidance.

El uso generalizado de redes inalámbricas y de alta velocidad limita la aplicabilidad de Nice o LP, debido a la ineficacia de los algoritmos de referencia en esos entornos.

2.4.4. Redes de alta velocidad y alta latencia

La aparición de redes de alta velocidad puso de manifiesto la incapacidad de las variantes de TCP tales como Reno, New Reno, SACK entre otras, para utilizar los recursos de estas redes de manera eficiente. En general, los algoritmos de control de congestión mejoran diferentes aspectos de la eficiencia para la transferencia de datos, sin cuestionar el principio básico en el que descansa, que se definió en 1988 como parte de Tahoe: el descubrimiento de los recursos disponibles de red, durante la fase de Congestion Avoidance, debe ser muy conservador [52]. En las implementaciones de TCP, este principio se realizó generalmente con una ventana de congestión (cwnd) que aumenta en un paquete por cada RTT si no se detectan errores. Esto funciona bastante bien solo si la capacidad de la red o las demoras de ida y vuelta (RTT) son relativamente bajas [53].

El tiempo mínimo requerido para que un flujo TCP descubra el límite de los recursos de un canal, suponiendo que no hay pérdidas de paquetes, es del orden del producto del retardo por ancho de banda del canal, BDP (Bandwidth Delay Product). En redes de alta velocidad con grandes BDP, el TCP convencional puede tener un comportamiento poco eficiente, porque sus algoritmos tardan mucho tiempo en saturar los enlaces. De esta manera, TCP puede no aprovechar las redes rápidas, incluso cuando no hay congestión, debido principalmente al incremento aditivo fijo de la fase de Congestion Avoidance. Aun suponiendo que la tasa de pérdidas es muy baja, el tiempo necesario para que TCP utilice en forma completa estos enlaces, es muy alto. También requiere de mucho tiempo para recuperar el nivel de uso del enlace después de una pérdida.

Además, TCP no es estable con valores altos de RTT. Si se compara el crecimiento de la ventana de congestión de dos flujos TCP, uno con un valor alto de RTT y el otro con un valor bajo, puede demostrarse que el flujo con mayor valor de RTT pasa la mayor parte del tiempo aumentando la ventana de

congestión, mientras que el otro, en el mismo periodo, alcanza el pico en varias ocasiones.

Para abordar esta deficiencia, varios investigadores y desarrolladores han explorado modificaciones de TCP para que su funcionamiento sea más eficiente en este tipo de redes. Sin embargo, deben conservar un grado de equidad con el TCP "estándar", debido a que, en las redes heterogéneas actuales, conviven distintas variantes de TCP y recorren distintos tipos de enlaces. Aunque las soluciones propuestas a lo largo del tiempo se basan en diferentes suposiciones y enfoques, tienen el mismo objetivo: mejorar el rendimiento de TCP en enlaces de alta velocidad o de gran latencia.

High-Speed TCP (HS TCP)

HS-TCP (High-Speed TCP) (RFC3649 [54], RFC3742 [55]) reemplaza el coeficiente de incremento de la fase Congestion Avoidance del estándar New Reno y el coeficiente de disminución después de una detección de pérdida menor, durante la fase de Fast Recovery (valores constantes en Reno), por valores en función del tamaño de ventana de congestión, $f_\alpha(cwnd)$ y $f_\beta(cwnd)$. Estas funciones se definen en términos del tamaño de ventana de congestión alcanzable y la tasa de pérdida requerida. Si la ventana de congestión es baja, HS-TCP se comporta como el TCP Reno. Sin embargo, una vez que el tamaño de la ventana de congestión supera un umbral definido, aumenta de manera agresiva. El algoritmo de HS-TCP es AIMD, tiene incremento aditivo (Ecuación 25) y decremento multiplicativo (Ecuación 26) como lo describen las siguientes ecuaciones,

$$\mathrm{cwnd} = \mathrm{cwnd} + \mathrm{f}_\alpha(\mathrm{cwnd}) \qquad \textbf{Ecuación 25}$$

$$\mathrm{cwnd} = \left(1 - \mathrm{f}_\beta(\mathrm{cwnd})\right) * \mathrm{cwnd} \qquad \textbf{Ecuación 26}$$

Cuando el tamaño de la ventana está por debajo del umbral, por ejemplo, la ventana de congestión es menor o igual a 38 segmentos, las funciones toman los valores $f_\alpha(cwnd)=1$ y $f_\beta(cwnd)=0,5$, lo que produce que se comporte como New Reno. En el otro extremo, cuando la ventana de congestión es grande, las funciones pueden tomar valores $f_\alpha(cwnd)=70$ y $f_\beta(cwnd)=0,1$. De esta manera,

cuando el tamaño de cwnd es grande y la tasa de pérdidas es baja, HS-TCP prueba los recursos de red de forma más agresiva que Reno y, al mismo tiempo, reacciona de forma más conservadora a los eventos de detección de pérdida. Este comportamiento aumenta considerablemente la eficiencia de las redes de alta velocidad y/o alta latencia. Sin embargo, se obtiene un mayor nivel de pérdida de paquetes: se pierden más paquetes durante los eventos de congestión, y estos ocurren con mayor frecuencia.

Durante la fase de Slow Start, cuando aún se desconoce un límite de la red, el sondeo exponencial propio de esta fase puede conducir a la pérdida de un número extremadamente grande de paquetes. Para resolver este problema, se propuso un algoritmo complementario, denominado *Limited Slow Start* [55], que limita el aumento del tamaño de la ventana durante la fase de Slow Start.

En redes de alta velocidad y alta latencia, HS-TCP explícitamente no considera que la equidad con los flujos TCP estándar sea significativamente importante, porque los flujos estándar no pueden utilizar de manera efectiva los recursos disponibles de estas redes. Sin embargo, por definición, durante situaciones de congestión grave HS-TCP es equivalente al Reno y, por lo tanto, hereda todas sus características. HS-TCP tiene problemas sustanciales con la equidad si los flujos tienen diferentes RTT. Aunque este problema se hereda de Reno, la investigación posterior descubrió que la escala del coeficiente AIMD (funciones en lugar de constantes) intensifica significativamente este problema.

Scalable TCP (STCP)

Se propuso [56] como alternativa al HS-TCP, que rechaza el concepto central de AIMD e introduce la idea de disminución multiplicativa aumento multiplicativo (MIMD). Durante la fase de Congestion Avoidance, incrementa el tamaño de la ventana de congestión cwnd en una fracción α de este tamaño por cada RTT (Ecuación 27). Durante la fase de Fast Recovery, al detectar una pérdida, reduce la ventana de congestión en una fracción diferente β (Ecuación 28), como se describe en las siguientes ecuaciones,

$$\text{cwnd} = \text{cwnd} + \alpha * \text{cwnd} \qquad \alpha = 0.01 \qquad \textbf{Ecuación 27}$$

$$\text{cwnd} = \text{cwnd} - \beta * \text{cwnd} \qquad \beta = 0.125$$ **Ecuación 28**

Las modificaciones propuestas resuelven el problema haciendo que las dinámicas de aumento y disminución sigan funciones exponenciales, que escalan bastante bien en muchos entornos. Sin embargo, la solución crea una serie de problemas críticos. Un flujo STCP lleva a la red a un estado de congestión casi constante. Esto es generalmente indeseable para la mayoría de las redes. Además, el enfoque MIMD no proporciona equidad entre los flujos STCP. Debido a estas políticas, un flujo STCP es extremadamente injusto, tanto para STCP como para flujos TCP que tienen valores RTT más altos.

Hamilton TCP (H-TCP)

En el diseño de *Hamilton TCP* [57], la tasa de incremento del tamaño de cwnd está determinada por el tiempo transcurrido desde el evento de pérdida anterior y no por el tamaño de la ventana. De esta manera, la tasa de incremento de la ventana de congestión, *α* en la fase de Congestion Avoidance, es una función no decreciente del tiempo transcurrido desde el último evento de congestión (Δ). Esto tiene una ventaja significativa sobre la dependencia de la ventana de congestión: no importa qué tan grandes hayan sido los tamaños de ventana de congestión iniciales, los flujos que experimentan las mismas condiciones de red exhibirán el mismo aumento de ventana de congestión dinámica. En otras palabras, un flujo de H-TCP es justo para otros flujos de H-TCP presentes en la misma ruta de red.

En la fase de Congestion Avoidance se incrementa el tamaño de la ventana de congestión cwnd de la siguiente manera (Ecuación 29) por cada RTT,

$$\text{cwnd} = \text{cwnd} + \alpha(\Delta) * \text{cwnd}$$ **Ecuación 29**

El valor de esta función depende de un umbral definido ΔLow. Si Δ es el tiempo transcurrido en segundos desde el evento de pérdida anterior, entonces para Δ≤Δlow, α(Δ)=1, es decir que el incremento de ventana es 1 por cada RTT. Sin embargo, para Δ>Δlow, α(Δ) se define según la siguiente Ecuación 30,

$$\alpha(\Delta) = 1 + 10 * (\Delta - \Delta\text{low}) + 0.5 * (\Delta - \Delta\text{low})^2$$ **Ecuación 30**

Como se observa, después de un evento de pérdida H-TCP compite justamente con TCP Reno, en el sentido de que ambos aumentan cwnd a la misma tasa y no vuelva a ingresar a su modo de "alta velocidad", incluso aunque el valor de cwnd sea grande y sólo lo hace después del tiempo Δlow.

TCP Hybla

En redes heterogéneas, especialmente con segmentos satelitales, los valores de RTT pueden ser diferentes en varios órdenes de magnitud, lo que puede resultar en una injusta distribución de los recursos de la red. Para resolver este problema, se ha propuesto el algoritmo Hybla. Este algoritmo [58] introduce modificaciones a las fases de Slow Start y Congestion Avoidance de New Reno, para lograr independizarlas del valor de RTT. Para obtener las tasas de aumento normalizados de cwnd en ambas fases, el factor de escala ρ se calcula según la siguiente ecuación (Ecuación 31).

$$\rho = \frac{\mathrm{RTT}}{\mathrm{RTTref}} \qquad \textbf{Ecuación 31}$$

Donde, RTTref es un valor de referencia (por ejemplo, 25 ms).

Al recibir un paquete ACK el incremento del tamaño de la ventana de congestión se define para la fase de Slow Start en la Ecuación 32 y para la fase de Congestion Avoidance en la Ecuación 33,

$$cwnd = cwnd + 2^{\rho} - 1 \qquad \textbf{\textit{Slow Start}} \qquad \textbf{Ecuación 32}$$

$$cwnd = cwnd + \frac{\rho^2}{cwnd} \qquad \textbf{\textit{Congestion Avoidance}} \qquad \textbf{Ecuación 33}$$

Como se observa en las ecuaciones, cuanto mayor sea el valor de RTT, mayor será el valor de ρ y, por lo tanto, la ventana de congestión se incrementará más rápidamente con cada ACK. Como consecuencia, durante el mismo período de tiempo, flujos con distintos valores de RTT, arribaran a diferentes tamaños de sus ventanas de congestión. Sin embargo, si se calcula la relación entre el valor de la ventana de congestión y el valor de RTT, se observa que los flujos pueden transmitir datos a velocidades similares.

Hybla introduce dos técnicas más que complementan el control de la congestión: ritmo de la transmisión de paquetes de datos y estimación del valor inicial del umbral de Slow Start (ssthresh). Para regular el ritmo de la transmisión, determina un retraso mínimo entre la transmisión de dos paquetes consecutivos para suavizar la naturaleza de ráfaga de las transmisiones TCP.

Debido a que cwnd aumenta en ρ^2 por cada RTT en la fase de Congestion Avoidance, su valor que puede ser relativamente grande. De esta mamera, puede ocurrir que cuando se alcance el límite de la capacidad de la red, ocurra una pérdida de un número importante de segmentos. Por esta razón, para permitir una recuperación más rápida en pérdidas múltiples, se recomienda que los extremos de la conexión implementen el uso de la opción SACK.

Hybla, en forma similar a Reno, no funciona eficientemente en redes de alta velocidad con baja latencia, lo que limita su aplicación a las redes con las características de las redes satelitales.

CUBIC TCP

En 2004 fue introducido BIC TCP (Binary Increase Congestion control), para resolver lo que se denominó “RTT fairness” (equidad entre flujos con valores distintos de RTT) [59]. Es un intento de crear un control de congestión que pueda escalarse bien en cualquier red de alto BDP y aun así lograr una relativa equidad RTT.

Este algoritmo extiende el control de congestión de New Reno con una fase operacional adicional denominada Rapid Convergence. Esta fase descubre rápidamente, mediante el uso de la búsqueda binaria, el tamaño óptimo de la ventana de congestión, confiando en una pérdida de paquetes como una indicación del crecimiento excesivo de la ventana.

Mientras la red entrega paquetes de datos, la ventana de congestión se actualiza a la mitad del valor del rango entre los dos valores umbrales W_{min} y W_{max}. Una entrega exitosa de datos, eleva el límite inferior (W_{min}) al valor de cwnd. En el caso contrario, cuando se detecta una pérdida, BIC establece el límite superior de búsqueda W_{max} con el tamaño actual de la ventana de congestión y entra en la conocida fase de Fast Recovery.

Usando una técnica de búsqueda binaria, BIC TCP selecciona una ventana de prueba en el punto medio de estos dos valores. Si este punto muestra una pérdida continua de paquetes, se convierte en el nuevo máximo y el proceso se repite. Si no, se convierte en el nuevo mínimo y el proceso se repite. El proceso finaliza cuando la diferencia entre las ventanas mínima y máxima es menor que un umbral predefinido denominado incremento mínimo, o S_{min}. La ventana mínima actual es el último punto en el que la conexión no experimentó pérdida de paquetes durante un RTT completo.

Aunque un verdadero algoritmo de búsqueda binaria presenta un tiempo de convergencia muy rápido (logarítmico), en una red de alto BDP puede crear un problema: si la ventana de congestión se incrementa demasiado rápido, se puede perder una cantidad importante de paquetes. Por este motivo, BIC no solo adopta el Limited Slow Start [55] de HS-TCP, sino que también limita el aumento en la fase *Rapid Convergence* cuando el rango de búsqueda es demasiado amplio.

Cuando aumenta la ventana en forma binaria y la distancia desde el tamaño de la ventana actual hasta el punto medio es grande, puede ocurrir que TCP intente llegar al punto medio en un solo RTT inyectando grandes ráfagas de paquetes en la red. Para evitar este comportamiento, se define un valor S_{max}, de tal forma que cuando la distancia al punto medio del intervalo (W_{min}, W_{max}), desde el tamaño de la ventana de congestión actual es mayor que S_{max}, el incremento se limita a S_{max} por RTT, a lo que se llama *bloqueo de ventanas (Window Clamping)*. Una vez que el punto medio está más cerca que S_{max} en la ventana de prueba, el aumento de la búsqueda binaria se hace cargo.

El enfoque BIC para el descubrimiento óptimo de la ventana de congestión tiene una característica única para los enfoques de control de la congestión basados en la pérdida: los pasos de sondeo de la ventana de congestión disminuyen a medida que la ventana se acerca a un valor objetivo.

Los autores de BIC TCP revisaron sus algoritmos básicos para formar un nuevo algoritmo de control llamado *CUBIC* [60]. En lugar de usar un umbral S_{max} utiliza una función polinómica cúbica, para controlar el crecimiento de la ventana. Las funciones cúbicas pueden tener porciones convexas y cóncavas,

lo que significa que pueden crecer más lentamente en algunas partes (cóncavas) y más rápidamente en otras (convexas).

Algunas de las propuestas de control de congestión anteriores (HS-TCP, STCP, BIC) imponen la equidad con los flujos TCP “estándar”, al cambiar las reglas de actualización de ventana de congestión en entornos de gran pérdida. El criterio de conmutación en la mayoría de los casos es un tamaño de ventana de congestión que corresponde a una cierta tasa de pérdida. Sin embargo, este criterio basado en la tasa de pérdida no es lo ideal, especialmente en redes heterogéneas donde el valor de RTT puede variar significativamente. Así, si dos flujos que compiten en el mismo cuello de botella, tienen diferentes RTT, es probable que el flujo con el mayor valor de RTT permanezca en un modo compatible todo el tiempo, mientras que el flujo, con el RTT más pequeño, cambia rápidamente a un modo escalable y adquiere todos los recursos disponibles. Esta observación permitió proponer el control de congestión CUBIC, que mejora el algoritmo BIC con funciones de crecimiento de ventana de congestión independientes de RTT. Para lograr esto, CUBIC toma prestado el enfoque H-TCP de definir cwnd como una función cúbica del tiempo transcurrido Δ desde el último evento de congestión, de acuerdo a la Ecuación 34,

$$\mathrm{cwnd} = \mathrm{C} * \left(\Delta - \sqrt[3]{\beta * \frac{\mathrm{Wmax}}{\mathrm{C}}} \right)^3 + \mathrm{Wmax} \qquad \textbf{Ecuación 34}$$

Donde C es una constante predefinida, β es un coeficiente de disminución multiplicativa en Fast Recovery, y W_{max} es el tamaño de la ventana de congestión justo antes de la última detección de pérdida registrada.

Esta función (Ecuación 34) no solo conserva la equidad RTT, ya que el crecimiento de la ventana no depende en gran medida del valor de RTT, sino también las propiedades de escalabilidad de las fases de BIC Limited Slow Start y Rapid Convergence. La función tiene un crecimiento muy rápido cuando la ventana actual está lejos del objetivo estimado y es muy conservadora cuando está cerca.

El polinomio cúbico que se observa en la siguiente figura (Figura 19), adecuadamente desplazado y escalado, se usa para determinar los cambios en el tamaño de cwnd. El punto es que la curva, aunque aumenta constantemente, es primero cóncava y luego convexa.

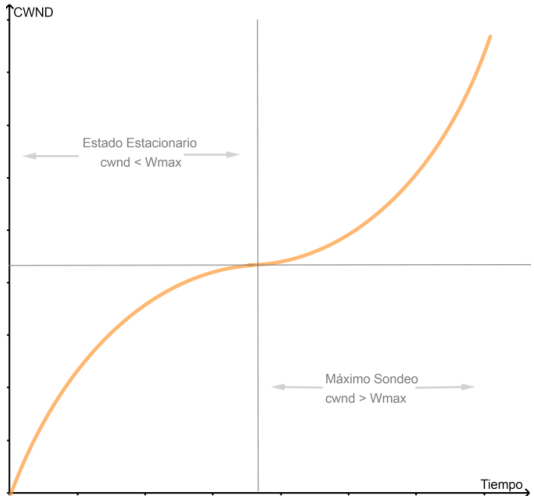

Figura 19: Polinomio Cúbico

En la Figura 19, se puede observar la sección cóncava de la función cubica, donde el cwnd<Wmax, permite el aumento rápido inicial en el tamaño de cwnd después de una pérdida y, cuando crece hasta valores cercanos de Wmax, la tasa de crecimiento se reduce considerablemente. A partir del momento en que el cwnd es mayor que Wmax, la función de crecimiento se vuelve convexa. En este punto, el emisor supone que la capacidad de la red ha aumentado desde la última pérdida. La estrategia de CUBIC es descubrir rápidamente esa capacidad adicional, por lo que aumentará el tamaño de la ventana en forma agresiva hasta encontrar el nuevo límite de la red. La función ubica es bastante agresiva cuando se compara con otras variantes de TCP. Un valor mayor de la constante C en la Ecuación 34, reduce el tiempo entre el evento de pérdida y el punto de inflexión (cambio de concavidad en la Figura 19), y por lo tanto, el tiempo entre pérdidas consecutivas.

CUBIC proporciona un mecanismo para garantizar que su rendimiento no sea inferior al rendimiento del control de congestión de Reno. Este mecanismo incluye el cálculo de un tamaño de ventana de congestión adicional que se aproxima al rendimiento de un flujo de Reno estándar correspondiente y, si es mayor que la ventana de CUBIC, utiliza este tamaño de ventana.

CUBIC es actualmente uno de los algoritmos de control de congestión más utilizado para TCP, debido a que ha sido el predeterminado para el paquete de Linux TCP desde 2006. Sin embargo, tiene como inconvenientes la infrautilización del ancho de banda disponible, dado que CUBIC no utiliza el 100% del ancho de banda disponible, y puede producir un gran número de pérdidas de paquetes.

FAST TCP

Inspirado en la idea de Vegas [61], de utilizar el retraso de cola como indicador de congestión, define una actualización periódica de la ventana de congestión, basada en la estimación del estado de la red, a través del análisis del retraso observado.

A diferencia de Vegas, define una actualización periódica a tasa fija de la ventana de congestión y, para calcular el nuevo tamaño de ventana de congestión, usa una ecuación (Ecuación 35) especialmente diseñada que incorpora una función simple de estimación de la congestión basada en el retraso de la siguiente forma,

$$\mathrm{cwnd} = \mathrm{cwnd} * \frac{\mathrm{RTTmin}}{\mathrm{RTT}} + \alpha \qquad \textbf{Ecuación 35}$$

Donde, cwnd es el tamaño de ventana de congestión actual, RTT es el valor de RTT medido, RTTmin es el menor valor observado de RTT, y α es un parámetro del protocolo.

En la Ecuación 35 se puede observar que si la red experimenta congestión ($RTT>RTT_{min}$), FAST disminuirá la ventana de congestión proporcionalmente al nivel de congestión estimado mediante mediciones de RTT. En el caso contrario, la ventana se incrementará basándose únicamente en el parámetro predefinido α. Si α es demasiado grande, el protocolo se escalará fácilmente a cualquier red con un alto BDP, pero tendrá problemas de convergencia. En el caso opuesto, cuando α es demasiado pequeño, se estabilizará fácilmente, pero tendrá problemas de escalabilidad.

Las características de FAST dependen, en gran medida, del verdadero valor mínimo de RTT, que es difícil de calcular en algunos entornos sin depender de

mensajes adicionales de la red. Tampoco el RTT es siempre un buen sustituto para el retardo en cola, especialmente cuando hay congestión a lo largo del camino inverso o cuando hay cambios de ruta. Finalmente, no es compatible con TCP "estándar" (Reno, New Reno o SACK).

TCP Libra

El diseño de *TCP Libra* [62] se basa en New Reno y modifica la tasa de aumento de la ventana de congestión en la fase de Congestion Avoidance, mediante el uso de una función que depende del valor RTT y de la estimación de la longitud de la cola. El algoritmo de Congestion Avoidance de Libra incrementa la ventana de congestión en α paquetes cada RTT, de acuerdo con la siguiente ecuación (Ecuación 36),

$$\alpha = k_1 * C * P * {RTT^2}/{(RTT + \gamma)} \qquad \textbf{Ecuación 36}$$

Donde, RTT es RTT actual, γ y k_1 son constantes predefinidas, C es un valor responsable de la escalabilidad de Libra y representa la capacidad estimada del enlace y P es un factor de penalización que reduce la tasa de incremento si la red experimenta congestión.

El factor de penalización P se puede representar con la siguiente expresión (Ecuación 37), basada en mediciones del retardo de cola,

$$P = e^{-k_2 * Q/Q_{max}} \qquad \textbf{Ecuación 37}$$

Donde k_2 es una constante, y Q es la estimación de retardo actual y Q_{max} es el retardo máximo observado.

En la Ecuación 36, la tasa de crecimiento de la ventana de congestión de TCP Libra, α, muestra un primer término (k_1*C) que hace que los pasos de incremento sean escalables a la capacidad del enlace. La penalización P fuerza a Libra a disminuir la intensidad de exploración de los recursos de red, disminuyendo la tasa de crecimiento exponencialmente, si se incrementa el valor estimado de la longitud de cola ($\frac{Q}{Qmax}$). La última parte, ($\frac{RTT2}{RTT + \gamma}$), es responsable de la equidad RTT (RTT Fairness) de Libra.

Libra también define un cambio en la política de disminución multiplicativa de la fase de Fast Recovery expresado en la Ecuación 38,

$$\mathrm{cwnd} = \mathrm{cwnd} - \beta * \mathrm{cwnd} \qquad \textbf{Ecuación 38}$$

Donde, el coeficiente β se escala con la siguiente expresión (Ecuación 39)

$$\beta = \frac{\theta}{(\mathrm{RTT} + \gamma)} \qquad \textbf{Ecuación 39}$$

Donde, θ y γ son constantes.

Cuando el valor actual de RTT es grande (congestión potencial en la red), el factor de escala reducirá el valor β. Sin embargo, esto es lo contrario de lo que debería hacer el control de la congestión: la disminución debería maximizarse en presencia de congestión y minimizarse cuando la red se encuentra en un estado libre de congestión.

Libra puede ayudar a mejorar la utilización de enlaces de alto BDP y las propiedades de equidad de TCP. Sin embargo, los mismos resultados muestran que Libra no siempre supera a otros enfoques de control de la congestión. Además, debido a la alta dependencia de la estimación del retardo de la cola, las propiedades de Libra empeorarán debido a sesgos de estimación.

TCP New Vegas

El algoritmo propuesto de New Vegas [63] define una fase llamada *Rapid Window Convergence*. La idea clave de esta nueva fase es la de no terminar la fase de Slow Start cuando la estimación del almacenamiento en buffer de la red excede el umbral, y continuar el sondeo de recursos de tipo exponencial con intensidad reducida.

Cuando Slow Start de New Vegas detecta que se ha excedido el umbral, recuerda el valor de la ventana de congestión actual en una variable de estado especial wr y cambia a la Rapid Window Convergence. En este estado, la ventana de congestión se puede aumentar en x paquetes por cada RTT, donde x se define por la siguiente ecuación (Ecuación 40),

$$x = w_r^{-2^{3+n}}$$ **Ecuación 40**

Donde, n es el número de veces que el indicador de congestión temprana se activa en la fase de Rapid Window Convergence.

Cuando el valor de n es mayor que 3 (n>3), termina la fase de Rapid Window Convergence y comienza una fase Congestion Avoidance de Vegas. En cualquier punto, si se detecta una pérdida de paquete, New Vegas reacciona exactamente igual que los algoritmos originales de Vegas. En otras palabras, si la pérdida se detecta usando tres ACK duplicados, New Vegas cambia a Fast Recovery seguida de Congestion Avoidance; si la pérdida se detecta utilizando el timeout, New Vegas restablece el tamaño de la ventana de congestión y pasa a Slow Start.

Para resolver el problema de generación de tráfico en ráfagas durante la inicialización y la reinicialización, New Vegas aplica la técnica de estimulación de paquetes, estableciendo una demora mínima entre la transmisión de dos paquetes consecutivos.

TCP-AR (Adaptive Reno)

TCP AR [64] define una ventana de congestión escalable en la fase de Congestion Avoidance. La función de aumento de la ventana de congestión se define para tener dos componentes: un componente de aumento constante lento que aumenta en uno por cada RTT, y un componente de aumento escalable que se incrementa en función de la estimación de throughput alcanzable y el retraso de cola.

El componente escalable es una función continua que tiene dos propiedades importantes. En primer lugar, cuando la red está libre de congestión, es decir, cuando el retardo de la cola es cercano a cero, la función toma un valor cercano a la estimación. En segundo lugar, cuando la red experimenta congestión o, dicho de otro modo, cuando el retardo de la cola es casi máximo, el valor del componente escalable es cero.

TCP AR mejora la utilización de la red y, mantiene una buena equidad Intra-protocolo. Sin embargo, depende de la estimación del retardo en las colas y las

métricas de throughput alcanzables, lo que hace que este algoritmo tenga las mismas vulnerabilidades si no obtiene el valor real de RTT.

TCP Fusion

En la fase de Congestion Avoidance, TCP Fusion [65], define tres funciones lineales separadas, que se conmutan dependiendo de un valor umbral de retardo de cola. Si el retardo actual es menor que el umbral predefinido, la ventana de congestión se incrementa a una tasa rápida por cada RTT, en una fracción predefinida de la estimación de throughput alcanzable (aumento escalable). Si el retardo de la cola aumenta más de tres veces el umbral, la ventana de congestión disminuye en función de la cantidad de paquetes almacenados en la red (estimación de Vegas). En el caso que el retardo se encuentre en algún punto del rango entre uno y tres veces el umbral, la ventana de congestión permanece sin cambios.

Para hacer que Fusion se comporte al menos tan bien como el control de congestión estándar de Reno, se calcula una ventana adicional similar a la de Reno junto con la ventana de congestión de Fusion.

TCP Fusion cambia la tasa de reducción de ventana de congestión β en la fase de Fast Recovery (Ecuación 38) de la siguiente manera (Ecuación 41),

$$\beta = \max\left(0.5, {}^{\mathrm{RTT}_{\min}}/_{\mathrm{RTT}}\right) \qquad \textbf{Ecuación 41}$$

Fusion ha mostrado algunas mejoras en términos de utilización de los recursos y características de equidad, en comparación con otros algoritmos escalables. Sin embrago, Fusion, además de tener las mismas vulnerabilidades que TCP AR, presenta un nuevo problema: la definición del umbral requiere la adaptación manual de Fusion a un entorno particular.

TCP Africa (Adaptive and Fair Rapid Increase Congest Congestion)

Este algoritmo [66] combina la escalabilidad de HS-TCP, cuando se determina que la red está libre de congestión, y el carácter conservador de New Reno, cuando la red experimenta congestión.

Para definir el estado de congestión, o no congestión, utiliza el método de TCP Vegas: la estimación del almacenamiento en buffer de la red Δ, se compara

con una constante predefinida α. Si África ve que hay pocos paquetes en las colas (Δ<α), pasa al "modo rápido" y aplica directamente las reglas de HS-TCP de las fases de Congestion Avoidance y Fast Recovery. Esto determina que tasas de crecimiento y decrecimiento de la ventana de congestión son funciones de la propia ventana de congestión. De lo contrario, aplica las reglas de Reno: aumentar en uno, disminuir a la mitad.

África mostró una buena utilización de los recursos en las redes de alto BDP, menor tasa de pérdida inducida en comparación con HS-TCP y STCP, y propiedades de equidad (intra, inter, RTT) comparables. La idea del control de congestión de modo múltiple para redes de alto BDP, con cambio de modo basado en retardo, ha sido ampliamente adoptada por diferentes propuestas, por ejemplo, el algoritmo C-TCP (Compound TCP) de modo dual que es actualmente uno de los controles de congestión de TCP más implementado en el mundo, ya que está integrado en el sistema operativo Microsoft Windows.

Compound TCP (C-TCP)

C-TCP [67] hace los ajustes del tamaño de la ventana basados en la pérdida de paquetes, pero también en función del retardo medido. En cierto sentido, es una combinación de TCP "estándar" y Vegas, pero con las características de escalabilidad de HS-TCP. También trata de usar una estimación del estado de la red basada en el retraso, para combinar el control de la congestión convencional de tipo Reno, con un control de la congestión que es escalable en redes de alta BDP.

C-TCP intenta abordar esta situación combinando un enfoque basado en el retraso, con un enfoque basado en la pérdida. Sin embargo, en lugar de definir explícitamente los modos rápido y lento, C-TCP define un componente escalable adicional que se agrega a los cálculos de la ventana de congestión. Introduce una nueva variable de control de ventana llamada dwnd (delay window), de manera que la ventana W de la Ecuación 16 se expresa de la siguiente manera (Ecuación 42),

$$W = \min(\text{rwnd}, \text{cwnd} + \text{dwnd}) \qquad \textbf{Ecuación 42}$$

Donde cwnd es la ventana de congestión, dwnd es la ventana de retardo (*delay window*) y rwnd es la ventana publicada por el receptor.

El manejo de cwnd es similar al de TCP estándar, pero el agregado del componente dwnd puede permitir que se envíen paquetes adicionales si las condiciones de demora son las apropiadas. Esta nueva ventana se actualiza sólo cuando la estimación de Vegas, Δ, muestra que la red no presenta congestión (Δ<γ).

Cuando la estimación Δ excede el umbral γ, el componente escalable dwnd se reduce suavemente en un valor proporcional a la estimación en sí misma de la siguiente manera (Ecuación 43),

$$\mathrm{dwnd} = \mathrm{dwnd} - \zeta * \Delta \qquad \textbf{Ecuación 43}$$

Donde, ζ es una constante predefinida.

Durante la fase de Congestion Avoidance, cuando se recibe un ACK, la ventana de congestión se actualiza de la siguiente manera (Ecuación 44),

$$\mathrm{cwnd} = \mathrm{cwnd} + \frac{1}{(\mathrm{cwnd} + \mathrm{dwnd})} \qquad \textbf{Ecuación 44}$$

La gestión de dwnd se basa en Vegas y solo es distinto de 0 en la fase de Congestion Avoidance, pues C-TCP usa el Slow Start convencional. Durante la fase de CA, el proceso de control para dwnd se expresa de la siguiente manera (Ecuación 45, Ecuación 46 y Ecuación 47):

$$\mathrm{dwnd} = \mathrm{dwnd} + (\alpha * \mathrm{cwnd}^{k} - 1) \qquad \text{Si } \Delta<\gamma \qquad \textbf{Ecuación 45}$$

$$\mathrm{dwnd} = \mathrm{dwnd} - \zeta * \Delta \qquad \text{Si } \Delta>\gamma \qquad \textbf{Ecuación 46}$$

$$\mathrm{dwnd} = \mathrm{cwnd} * (1 - \beta) - \frac{\mathrm{cwnd}}{2} \qquad \text{ante una perdida} \qquad \textbf{Ecuación 47}$$

Donde, Δ está definido por la Ecuación 23.

En la Ecuación 45, se describe un comportamiento donde la red puede estar infrautilizada y, C-TCP crece de acuerdo con el polinomio $\alpha*\mathrm{cwnd}^{k}$. En la Ecuación 46, se observa que la longitud de las colas parece estar creciendo

más allá del umbral α deseado y, por lo tanto, la constante de ξ dicta cómo el componente basado en el retraso se debe reducir rápidamente.

Cuando se detecta la pérdida (Ecuación 47), el valor de dwnd tiene su propio factor de disminución multiplicativo β.

El valor de k en la Ecuación 45, afecta el nivel de agresividad y su valor predeterminado es de k=0.75. Los valores de α (Ecuación 45) y β (Ecuación 47) afectan la suavidad y la capacidad de respuesta, sus valores predeterminados son α=0.125 y β=0.5, respectivamente. Para γ, se sugiere un valor predeterminado de 30 paquetes basado en la evaluación empírica.

C-TCP logra buena utilización de los enlaces de alto BDP y buenas propiedades de equidad. Sin embargo, puede ser víctima de algunos de los problemas heredados de Vegas, incluido el cambio de rutas y congestión persistente en el camino inverso. Además, se observa que, si muchos flujos Compound TCP comparten el mismo camino, el rendimiento puede ser bajo.

TCP Illinois

Está diseñado [68] para ser muy agresivo cuando determina que la red está en un estado libre de congestión y, por otro lado, ser muy conservador cuando la red está experimentando congestión. En forma similar al TCP “estándar”, administra el tamaño de la ventana de congestión usando dos parámetros α y β y se basa en la pérdida de paquetes para definir el valor de la ventana de congestión. Sin embargo, los valores de α y β, no son constantes, se actualizan en base a la estimación del retraso de la red. Define la tasa de crecimiento α en la fase de Congestion Avoidance ($cwnd = cwnd + \alpha * cwnd$) y la tasa de disminución β en la fase de Fast Recovery ($cwnd = cwnd - \beta * cwnd$) de la ventana de congestión, para ser funciones del retardo estimado de cola. El cálculo del retardo sigue la definición de Vegas.

La idea es estimar el retraso promedio máximo en cola, como la diferencia entre los tiempos promedios de ida y vuelta máximos y mínimos observados por el flujo durante la conexión. A continuación, estimar el retraso de cola actual y, si es cercana a la demora máxima estimada, entonces establecer α pequeña y β grande y, en el caso contrario, a la inversa. El algoritmo vuelve a la operación del TCP “estándar” si la ventana de congestión está por debajo de

un umbral. Una ventaja de este algoritmo es que, al adaptar el parámetro β de esta manera, aumenta la inmunidad frente a la pérdida de paquetes no debidos a la congestión. Sin embargo, si un receptor retrasa un ACK puede aumentar el tiempo de ida y vuelta, por lo tanto, puede suceder que el emisor sobreestime el estado de congestión.

En la siguiente figura (Figura 20) se observa que el coeficiente de aumento α depende inversamente del retardo de la cola, mientras que el coeficiente de disminución β es directamente proporcional.

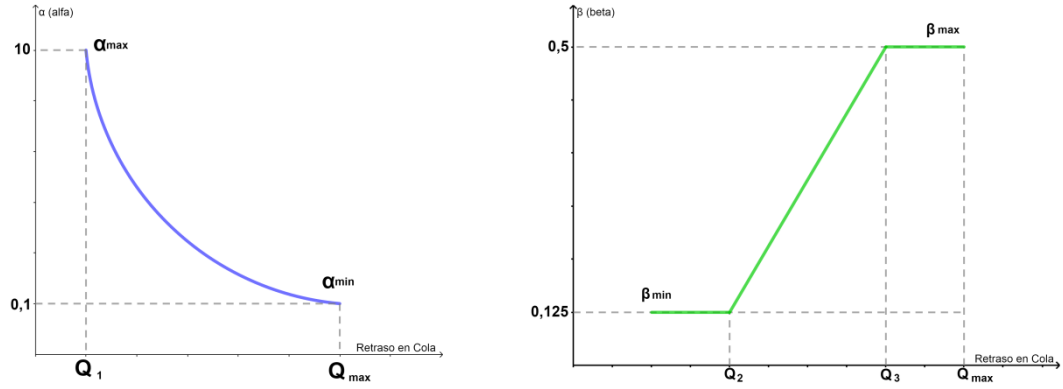

Figura 20: Coeficientes α y β como función del retraso en cola

Los valores mínimos y máximos de α y β, y los umbrales de retardo en cola Q_1, Q_2 y Q_3, se pueden modificar para lograr las características de rendimiento deseadas. En la implementación de Linux, Illinois establece los valores predeterminados de $\alpha_{max}=10$, $\alpha_{min}=0.3$, $\beta_{min}=0.125$, $\beta_{max}=0.5$, $Q_1=0.01*Q_{max}$, $Q_2=0.1*Q_{max}$ y $Q_3=0.8*Q_{max}$, donde Q_{max} es el retardo máximo de cola observado durante la vida útil de la conexión.

De acuerdo con la especificación de Illinois, los coeficientes α y β se actualizan una vez por cada RTT. Sin embargo, para mitigar los efectos del ruido de medición del retardo de cola, se permite establecer el coeficiente α al máximo, sólo si durante n RTT consecutivos el valor del retardo de cola es menor que el primer umbral Q1. Además, Illinois cambia al modo de compatibilidad (α = 1 y β = 0.5) cuando el tamaño de la ventana de congestión es menor que un umbral predefinido. Este cambio, similar a HS-TCP y a STCP, mejora las propiedades de equidad de Illinois hasta cierto punto, por lo que se comporta como New Reno durante los eventos de congestión graves.

Cuando recibe un ACK, el tamaño de la ventana de congestión se incrementa de la siguiente manera (Ecuación 48),

$$\mathrm{cwnd} = \mathrm{cwnd} + \frac{\alpha}{\mathrm{cwnd}} \qquad \textbf{Ecuación 48}$$

Donde, α es una función del retardo en cola estimado (Figura 20).

Cuando se pierde un paquete, el tamaño de la ventana de congestión disminuye de la forma expresada en la siguiente Ecuación 49,

$$\mathrm{cwnd} = (1 - \beta) * \mathrm{cwnd} \qquad \textbf{Ecuación 49}$$

Donde, β es una función del retardo en cola estimado (Figura 20).

TCP Illinois mostró que es capaz de utilizar los recursos disponibles en las redes de alto BDP mejor que Reno. Al mismo tiempo, preserva y mejora las propiedades de equidad. Sin embargo, al depender de la estimación del uso de la red a través de la medición del retardo, adolece de todos los problemas de Vegas y puede retroceder al modo Reno ($\alpha=\alpha_{min}$ y $\beta=\beta_{max}$) por una mala estimación de los valores mínimos o máximos de RTT observados.

YeAH TCP (Yet Another High-speed)

YeAH TCP [69] combina la detección de pérdida de paquetes y la medición de RTT como mecanismos para estimar el estado de la red. El algoritmo propuesto, define los modos lento y rápido. En el modo rápido, la ventana de congestión se incrementa de acuerdo con una regla agresiva. En el modo lento, la ventana de congestión se incrementa como en el TCP “estándar”. El cambio entre los modos lento y rápido se decide de acuerdo con el número estimado de paquetes en cola.

Para proporcionar un mecanismo confiable para la conmutación de modo, YeAH define el uso simultáneo de dos métricas basadas en retardos: la estimación tipo Vegas de una cantidad de paquetes almacenados en la red y el nivel de congestión de la red. Sin embargo, hay dos diferencias en la definición de la última métrica: En primer lugar, en los cálculos de retardo de puesta en cola, YeAH usa el mínimo de los RTT medidos recientemente en lugar de un RTT promediado. En segundo lugar, se mide el nivel de congestión, no como

una fracción del retardo máximo de puesta en cola, sino como una fracción del RTT mínimo observado durante la vida útil de la conexión.

Para la estimación del número de paquetes en cola, se define el RTT_{base} como el mínimo valor de RTT observado durante la conexión y RTT_{min} como el valor minino de RTT observado durante la ventana de congestión en curso, valor que se actualiza una vez por ventana. De esta manera, el retraso actual de la cola (RTT_{queue}) se puede estimar de la siguiente manera (Ecuación 50),

$$\mathrm{RTT_{queue}} = \mathrm{RTT_{min}} - \mathrm{RTT_{base}} \qquad \textbf{Ecuación 50}$$

Este valor, RTT_{queue}, se utiliza para calcular la cantidad de paquetes en cola de la siguiente manera (Ecuación 51),

$$\mathrm{Q} = \mathrm{RTT_{queue}} * (\frac{\mathrm{cwnd}}{\mathrm{RTT_{min}}}) \qquad \textbf{Ecuación 51}$$

Si YeAH estima un bajo nivel de uso de los buffers en la red y la estimación del retardo en la cola muestra un bajo nivel de congestión, entonces se comporta exactamente como S-TCP; de lo contrario, se aplica el modo lento de Reno.

YeAH mantiene una alta eficiencia en redes de alto BDP que mantienen el tamaño de las colas de red a un nivel muy bajo. Además, los enfoques que combinan métricas basadas en la demora pueden mejorar las propiedades de equidad. Sin embargo, el rendimiento de YeAH, en forma similar a todos los enfoques basados en el retraso, puede degradarse cuando las mediciones RTT tienen un ruido significativo.

CAPITULO 3 - Redes Inalámbricas

Las redes inalámbricas son el medio de comunicación de más rápido crecimiento en un mundo moderno. Permiten flexibilidad y movilidad al usuario, sin tener que sacrificar la conexión a Internet o a la red del lugar de trabajo. Con el incremento de la popularidad de las tecnologías inalámbricas, el protocolo TCP ha tenido que extender su capacidad para ser utilizado tanto redes heterogéneas, como en redes completamente inalámbricas [70].

Las sucesivas versiones del protocolo TCP fueron brindando mejoras, a los efectos de lograr que su rendimiento se adaptara a los nuevos desafíos. El control de congestión de la variante TCP Reno fue diseñado en función de las redes cableadas, donde los datos prácticamente llegan correctos y en orden. Sin embargo, se presentan desafíos importantes en los enlaces inalámbricos. Debido a su naturaleza, es frecuente que las pérdidas de paquetes no se deban a la congestión y es habitual el reordenamiento de paquetes. Estos enlaces poseen más pérdidas, dado que las señales que se propagan sufren de atenuación, ruido, interferencia, desvanecimiento, contención, entre otros [71]. De esta manera, los paquetes que se reciben pueden estar dañados y por lo tanto ser descartados, produciendo la pérdida de paquetes en tránsito. El reordenamiento de paquetes se incrementa debido a las altas tasas de errores de transmisión y, en algunos casos, a la movilidad en los nodos. Por lo tanto, el control de congestión, enfrenta nuevos desafíos en el entorno inalámbrico.

La mayoría de las redes inalámbricas implementan protocolos de acceso basados en contención. Los paquetes pueden perderse o corromperse debido a varios problemas basados en contención como colisión o por problemas de estaciones ocultas y expuestas. Por lo que nuevamente, se produce una reducción innecesaria de la tasa de envío.

La disponibilidad del enlace inalámbrico depende de las condiciones del entorno en el que están ubicados y de los objetos que bloquean la propagación de la señal. Esto puede producir pérdidas de paquetes aleatorias (random packet loss), que se manifiesta en la corrupción de algunos pocos bits y produce que el paquete se descarte al ser recibido. Asimismo, pueden generarse desconexiones por la movilidad de los objetos del entorno, o bien, del mismo host. En este caso, se pierden todos los paquetes hasta que el host se reconecta a la red.

Empíricamente se ha constatado que las tasas de error (BER) en los canales radio móviles varían dentro del rango 10^{-6} a 10^{-1}, dependiendo de las condiciones del enlace. Las redes fijas de fibra óptica presentan, en cambio, tasas inferiores a 10^{-12}.

El efecto combinado produce, en general, errores en forma de ráfagas. Esto significa que los canales sufren periodos erráticos durante los cuales toda comunicación a través de ellos resulta inviable. Existe, por tanto, cierta dependencia estadística en la ocurrencia de los errores. Es decir, el comportamiento de los canales de radio está limitado por las frecuentes ráfagas de error. Durante los periodos de ráfaga, el nodo recibe sólo una señal muy débil de manera que todos los intentos de transmisión de datos resultan fallidos con muy alta probabilidad.

3.1. Clasificación

Las comunicaciones inalámbricas pueden clasificarse de distintas formas dependiendo del criterio que se considere. En este caso, se clasifican de acuerdo con su alcance, de menor a mayor área de cobertura, de la siguiente manera:

WBAN (Wireless Body Area Network): Conseguir mayores grados de integración entre los usuarios y los servicios implica la existencia de conexiones entre los dispositivos electrónicos y de comunicaciones que el usuario lleva en su propia ropa o se encuentra en el entorno circundante. Los dispositivos que pueden trabajar con esta tecnología pueden ser equipos complejos como un teléfono celular o un componente simple como un auricular o un visor adaptado a unas gafas. Se comunican normalmente a distancias de 1 o 2 metros. El uso más extendido que se le está dando a esta tecnología es en el ámbito de la salud.

WPAN (Wireless Personal Area Network): Red de cobertura personal, Por ejemplo: Bluetooth (protocolo que sigue la especificación IEEE 802.15.1); ZigBee (IEEE 802.15.4, utilizado en aplicaciones como la domótica, que requieren comunicaciones seguras con tasas bajas de transmisión de datos y maximización de la vida útil de sus baterías, bajo consumo).

WLAN (Wireless Local Área Network): Es un sistema de comunicación muy utilizado como alternativa a las redes LAN cableadas o como extensión de éstas. Utiliza tecnología de radiofrecuencia que permite mayor movilidad a los usuarios al minimizar las conexiones cableadas. Por ejemplo, la familia de estándares 802.11.

WMAN (Wireless Metropolitan Area Network): Tecnologías basadas en WiMAX (Worldwide Interoperability for Microwave Access), estándar que ofrece conectividad desde 1 GHz hasta 60 GHz. Es un estándar de comunicación inalámbrica basado en la norma IEEE 802.16.

WWAN (Wireless Wide Area Network): Tecnologías como UMTS (Universal Mobile Telecommunications System), utilizada con los teléfonos móviles de tercera generación. EDGE, GPRS (General Packet Radio Service) tecnología digital para móviles, ambas para redes 2G y 2.5G, y/o también las otras tecnologías como CDMA y sus versiones más recientes de 4G.

En la siguiente figura (Figura 21) se observa la clasificación de las redes inalámbricas de acuerdo a la distancia.

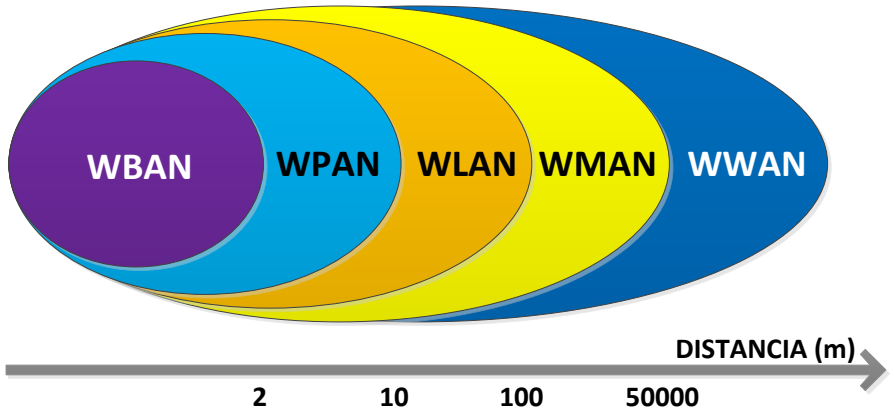

Figura 21: Clasificación de las redes inalámbricas según la distancia

Cada uno de estos escenarios tiene requerimientos diferentes y el diseño del TCP inalámbrico debe considerar las características y las necesidades propias de cada uno. Sin embargo, hay una característica común que en todas las redes inalámbricas se debe enfrentar el alto valor de BER (bit error rate).

3.2. Tipos de Redes inalámbricas

Hay muchos sistemas de comunicación inalámbrica disponibles para múltiples propósitos. Algunos de estos sistemas son: radios móviles, portátiles de dos vías, teléfonos celulares, GPS, dispositivos de apertura de puertas de garaje, mouses inalámbricos, teclados y auriculares, televisión satelital y teléfonos inalámbricos. De esta forma, muchos usuarios domésticos, empresas y campus utilizan redes inalámbricas de área local en lugar de redes cableadas. Esto da lugar a que los campos de aplicación de los enlaces inalámbricos sean múltiples y muy variados [72].

A partir del crecimiento del uso de la tecnología inalámbrica, nuevas formas de utilizar estos enlaces han sido objetos de atención y estudio en los grupos de investigación en los últimos años, donde es particularmente deseable mejorar el rendimiento de TCP. Entre ellas se destacan las Movil AdHoc Networks (MANET), redes Mesh (WMN), redes de sensores inalámbricos (WSN), redes celulares, y las redes inalámbricas de área local (WLAN), entre otras [73].

3.2.1. MANET (Mobile Ad Hoc Networks)

Las MANET [74], consisten en un número de dispositivos móviles que se unen para formar una red, sin el soporte de ninguna infraestructura existente o cualquier otro tipo de estación fija. Formalmente, un MANET puede definirse como un sistema autónomo de nodos, que también cumplen funciones de router, conectados por enlaces inalámbricos, cuya unión forma una red de comunicación. Estos nodos de MANET se configuran automáticamente y no requieren una infraestructura central para comunicarse entre sí. Los nodos no permanecen estacionarios en el entorno, por lo que la topología de la red puede cambiar dinámicamente de una manera impredecible. Esto se debe a que los nodos pueden moverse libremente y cada nodo tiene una potencia de transmisión limitada, que restringe el acceso al nodo sólo en el rango vecino. De esta manera, los paquetes de información se transmiten de manera remota desde un origen a un destino, a través de nodos intermedios. A medida que los nodos se mueven, la conectividad puede cambiar según las ubicaciones relativas de otros nodos. El rendimiento de la red puede degradarse ya que la ubicación de los nodos móviles debe aprenderse dinámicamente. Es habitual que los paquetes realicen varios saltos a través de distintos nodos hasta llegar a su destino, por la baja potencia que posibilita el consumo de energía que pueden tener cada nodo. Debido a que el alcance de transmisión de los dispositivos es limitado, pueden llegar a ser necesarios nodos intermedios para transferir datos de un nodo a otro. Por ello, en una red MANET, cada nodo puede operar como fuente, destino o router (naturaleza “multihop”). En estos casos, TCP enfrenta desafíos en virtud de la naturaleza de los entornos inalámbricos, como desconexiones o altas tasas de error, y se agrega, además, que la topología de la red cambiaría con frecuencia de acuerdo a la movilidad de los nodos.

En la siguiente figura (Figura 22) se observa una red ad hoc móvil, donde el nodo M3 se desplaza desde una ubicación hacia otra, modificando la topología de la red. En el punto de inicio, los paquetes para M3 debían ser ruteados a través de M1. Sin embargo, en la nueva ubicación debe enviarse a través de M7.

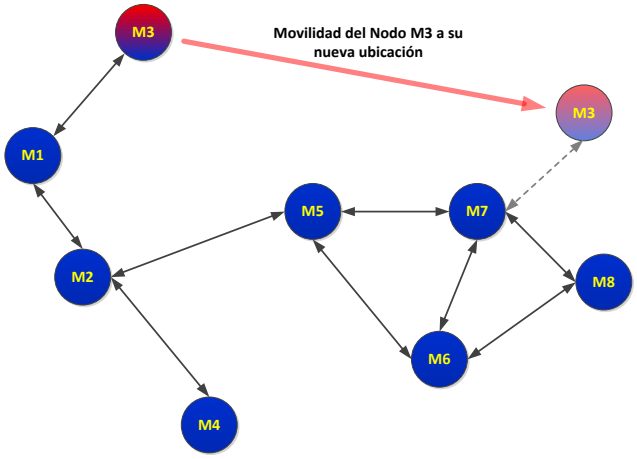

Figura 22: Cambio de topología por movimiento de un nodo en redes MANET

Un ejemplo de estas redes son las redes VANET (Vehicular Ad-Hoc Network). Las VANET son un subconjunto de las redes móviles ad-hoc capaces de comunicar información entre diversos vehículos y el sistema de tránsito. En principio, este tipo de red es una subclase de MANET. Sin embargo, se comporta de maneras fundamentalmente diferentes a consecuencia de la frágil conectividad de los vehículos, la alta velocidad del movimiento, y la alta movilidad. En la siguiente figura (Figura 23) [75] se puede observar un ejemplo de aplicación urbana.

Figura 23: Red Ad Hoc Vehicular (VANET)

La idea es proporcionar conectividad entre los diferentes usuarios móviles con una comunicación eficiente de vehículo a vehículo. Esta conectividad puede ser entre vehículos que se comunican con vehículos cercanos y con unidades laterales del medio a través de enlaces inalámbricos dinámicos. Las aplicaciones de las VANET incluyen el monitoreo de tráfico, control de flujos de tráfico, prevención de colisiones, servicios de información y cálculo de rutas de

desvío en tiempo real. Debido a la variedad de aplicaciones críticas para la seguridad, estas redes están generando una especial atención entre los investigadores e ingenieros de las industrias, académicas y automotrices para aplicaciones de seguridad vial tales como servicios de emergencia como accidentes, embotellamientos, desvíos de tráfico, seguridad pública, condiciones de salud, etc.

3.2.2. WMN (Wireless Mesh Networks)

En los últimos años, las WMN [76] han sido utilizadas para el suministro de servicios de internet inalámbrico de banda ancha para los usuarios. Dentro de las ventajas de las WMN, se incluyen su capacidad de auto organización y de recuperación automática, la infraestructura de bajo costo, la rápida implementación, la escalabilidad y la facilidad de instalación.

Los nodos de una red WMN deben auto organizarse y auto configurarse dinámicamente, establecer una red “Ad Hoc” y mantener la conectividad de la malla. Estos nodos pueden ser routers (Mesh Routers) o clientes (Mesh Clients). La siguiente figura (Figura 24) representa un ejemplo de una WMN, donde se pueden observar puertas de enlace de Internet (IGW), enrutadores de malla (MR) y clientes de malla (MC).

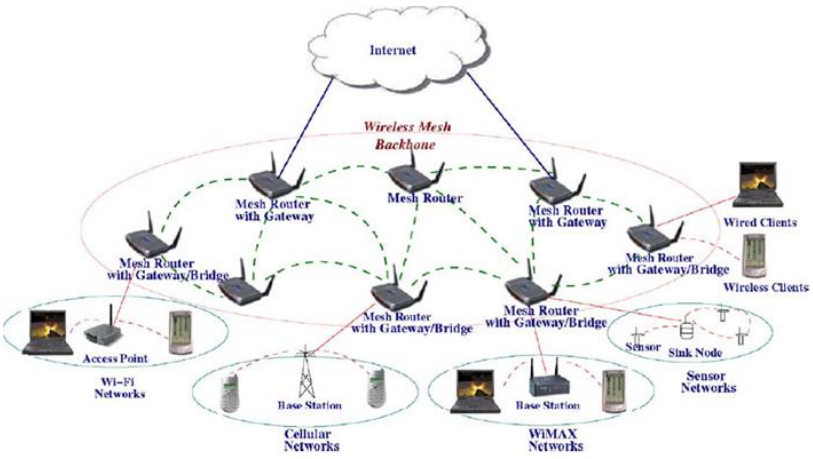

Figura 24: Wireless Mesh Networks [76]

Además de la capacidad de Gateway/Bridge de un Router inalámbrico convencional, un Mesh Router debe implementar funciones adicionales. Para lograr la flexibilidad de la red, estos routers generalmente poseen múltiples

interfaces inalámbricas que pueden ser de la misma o de diferentes tecnologías de acceso. A pesar de estas diferencias, estos routers generalmente están implementados sobre un hardware similar y forman parte del “Backbone” de la red Mesh para los clientes.

Las WMN generalmente operan en las bandas sin licencia, lo que implica que la comunicación este plagada de interferencias y colisiones debido a terminales ocultos o expuestos. Esto da como resultado un rendimiento de extremo a extremo excesivamente bajo. En particular para WMN, las fluctuaciones en la calidad de los enlaces, la carga excesiva en algunos de ellos, la congestión y la capacidad limitada debido a la naturaleza semidúplex de los canales de radios son algunos factores limitantes clave que dificultan su implementación. Además, los paquetes que utilizan múltiples saltos experimentan un rendimiento menor, en comparación con los paquetes que atraviesan menos saltos, lo que lleva a una inequidad espacial. La calidad fluctuante de los enlaces inalámbricos obliga a incluir en el diseño de enrutamiento la estabilidad del enlace en las métricas para la toma decisiones. Asimismo, la implementación del protocolo de control de transmisión (TCP) “estándar” no tiene un buen rendimiento en las redes inalámbricas multihop, en vista de que las características inherentes de TCP pueden dar como resultado retrasos excesivos de paquetes, reordenamiento de los mismos, etc.

3.2.3. WSN (Wireless Sensor Networks)

Las redes de sensores inalámbricos [77] son una clase de redes ad hoc sin infraestructura física preestablecida ni administración central, que permiten un monitoreo y análisis confiable de distintos entornos. Una red de sensores inalámbricos es una colección de pequeños dispositivos de bajo consumo que convierte un atributo físico en datos. Cualquiera de estos dispositivos puede incluir un módulo de detección, un módulo de memoria y, por lo general, una fuente de energía agotable como una pequeña batería.

Las WSN están constituidas por sensores autónomos distribuidos espacialmente con el propósito de ser capaces de comunicarse entre sí con un mínimo de consumo de energía, que entrega una su colección de datos. Para este tipo de redes, el estándar más utilizado es IEEE802.15.4. Este estándar

determina las comunicaciones en las capas físicas y de control de acceso al medio MAC, en las redes de sensores inalámbricos de área personal de baja velocidad (LR-WPAN). Las LR-WPAN son caracterizadas por la baja tasa de datos que transportan, el bajo consumo de energía requerido para su funcionamiento, la variabilidad de la topología de red y el conocimiento de la ubicación. Además de la degradación del rendimiento asociada con la naturaleza poco confiable del canal inalámbrico, mayores tasas de error de bits y pérdida de paquetes, comunes a este tipo de enlaces, las redes de sensores se caracterizan por múltiples saltos inalámbricos, que degrada el rendimiento de TCP. La tecnología inalámbrica es la vía por la que se comunican entre ellos y hacia Internet. En este sentido, cobran importancia las redes de sensores inalámbricos (WSN) como la tecnología que permite la escalabilidad del IoT (internet of Things), con la funcionalidad suficiente para proporcionar su integración con la arquitectura actual de Internet. La tecnología clave para la creación de redes IP en los dispositivos inalámbricos es 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks), un estándar que especifica cómo se transportan paquetes IPv6 sobre IEEE 802.15.4.

En la siguiente figura (Figura 25) se observa la topología de un sistema WSN. Por un lado, numerosos dispositivos distribuidos espacialmente, que utilizan sensores para controlar diversas condiciones en distintos puntos, tales como la temperatura, el sonido, la vibración, la presión y movimiento o los contaminantes.

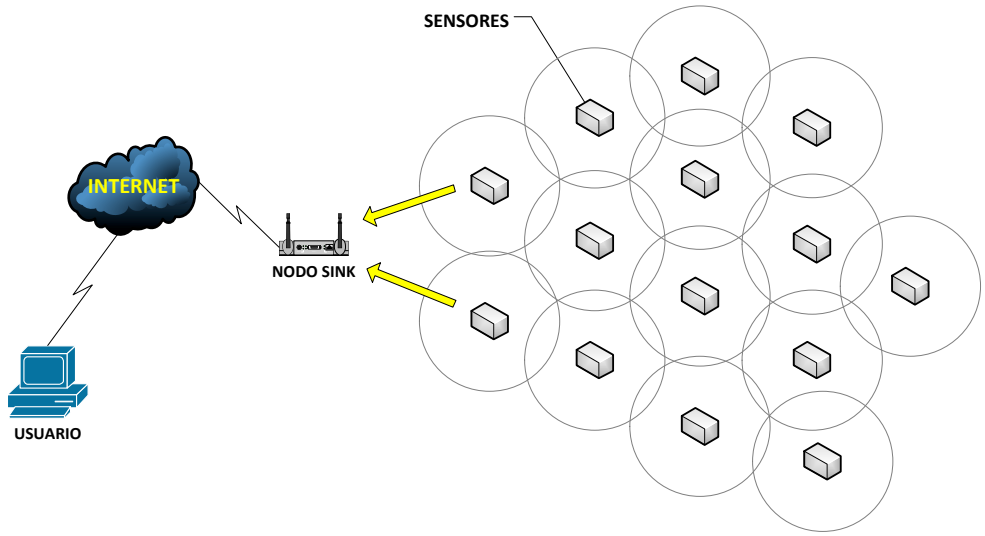

Figura 25: Wireless Sensor Networks

El nodo sumidero (sink) se convierte en la raíz de la topología y se encarga de la recolección de los datos, actuando “Gateway” hacia las redes externas, incluso internet.

Las limitaciones inherentes de los medios inalámbricos, como el ancho de banda bajo, las transmisiones propensas a errores y la necesidad de acceso a canales sin colisiones también están presentes en las redes de sensores. Además, dado que los nodos inalámbricos no están conectados de ninguna manera a una fuente de alimentación constante, obtienen energía de las baterías. Esto limita la cantidad de energía disponible para los nodos, y como se implementan en lugares donde es difícil reemplazar los nodos o sus baterías, es deseable aumentar la vida útil de la red. Por lo tanto, los protocolos diseñados para estas redes deben distribuir estratégicamente el uso de energía, lo que aumenta la vida media del sistema en general.

3.2.4. Satélites

La comunicación a través de satélites proporciona conexiones globales en cualquier momento [11]. Esos desempeñan roles importantes durante desastres naturales, ya que pueden llegar a áreas remotas e inaccesibles.

Un sistema de comunicación por satélite se puede definir como un sistema de comunicaciones autónomo. Los satélites tienen la capacidad de recibir señales de la Tierra y retransmitir esas señales con el uso de un transpondedor. Un transpondedor es un receptor y transmisor integrado de señales de radio.

Como se observa en la siguiente figura (Figura 26), existen diferentes tipos de satélites con distinto tipo de orbitas, tales como los GEO (Geostationary Earth Orbit) aproximadamente 36,000 km sobre la superficie de la tierra, LEO (Low Earth Orbit) entre 500–1500 km sobre la superficie, MEO (Medium Earth Orbit) o ICO (Intermediate Circular Orbit) alrededor de 6000–20,000 km de altura y HEO (Highly Elliptical Orbit) de órbita elíptica.

Las Redes Satelitales se caracterizan por el largo retardo de propagación del enlace satelital, lo que genera una alta latencia en el enlace. También es característica la presencia de una la alta tasa de error en tránsito que, sumado

a un valor elevado del producto de latencia por ancho de banda (BDP), tiene un gran impacto en el rendimiento de la red de satélites.

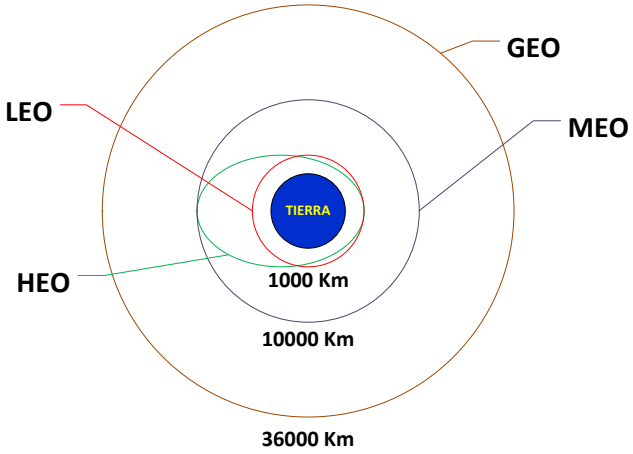

Figura 26: Tipos de orbitas

El protocolo TCP juega un papel importante en la transmisión de datos en la comunicación por satélite, pero su rendimiento es muy bajo debido a las características antes mencionadas, el retardo de extremo a extremo, las variaciones de retardo, la asimetría de ancho de banda y una mayor tasa de errores de bits, entre otras.

Es preciso señalar, que actualmente es una tecnología no presenta interés en desarrollo en los temas de la tesis. Sin embargo, resulta cita obligada por el rol desempeñado en los orígenes de las redes de datos.

3.2.5. Redes Celulares

El Sistema Global para Comunicaciones Móviles (GSM), el estándar celular, se describió originalmente como una red de conmutación de circuitos optimizada para telefonía vocal de full dúplex. Fue la tecnología de acceso inalámbrico más popular definida en 1990 por ETSI para las redes celulares digitales de segunda generación (2G) y reemplazaba los sistemas analógicos de primera generación (1G) como NMT (Nordic Mobile Telephony) y TACS (Total Access Communication System). IS-95 (Interim Standard 95) es otro estándar de tecnología inalámbrica móvil 2G que utiliza CDMA (Code Division Multiple Access), un esquema de acceso múltiple para que la radio digital envíe datos

de voz, datos y señalización entre usuarios móviles y estaciones de transmisión celular [73].

El estándar GSM se mejoró para incluir el transporte de paquetes de datos a través de GPRS (General Packet Radio Service). Las velocidades de transmisión de paquetes de datos se incrementaron a través de la introducción de EDGE (Enhanced Data rates for GSM Evolution) en el año 2003, que permite velocidades de datos de 59.2 kb/s. La norma GSM fue seguida por la tercera generación de Universal Mobile Telecommunications System (UMTS), el estándar fue desarrollado por Third Generation Partnership Project (3GPP). Las redes GSM evolucionaron aún más a medida que comenzaron a incorporarse los estándares LTE-Advanced (Long Term Evolution) de cuarta generación (4G).

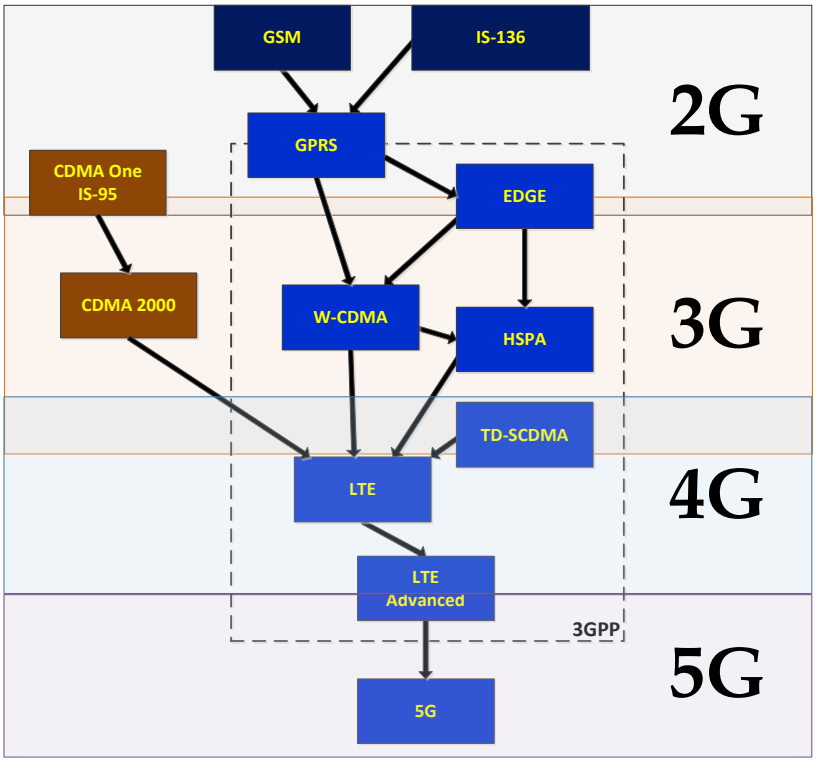

Figura 27: Evolución del estándar de redes celulares

En la figura anterior (Figura 27) se observa la evolución del estándar móvil de 2G a 5G.

UMTS (Universal Mobile Telecommunications System) especifica un sistema de red completo, que cubre la red de acceso de radio UTRAN (Terrestrial Radio Access Network), core network (Mobile Application Part), y la autenticación de

los usuarios móviles a través de las SIM cards. Existen varios tipos de canales: canales físicos, canales de transporte (canales de transporte comunes y canales de transporte dedicados) y canales lógicos.

LTE (Long Term Evolution) es un estándar adecuado para el acceso de datos de última milla en la tecnología de red celular con un conjunto de mejoras al UMTS que se introdujo en 3GPP.

Desde el punto de vista de TCP, las redes celulares están conformadas por estaciones base que interconectan una red fija de alta velocidad y una red móvil lenta. De esta manera, TCP debe contemplar las características de estas redes y el problema compartido de todas las redes inalámbricas, el valor de BER alto. Es decir, debe abordar la degradación del rendimiento causada por desconexiones frecuentes debido a la transferencia de dispositivos móviles entre estaciones bases, o el bloqueo temporal de las señales de radio por obstáculos.

3.2.6. WLAN (Wireless Local Area Networks)

Las redes inalámbricas de área local (WLAN) se basan, principalmente en la familia de estándares IEEE 802.11. El estándar IEEE 802.11, también comúnmente denominado como Wi-Fi (Wireless Fidelity), ha recorrido un largo camino desde el lanzamiento de su primera versión en 1997. La primera versión se introdujo como una alternativa o extensión a las LAN cableadas existentes que se basaban en la tecnología Ethernet y el estándar IEEE 802.3. Desde su aparición, el estándar IEEE 802.11 ha sufrido modificaciones continuas para adaptarse a las nuevas funcionalidades y tecnologías y también para satisfacer las necesidades cambiantes del mundo digital en constante expansión. Cada nuevo estándar fue introduciendo modificaciones, pero siempre debieron mantenerse compatibles con las versiones más antiguas.

El uso generalizado de las redes WLAN IEEE 802.11 se debió principalmente al bajo costo de instalación, a la sencilla implementación, la accesibilidad y la flexibilidad. Al mismo tiempo, la mayoría de los dispositivos móviles y portátiles están equipados con interfaz de tecnología Wi-Fi. Claramente, esto resulta en un aumento continuo del tráfico que fluye a través de las redes y en particular

de las WLAN. Esto, a su vez, ha incrementado y continuará aumentando la demanda de contenido multimedia en dispositivos móviles, lo que se traduce en la necesidad constante de que las WLAN evolucionen y proporcionen soluciones nuevas y eficaces para enfrentar los nuevos desafíos.

Los usuarios suelen conectar sus dispositivos móviles mediante Redes Celulares y las WLAN. Estas redes se denominan redes de acceso inalámbrico. Las redes de datos celulares difieren de las redes WiFi en que proporcionan una cobertura de señal más amplia y una conectividad más confiable bajo movilidad, implementando en sus sistemas mecanismos de retransmisión locales, transparentes para TCP, que mitigan las retransmisiones de TCP y reducen el desperdicio de recursos valiosos en redes celulares. Si bien estos mecanismos reducen drásticamente el impacto de las pérdidas y mejoran el rendimiento del TCP, tienen un costo en mayor latencia y la variabilidad del ancho de banda.

3.3. Redes Inalámbricas de Área Local (WLAN)

Las redes inalámbricas Wi-Fi, que utilizan la norma IEEE 802.11, han tenido un gran crecimiento en los últimos años, difundiéndose en una gran cantidad de dispositivos electrónicos (laptops, celulares, tablets, etc.) que implementan este estándar para acceder a Internet desde una red local [78].

IEEE 802.11 es el estándar de transmisión de datos inalámbricos dominante. Estas especificaciones son creadas y mantenidas por el grupo de trabajo IEEE 802.11. La versión base del estándar fue lanzado en 1997, y ha tenido modificaciones posteriores [79].

El comité del estándar IEEE 802 define dos capas separadas para la capa de enlace de datos del modelo de referencia OSI, la subcapa de control de enlace lógico LLC (Logical Link Control) y la subcapa de control de acceso al medio MAC (Media Access Control). El estándar IEEE 802.11 incluye la especificación de la capa física del modelo de referencia OSI (Open System Interconnection) y de la subcapa de Control de Acceso al Medio (MAC, Medium Access Control).

El nivel físico realiza la transmisión de bits a través del medio inalámbrico, encargándose de la modulación y codificación, sintonización de canales, etc.

La capa física se divide en dos subcapas, la Physical Medium Dependent (PMD), que maneja las distintas bandas de frecuencia de trabajo y los esquemas de modulación disponibles y la subcapa Physical Layer Convergence Protocol (PLCP), la cual es responsable de interactuar entre la capa MAC y la subcapa PMD. El objetivo básico es hacer a la capa MAC lo más independiente posible de las diferentes especificaciones físicas. Es importante notar que la subcapa PLCP anexa un encabezado y un preámbulo a cada paquete, modulado a una tasa específica.

La capa MAC, parte inferior del nivel de enlace, es la que gestiona el direccionamiento y el acceso múltiple a un medio compartido. MAC también incluye las funcionalidades de fragmentación, encriptación, sincronización, gestión de potencia y tránsito entre redes.

El estándar y las subsiguientes enmiendas constituyen la base de los productos para redes inalámbricas que utilizan la marca Wi-Fi. De forma complementaria a los a los estándares IEEE 802.11, surge la necesidad técnica y comercial de asegurar la interoperabilidad entre los diferentes productos de diversos fabricantes y vendedores Para este propósito se creó la Wi-Fi Alliance [80]. Una de sus tareas consiste en certificar la compatibilidad de los distintos productos 802.11. El término “WiFi” se utiliza para distinguir los productos 802.11 certificados por la organización.

El estándar IEEE 802.11 define las especificaciones para la capa física (PHY) y la capa de control de acceso al medio (MAC) que se comunica por arriba con la capa de control de enlace lógico, tal y como se muestra en la Figura 28.

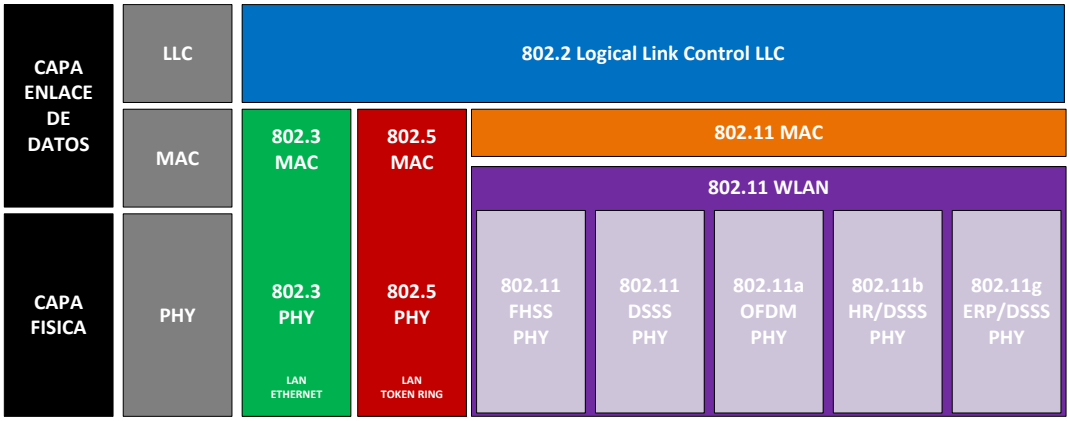

Figura 28: Capa de enlace de datos y capa física del modelo OSI

3.4. TCP en Redes Inalámbricas

TCP encuentra desafíos importantes en el entorno inalámbrico debido a la naturaleza más impredecible de los enlaces, que no fueron parte de su diseño original [81].

El éxito de este protocolo se basa en su capacidad de adaptarse a los cambios en la red y a su fiabilidad extremo a extremo, lo que resultó en su amplia utilización. Al no ser diseñado para uso a las redes inalámbricas, se le presentan desafíos críticos, derivados de un ambiente complejo con tasas de error de bits (BER) nada despreciables debido al medio físico y a la movilidad de los usuarios. Como resultado, los enlaces inalámbricos exhiben una tasa de error mayor, menor velocidad que los enlaces por cable, mayor latencia, mayor Jitter y desconexiones temporales.

Las señales que se propagan a través de canales inalámbricos sufren la degradación y los paquetes pueden perderse en tránsito o bien ser recibidos con información que fue alterada y se descartan, lo que lleva a la aparición de pérdidas de paquetes que no se deben a la congestión.

Debido a lo anteriormente expuesto, se produce un comportamiento poco eficiente en este entorno, debido a la incapacidad de TCP de discriminar entre los distintos tipos de pérdidas, la de los enlaces inalámbricos, y las perdidas por congestión en la red. Sin embargo, en ambos casos, el emisor reduce su tasa de envío activando el control de congestión aun cuando no haya una congestión incipiente, lo que degrada el rendimiento de TCP en este medio.

3.4.1. Características de las redes inalámbricas

Los enlaces inalámbricos están sujetos a muchos factores, como las condiciones climáticas, los obstáculos urbanos, las interferencias de trayectos múltiples, los objetos en movimiento y la movilidad de los dispositivos [82]. Como resultado, poseen valores de BER mucho más altos y las limitaciones de la cobertura de radio resulta en desconexiones temporales y reconexiones durante una sesión de comunicación. Distintos aspectos inherentes a las redes inalámbricos pueden afectar el rendimiento de TCP. Dentro de sus

características propias, se destacan a continuación las que tienen efecto directo sobre el rendimiento de TCP.

Alta tasa de error de bits (BER): Los nodos utilizan el canal inalámbrico como medio para enviar y recibir datos. Estos son propensos a tener valores de BER alto, típicamente más alto que en las redes cableadas y, también varía fuertemente con los cambios en el entorno. Dado que las redes 802.11 operan dentro de bandas de frecuencia sin licencia, las señales de otros dispositivos, o el ruido electromagnético puede interferir con las señales. De esta forma, el canal inalámbrico es vulnerable a errores debido a la atenuación de la señal, la interferencia, los obstáculos y el desvanecimiento por caminos múltiples. Además, a medida que un dispositivo inalámbrico se aleja de la estación base, la intensidad de la señal disminuye y provoca la pérdida de datos. De esta manera, los enlaces inalámbricos son más propensos a los errores en la transmisión. Esta naturaleza poco confiable del medio inalámbrico causa un número sustancial de pérdidas de paquetes.

Se define como perdida aleatoria (Random Loss) cuando se pierde un solo paquete en un RTT. Sin embargo, la pérdida de varios paquetes consecutivos es habitual en estos enlaces, lo que se conoce como pérdidas en ráfaga (Burst Loss).

Las pérdidas de paquetes pueden tener distintas causas:

- Los obstáculos reducen el nivel de la señal captada por el receptor. Esta atenuación, inherente al canal radio, es conocida como desvanecimiento lento o shadow fading. La reducción en el nivel de la señal por causa de este desvanecimiento es transitoria si el obstáculo en cuestión, el emisor o bien el receptor, están en movimiento. La duración de la pérdida de señal puede durar unos pocos milisegundos o, incluso, prolongarse varios segundos dependiendo de factores como la naturaleza del obstáculo o la velocidad relativa de desplazamiento.
- La señal radio también puede verse sometida a reflexiones lo que produce que la señal puede dispersarse y viajar en diferentes caminos debido a la reflexión, difracción y dispersión causada por obstáculos antes de que llegue al receptor. Este fenómeno, conocido como

propagación multicamino o multipath, puede ocasionar la llegada al receptor de señales con diferencias de fase tales que, su suma dé lugar a una interferencia destructiva cuyas consecuencias pueden llegar hasta la cancelación mutua, es decir, la eliminación total de la potencia de señal en recepción. La propagación multicamino da lugar a desvanecimientos rápidos o de Rayleigh.

- Las interferencias procedentes de transmisiones concurrentes pueden degradar la relación señal/ruido en el receptor.
- La limitación de la potencia transmitida por los terminales móviles, debida tanto a la necesidad de minimizar las interferencias como a consideraciones relacionadas con su alimentación y uso de energía, contribuye, también, a la presencia de altas tasas de error en los enlaces radio.
- El ruido ambiente es inevitable en un entorno abierto y frágil como el canal radio. En núcleos urbanos, este factor es especialmente influyente.

Contención: La Contención en el uso de un canal inalámbrico compartido limita la capacidad de un nodo para enviar paquetes. La capa MAC de IEEE 802.11, utiliza un esquema de control de acceso al medio basados en la contención, donde los nodos compiten por la utilización del canal compartido antes de transmitir. Esto puede producir que las señales que se transmiten interfirieran entre sí. De esta manera, se detectará una colisión y estas pueden fallar cuando existan transmisiones concurrentes dentro del rango de interferencia. El rango de interferencia de un dispositivo se define como la región en la cual una transmisión iniciada por el dispositivo interpondrá o corromperá otras transmisiones en curso. Hay dos casos particulares de comportamiento que se pueden destacar, el problema del nodo oculto y el problema del nodo expuesto. El nodo oculto es el que está dentro del rango de interferencia del receptor, pero fuera del rango de detección del transmisor, por lo tanto, no puede detectar su transmisión de datos. El receptor puede no recibir correctamente el paquete debido a la colisión de la transmisión del nodo oculto, produciendo errores o pérdidas de paquetes. Un nodo expuesto es el que está dentro del rango de detección del emisor, pero fuera del rango de interferencia del

receptor. Aunque su transmisión no interfiere con el receptor, el emisor no puede iniciar la transmisión porque detecta un medio ocupado.

Movilidad: Una de las principales ventajas de usar dispositivos inalámbricos es movilidad. La movilidad de los terminales brinda claros beneficios, pero un nodo que se mueve del rango de una estación base a otra, o cuando la celda tiene un gran número de usuarios y el ancho de banda no es suficiente para satisfacer sus necesidades, puede experimentar un breve período de desconexión.

En una red del tipo infraestructura cuando el nodo móvil cambia de celda, se lleva a cabo el proceso de traspaso llamado handoff, en el cual transfiere el control de la comunicación desde la estación base inicial a una nueva estación base con cobertura en la nueva celda. Durante el intervalo de tiempo que dura este proceso, los paquetes destinados al móvil y aquellos transmitidos por este no pueden ser entregados. Esto provoca incremento en la latencia y pérdida de paquetes.

En algunas ocasiones, el usuario móvil podría moverse temporalmente fuera del alcance de su estación base y entrar en zonas de sombra. En estos casos, se producen desconexiones temporales que producen pérdidas de paquetes. Un evento de desconexión breve en realidad puede detener la transmisión TCP durante un período mucho más largo.

Energía limitada: El nodo móvil es generalmente un dispositivo pequeño, con una potencia y capacidad de procesamiento limitadas, lo que permite que un dispositivo se mueva libremente, aunque requiere un uso prudente de la energía disponible. En general son alimentados por batería lo que significa que la energía disponible es limitada. Para conservar energía, un dispositivo móvil debe ser capaz de transmitir y recibir de manera inteligente para minimizar el número de transmisiones y recepciones para ciertas operaciones de comunicación. Por supuesto, TCP no se diseñó teniendo en cuenta el consumo de energía.

Ancho de banda limitado y variable: Los enlaces inalámbricos poseen, en general un ancho de banda menor a una red cableada, por lo que su uso óptimo es un problema importante en las redes heterogéneas. En algunos

casos puede ocurrir que el ancho de banda disponible sea variable en el tiempo. Por ejemplo, el movimiento de los nodos desde el rango de una estación base a otra, hace que el número de nodos móviles asociados con una estación base sea variable en un momento dado. Dado que todos los nodos en un rango dado, comparten el ancho de banda total, el ancho de banda disponible por nodo también puede variar en el tiempo.

BDP (bandwidth delay product): En algunos casos, los enlaces inalámbricos poseen una mayor latencia que los enlaces cableados. Esto afecta el throughput de TCP y los retardos percibidos por los usuarios. En algunos enlaces inalámbricos, tales como redes 2.5G / 3G o redes satelitales, el producto de retraso de ancho de banda (BDP) tiende a ser grande, lo que demora el tiempo necesario para la utilización eficiente de la capacidad del canal, y la recuperación después de un evento de perdida.

3.4.2. Efectos sobre el rendimiento de TCP

El diseño del protocolo TCP en redes inalámbricas requiere tener en consideración distintos aspectos de la comunicación que los considerados en las redes cableadas y, de esta manera, TCP exhibe debilidades en estos entornos heterogéneos. Los problemas se derivan de las características únicas de las redes inalámbricas, y las consideraciones de diseño del protocolo y lo que produce que se degrade significativamente su rendimiento [83]. Esto se manifiesta en diversas formas como la ineficiencia en la utilización de los recursos de la red y la interrupción innecesarias de las transmisiones de datos.

Un TCP como Reno no puede manejar los valores altos de BER y las desconexiones frecuentes de manera eficiente. Como deduce que todas las pérdidas de paquetes son el resultado de la congestión de la red, la pérdida aleatoria de paquetes del enlace inalámbrico produce, en forma innecesaria, que el emisor TCP reduzca su velocidad de envío [84].

Análogamente las variantes de TCP basadas en retroalimentaciones de ACK, pueden resultar afectadas por las malas condiciones del camino inverso, y reducir innecesariamente la velocidad de envío y, como resultado, una degradación del rendimiento de TCP. Una gran parte de los estudios ignoran el

hecho de que los paquetes ACK también pueden experimentar congestión o errores de transmisión produciendo la pérdida de estos paquetes [85].

Otro de los efectos es el de compresión ACK. La puesta en cola en la ruta inversa de un flujo TCP puede causar la llegada casi instantánea de ACK sucesivos al emisor, lo que puede causar grandes ráfagas de transmisión de paquetes y, por lo tanto, posibles pérdidas de paquetes por congestión. En cualquiera de los casos, para algunas variantes de TCP, que controlan la velocidad de envío en función de la tasa de llegadas de los ACK, el emisor deduciría un estado incorrecto sobre el estado de la red [86].

Efectos debidos a la alta tasa de error de bits (BER)

Los paquetes perdidos debido al alto valor del BER, es una condición transitoria, a diferencia de la congestión. TCP toma medidas de control de congestión cada vez que se produce una pérdida en el enlace inalámbrico y reduce su ventana de congestión, reduciendo así la tasa de envío. El problema en las variantes de TCP de tipo reactivo, es que un paquete perdido desencadena mecanismos de control de congestión, que esencialmente hacen que la ventana de congestión del emisor se reduzca. Con la interpretación errónea de la naturaleza de la pérdida, los mecanismos de control de congestión reaccionan de manera inapropiada manteniendo baja la velocidad de envío de una conexión TCP, lo que lleva a un bajo rendimiento.

Los algoritmos de Fast Retransmit y Fast Recovery, introducidos por TCP Reno permiten recuperarse de pérdidas de paquetes aleatorios esporádicos con bastante rapidez, si tales pérdidas solo ocurren una vez dentro de un RTT. Sin embargo, los ruidos, la movilidad y otros factores en el entorno inalámbrico pueden causar errores de bits aleatorios en ráfagas cortas, lo que conduce a una mayor probabilidad de pérdidas aleatorias de múltiples de paquetes dentro de un RTT. Las retransmisiones también tienen un importante efecto en el valor del RTO al implementarse el backoff exponencial para establecer su valor. De esta manera, múltiples retransmisiones fallidas dentro de un RTT causarían que el temporizador de retransmisión retroceda exponencialmente y TCP reduzca a la mitad continuamente la ventana de congestión. El aumento del valor de RTO también aumentará el tiempo de inactividad en la red, lo que reducirá el rendimiento de extremo a extremo.

En cualquiera de los casos, puede suceder que ocurran timeouts en serie y múltiples retransmisiones del mismo paquete debidos a una sola ráfaga de errores. Esto puede producir la expiración de los temporizadores de retransmisión en serie para una conexión, de modo que se produzcan múltiples timeouts consecutivos y múltiples retransmisiones dentro de ese evento. El valor del temporizador de retransmisión (RTO) se duplica con cada intento fallido (backoff exponencial). Varias fallas consecutivas de retransmisión pueden resultar en un período de inactividad de la conexión importante, incluso después de que las condiciones de la red se hayan restablecido a la normalidad. El fenómeno que se denomina Serial Timeouts y, de esta manera, cuando el host móvil se vuelve a conectar, al TCP emisor puede estar esperando un timeout, sin que haya transferencia de datos [87].

Efectos debidos a la movilidad

Cuando el nodo móvil se desconecta, se pierden paquetes consecutivos de datos y de ACK. El TCP emisor debe esperar hasta el primer timeout para retrasmitir el paquete, pues no recibió ningún ACK duplicado y como consecuencia no ejecuta el algoritmo de Fast Retransmit. El emisor ignora estos motivos de las pérdidas e invoca los algoritmos de control de congestión erróneamente, reduciendo así el rendimiento efectivo tras la reconexión. En consecuencia, en el caso de TCP Reno, se recupera reduciendo el umbral ssthresh a la mitad del tamaño, el valor de la ventana de congestión se establece nuevamente en 1 e invoca al algoritmo de Slow Start, causando un impacto muy negativo en el rendimiento de TCP.

Durante la desconexión, se descartan tanto los paquetes de datos como los ACK, y cada intento de retransmisión conduce a una retransmisión fallida. Estos intentos consecutivos fallidos de retransmisión harán que en el emisor TCP retroceda exponencialmente los temporizadores de retransmisión, de acuerdo al algoritmo de Karn, generando tiempos de espera cada vez más largos. Esto produce que el emisor esté en estado de espera aun cuando el host móvil se haya reconectado. El canal permanecerá inactivo desperdiciando recursos de red y disminuyendo el rendimiento efectivo.

En los casos más severos, el host móvil se desconectó y se volvió a conectar y, sin embargo, el emisor podría seguir en tiempo de espera. Análogamente, una

vez que deja de estar en espera porque recibe el ACK del host móvil, requiere de un tiempo importante para llegar a los niveles de velocidad de trasmisión anteriores a la desconexión del móvil. Es un hecho que se puede observar que el TCP emisor no recupera el tamaño de la ventana de congestión debido a la continua reducción a la mitad del umbral (ssthresh) en cada desconexión.

Efectos debidos al reordenamiento de paquetes (packet reordering)

El reordenamiento de paquetes se refiere al comportamiento de la red donde, el orden de recepción de un flujo de paquetes, difiere de su orden de envío. El reordenamiento de paquetes no es un evento poco frecuente. TCP asume, en la mayoría de los casos, que la subred no desordena paquetes o que este reordenamiento es muy leve. Esto puede producir retransmisiones innecesarias lo que puede provocar una degradación sustancial en el rendimiento. Además de los problemas de movilidad, la retransmisión de la capa de enlace es otra causa para el reordenamiento de paquetes. Como efecto secundario de la retransmisión local, se altera el orden de los paquetes de un flujo.

El reordenamiento de paquetes puede producir ACK duplicados que algunas variantes de TCP utilizan como indicio de una perdida de paquete, retransmitiéndolo e invocando a los algoritmos de control de congestión. Esta acción no solo reduce innecesariamente su tasa de envío, sino que además retransmite innecesariamente un paquete.

Efectos debido al ancho de banda limitado y variable

La tasa de aumento en la ventana de congestión de TCP depende de la tasa de llegada de los ACK. La velocidad de llegada de los ACK, a su vez, depende del ancho de banda en las rutas directa e inversa. Los enlaces inalámbricos tienen menor capacidad de canal en comparación con sus contrapartes cableadas. Esto hace que el retraso de extremo a extremo sea mayor, es decir, el valor de RTT es grande. Las conexiones que tienen RTT más cortos dan lugar a un crecimiento más rápido de la ventana de congestión. Las conexiones con enlaces inalámbricos están en desventaja por este aspecto. Las conexiones típicas duran muy poco tiempo y la mayor parte de la comunicación se completa en la fase inicial de Slow Start. De esta forma, el rendimiento de TCP se ve afectado considerablemente. Como consecuencia del aumento de

RTT, el valor de RTO en el emisor se hace grande ya que depende de RTT. Los valores de RTO grandes hacen que el TCP sea menos sensible a las pérdidas de congestión reales. Por lo tanto, si se produce congestión, el emisor requiere un tiempo mayor para reducir su ventana de congestión, lo que agrava el escenario de congestión. Si RTO es grande, TCP tampoco podrá detectar condiciones libres de errores rápidamente. También se observa el problema de aumentar el tiempo de conexión, lo que resulta en un mayor consumo de energía [88].

3.5. Soluciones Propuestas

La creciente utilización de las redes inalámbricas ha puesto de manifiesto la necesidad de modificar el protocolo TCP. Originalmente diseñado para redes cableadas donde la congestión es la causa principal de las pérdidas de paquetes. Como ya se discutió, TCP reacciona inadecuadamente ante las pérdidas de paquetes no relacionadas con la congestión. De hecho, si se pierde un paquete de datos debido a interferencias de radiofrecuencia de corta duración en el canal de trasmisión, a pesar de que no hay desbordamientos de buffer, TCP tomara la decisión de reducir la ventana de congestión en forma innecesaria. En cambio, debería simplemente recuperarse de la pérdida y continuar la transmisión a la misma velocidad, como si nada hubiera sucedido.

TCP ha sido modificado en numerosas ocasiones para mejorar su rendimiento, lo que ha generado el surgimiento de variantes de TCP. Sin embargo, los distintos mecanismos implementados, en las distintas variantes, no reaccionan de la misma forma en los distintos entornos inalámbricos. A pesar de esto, se han realizado esfuerzos significativos en la investigación y el desarrollo de técnicas que mejorarían el rendimiento de TCP en la parte inalámbrica de redes heterogéneas.

Para conseguir este objetivo, las soluciones se dividieron, en primera instancia, en dos conceptos diferentes. El primero requiere algún tipo de soporte de la red, por ejemplo, que los nodos divulguen el estado de la red (ECN notificación de congestión explícita RFC 3168 [89]), confiando en canales de red para recuperar de las pérdidas no relacionadas con la congestión (retransmisión de capa de enlace), o bien aislando las rutas de transmisión inalámbricas de las

cableadas utilizando un host intermedio (I-TCP, M-TCP). La segunda engloba a las soluciones que mantienen la filosofía de Host-To-Host y al mismo tiempo proporcionan cierto nivel de resistencia a las pérdidas de paquetes no relacionadas con la congestión. La técnica de estimación de ancho de banda sentó las bases para que el lado emisor distinguiera entre una pérdida relacionada con la congestión y una pérdida no relacionada (aleatoria) sin ningún soporte de la red [12].

Estos esfuerzos para mejorar el rendimiento TCP de los diferentes grupos de investigación pueden agruparse en distintas propuestas, como se observa en la siguiente figura (Figura 29).

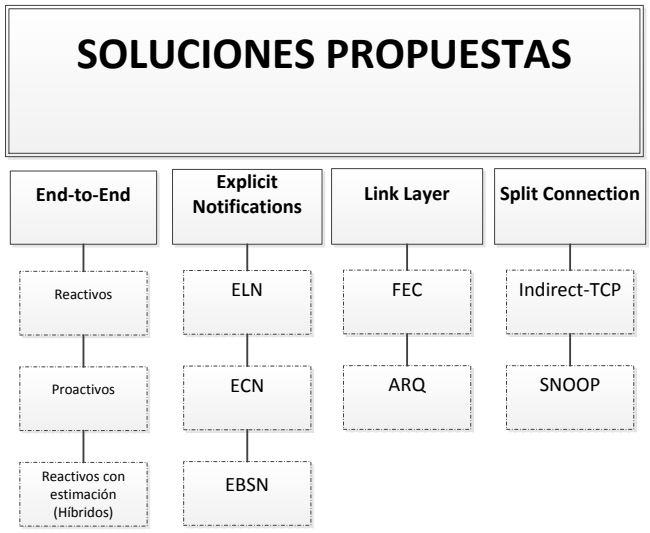

Figura 29: Soluciones propuestas

3.5.1. End-to-End (extremo a extremo)

El enfoque de extremo a extremo mantiene la estructura de la capa de red y requiere solo modificación en los extremos de la conexión, evitando así añadir complejidad a la red. Las diferentes variantes de TCP se implementan para intentar distinguir las pérdidas inalámbricas de las pérdidas por congestión. Para obtener el valor óptimo del tamaño de la ventana de congestión, el emisor y, en algunas condiciones el receptor, modifica sus mecanismos para manejar los casos de pérdidas de paquetes. De esta manera, el protocolo TCP debe realizar el control de la congestión extremo a extremo, ya que el diseño TCP asume que la red es una “caja negra”, lo que significa que no dependen de ningún soporte especial o información explicita de los nodos de la red.

El enfoque preferido por la mayoría de los investigadores es desacoplar el control de la congestión de la recuperación de pérdidas no relacionadas con la congestión. El receptor proporciona retroalimentación que refleja la condición de la red, y el emisor toma las decisiones para el control de la congestión.

El gran desafío es la capacidad de sondear con precisión el ancho de banda disponible, lo que es clave para un mejor rendimiento. Los mecanismos de control de congestión con este enfoque, se implementaron de forma reactiva, de forma proactiva o diferentes combinaciones de ambos.

En el caso del control de congestión reactiva, como se observó en el Capítulo 2 en el TCP Reno, la ventana de congestión se ajusta en función de la retroalimentación de ACK y ACK duplicados (DUPACK) generados en el receptor. TCP sondea el ancho de banda disponible aumentando continuamente la ventana de congestión gradualmente hasta que la red alcance el estado de congestión. En este sentido, la congestión es inevitable. Se han propuesto variantes reactivas de TCP como mejoras de Reno. Una de ellas, New Reno, modifica los mecanismos de Reno para hacer frente a múltiples pérdidas en una sola ventana, lo que es especialmente útil en las redes inalámbricas. Sin embargo, la limitación de New Reno es que no puede distinguir la causa de la pérdida de paquetes.

El uso de la opción SACK le indica al emisor que bloque de datos que se ha recibido con éxito en el receptor cuando se producen pérdidas de paquetes. Por lo tanto, el remitente tiene un mejor conocimiento sobre el número exacto de paquetes que se han perdido. SACK requiere modificaciones en los lados del remitente y el receptor. Pero tampoco puede definir la causa de las pérdidas y si debe o no reducir la tasa de envío.

En el control proactivo de congestión, el emisor intenta ajustar la ventana de congestión de manera proactiva a una tasa de envío óptima, de acuerdo con la información recopilada a través de la retroalimentación, que puede indicar indirectamente la condición de la red. De esta forma, el emisor reacciona de manera inteligente al estado de la red o a las pérdidas de paquetes. Se pueden emplear diferentes estrategias en el diseño para que el emisor pueda deducir la condición de la red. Como se vio en el Capítulo 2, TCP Vegas estima los

paquetes almacenados en el buffer del nodo cuello de botella observando el valor de RTT mínimo como referencia.

Los esquemas proactivos, en general, exhiben una manera más efectiva para manejar las pérdidas aleatorias, dada la forma de ajuste del tamaño de cwnd. Sin embargo, estos esquemas pueden llevar a problemas relacionados con la equidad con otros tráficos en la red.

Existen otros esquemas que combinan los enfoques, llamados reactivos con estimación de ancho de banda. En general posen un control de congestión reactivo y utilizan la estimación del ancho de banda para realizar ajustes. Un ejemplo de este enfoque es TCP Veno que se detalló en el Capítulo 2, y adopta la misma metodología que Vegas para estimar los paquetes en la red. Sugiere además una forma de diferenciar la causa de la pérdida de paquetes.

Los mecanismos de extremo a extremo realizan las retransmisiones en el emisor TCP y conservan claramente la semántica end-to-end. Las variantes que pertenecen a este enfoque son las que se analizan en este trabajo.

3.5.2. Explicit Notifications (Notificaciones explícitas)

Una forma de enfrentar el problema de disparar innecesariamente el control de congestión debido a las pérdidas propias del canal inalámbrico, es hacer que TCP distinga entre las pérdidas debidas a la congestión y las pérdidas debidas a otras causas.

En el enfoque anterior, los algoritmos de control de congestión se basan en la idea de que la red es una caja negra, que debe ser probada para buscar el punto de operación óptimo. Sin embargo, las soluciones de Explicit Notifications (ELN, ECN, etc.) intentan resolver la ambigüedad de la pérdida informando explícitamente al emisor sobre el motivo de la pérdida y, de esta manera, mejorar su respuesta y por ende el rendimiento.

Para ello, el router le notifica al emisor explícitamente la causa de la pérdida y con esta información, evita que TCP invoque el algoritmo de control de congestión cada vez que detecta una pérdida.

El principal inconveniente de estos esquemas es que se necesitan modificaciones tanto en los extremos como en los nodos de la red. Además,

para una implementación completa, se requiere soporte universal por parte de todos los routers intermedios, lo que difícilmente se puede lograr debido a la heterogeneidad de Internet.

ELN (Explicit Loss Notification)

La Notificación de Pérdida Explícita (ELN), implementa una opción a los acuses de recibo (ACK) de TCP. El mecanismo ELN informa al emisor la causa exacta de la pérdida del paquete, utilizando un bit en el encabezado del protocolo TCP, de modo que las retransmisiones se puedan desacoplar del control de congestión. Al enviarse la notificación como parte de la misma conexión, simplifica la implementación en el emisor.

Cuando se pierde un paquete en el enlace inalámbrico, los ACK subsecuentes al paquete perdido se marcan para identificar que se ha producido una pérdida no relacionada con la congestión. El emisor, al recibir esta información, realiza retransmisiones sin invocar los procedimientos de control de congestión asociados. Para identificar el paquete perdido debido a errores en los enlaces inalámbricos, el receptor utiliza el indicador de un error de CRC generado como resultado de la corrupción de un paquete en la capa de enlace. Esta información se pasa a la capa de transporte, que envía el mensaje ELN.

ELN-RXMT (Rexmit on first dup ELN ack) es una mejora de ELN, donde el emisor retransmite el paquete al recibir el primer acuse de recibo duplicado con la opción ELN, en lugar de esperar el tercer ACK duplicado, sin reducir el tamaño de su ventana.

Una de las propuestas de ELN es *TCP Casablanca* y funciona de la siguiente manera: el emisor marca cada paquete saliente con una de las marcas (IN/OUT) en un patrón consistente. Cuando se produce congestión en los nodos intermedios, debe existir un mecanismo de gestión de colas que descarte solo los paquetes marcados con la marca (OUT). De esta manera, el receptor recibe los paquetes con un patrón consistente de descartes, porque solo se descartan los paquetes marcados con (OUT), por lo que el receptor reconoce que los errores son errores de congestión. Por otro lado, si se produce un error inalámbrico, las caídas serán aleatorias entre todos los paquetes marcados y, por lo tanto, el receptor puede reconocer que estos

errores aleatorios son errores inalámbricos. Si el receptor diagnostica un error inalámbrico, marca los acuses de recibo con una notificación de pérdida explícita (ELN). Cuando el remitente TCP recibe un acuse de recibo duplicado, comprueba si el acuse de recibo contiene ELN y, de ser así, TCP considera que la pérdida es una pérdida propia de un enlace inalámbrico; de lo contrario, considera que se trata de un error de congestión. En caso de error de congestión, TCP reduce la ventana de congestión; de lo contrario, solo reenvía el paquete. Otra propuesta es *TCP Ifrane* que está basado en TCP Casablanca y los cambios se implementan solo en el emisor. Cuando el remitente envía un paquete, registra si este paquete está marcado como saliente (OUT) o entrante (IN). Si recibe un acuse de recibo duplicado que indica que se perdió un paquete, mira su registro y verifica con que marca se envió ese paquete. Si el paquete se marcó como OUT, el error se considera un error de congestión; de lo contrario, se considera un error inalámbrico.

ECN (Explicit Congestion Notification)

La notificación explicita de congestión se realiza mediante retroalimentaciones explícitas de la red sobre el estado de congestión de los enlaces. Para ello utiliza un campo de dos bits del encabezado IP, denominado ECN, que está compuesto por dos flags, ECT y CE, destinados a la señalización entre routers y extremos de la conexión. Además, utiliza dos flags en el encabezado TCP, ECE y CWR, para señalización entre los extremos TCP. ECE (ECN-Echo) lo utiliza el receptor para indicar al emisor que ha recibido un paquete con información ECN, mientras que CWR (Congestion Window Reduced) es utilizada por el emisor para indicarle al receptor que ha reducido su cwnd.

La Notificación explícita de congestión es una extensión de la técnica de detección temprana aleatoria (Random Early Detection RED). RED es un mecanismo activo de gestión de colas en los routers, que detecta la congestión antes de que la cola se desborde y proporciona una indicación de esta congestión a los nodos finales. Un router RED señala una congestión incipiente a TCP al descartar paquetes probabilísticamente antes de que la cola se quede sin espacio en el buffer. RED mantiene dos niveles de umbral: mínimo y máximo. Descarta el paquete probabilísticamente si y solo si el tamaño promedio de la cola se encuentra entre los umbrales mínimo y máximo. Si el

tamaño promedio de la cola excede el umbral máximo, todos los paquetes que llegan se descartan. En ECN, los paquetes se marcan en lugar de descartarse cuando el tamaño promedio de la cola está entre mínimo y máximo. Cuando el receptor recibe dicha notificación de congestión, el receptor TCP informa al emisor en el ACK posterior, sobre la congestión incipiente, que a su vez activará el algoritmo de control de congestión. De esta manera, el emisor es capaz de distinguir entre las pérdidas debidas a la congestión y otras pérdidas.

Los beneficios de ECN son numerosos, especialmente cuando se implementa con un esquema de gestión de colas activo que puede proporcionar una notificación temprana de congestión [90]. Sin embargo, el problema del enfoque es que necesita cambios en la red y en los extremos de la conexión.

Una de las variantes de TCP que hace uso de las notificaciones explicitas de congestión es *TCP Jersey*. Es una variante proactiva que adapta la velocidad de envío de acuerdo con la condición de la red, pero con una implementación diferente. El ancho de banda disponible se estima en función de la tasa de acuses de recibo. La alta tasa de llegada de reconocimiento significa que los paquetes pueden llegar al otro extremo rápidamente y, por lo tanto, una alta capacidad de red. Además, en este enfoque, los nodos intermedios deberían poder marcar paquetes cuando se espera congestión para notificar al remitente.

TCP Jersey consta de dos componentes clave, el algoritmo de estimación del ancho de banda ABE (Available Bandwidth Estimation) y la advertencia de congestión CW (Congestion Warning) que requiere la configuración de los routers de la red para que alerten sobre la congestión a los hosts finales, marcando los paquetes cuando hay signos de incipiente congestión. Por lo tanto, requiere que este implementado ECN para obtener CW. ABE es una adición del lado del emisor TCP que estima continuamente el ancho de banda disponible para la conexión y lo utiliza para ajustar su velocidad de transmisión. El marcado de paquetes por los routers ayuda al emisor a diferenciar efectivamente las pérdidas de paquetes causadas por la congestión de la red de aquellas causadas por errores de enlace inalámbrico. Basado en la indicación de congestión implicada por CW y la estimación de ABE, TCP Jersey calcula el tamaño óptimo de la ventana de congestión.

TCP Jersey puede sufrir una degradación del rendimiento cuando coexiste con tráfico no TCP en el enlace inverso, porque no puede estimar la capacidad del enlace correctamente ya que el tráfico agregado en la ruta inversa puede retrasar los ACK, por lo que subestimará el ancho de banda disponible. La versión mejorada se llama TCP New Jersey y utiliza marcas de tiempo (Timestamps) de los ACK en lugar de su tasa de llegada para calcular el ancho de banda estimado, lo que resuelve el problema porque cada confirmación tiene una marca de tiempo que permite remitente para calcular el retraso de la ruta hacia adelante.

EBSN (Explicit Bad State Notification)

Notificación explícita de mal estado (EBSN) propone un mecanismo para actualizar el temporizador TCP en el emisor para evitar que disminuya su ventana de congestión. La estación base BS intenta transmitir el paquete utilizando el esquema de retransmisión de la capa de enlace. Sin embargo, mientras el BS está realizando una recuperación local, el RTO del emisor aún puede expirar, causando una retransmisión innecesaria. Cuando falla por problemas del enlace inalámbrico, la estación base envía una notificación explícita de mal estado EBSN al emisor. El enfoque EBSN evita el timeout del emisor mediante el uso del mensaje EBSN durante la recuperación local. El mensaje EBSN hace que el emisor restablezca su temporizador de tiempo de espera de retransmisión. De esta manera, se eliminan los tiempos de espera en el emisor durante la recuperación local, de modo que el paquete no sea retransmitido por la capa de transporte.

3.5.3. Link Layer (Capa de Enlace)

Si bien el enfoque tradicional consiste en delegar el control de congestión y el control de errores a las capas superiores, el alto valor de BER es un problema del enlace inalámbrico y, por lo tanto, puede resultar interesante ubicar la solución del problema haciendo más rápida la retransmisión local sin avisar a TCP. De esta forma, la recuperación de errores en forma local puede ser más rápida y más adaptable a las características del enlace. Lo que significa que TCP no está involucrado en manejar las perdidas inalámbricas y los errores de los enlaces inalámbricos, sino que es resuelto en la capa de enlace de datos.

En este esquema, se implementan protocolos confiables de nivel de enlace, que aportan sus propias técnicas de retransmisión para mejorar la confiabilidad de la comunicación, independiente de los protocolos de nivel superior. De esta manera, trata de aumentar la calidad del enlace inalámbrico, ocultando a la capa de transporte (TCP) las características del enlace inalámbrico y tratando de resolverlo en la capa de enlace. Se implementa sobre el enlace inalámbrico entre la estación base y el nodo móvil.

Este tipo de soluciones se basan en la modificación del protocolo para incluir principalmente dos técnicas: corrección de errores, usando técnicas *FEC* (Forward Error Correction), y retransmisiones basadas en *ARQ* (Automatic Repeat Request). Las técnicas de corrección de errores tratan de proporcionar suficiente información redundante en cada paquete, para permitir que el receptor no solo detecte el error, sino que también lo corrija sin requerir retransmisión. Por otro lado, las técnicas ARQ permiten que el receptor solo detecte el error y, por lo tanto, solicite la retransmisión del paquete localmente; limitando así la respuesta del TCP, principalmente, a las pérdidas por congestión. En el RFC3366 [91] se revisaron las principales estrategias que se adopta ARQ para optimizar el rendimiento de TCP sobre los enlaces inalámbricos.

En la mayoría de los sistemas inalámbricos actuales, el uso del esquema de ARQ permite recuperarse de los errores inalámbricos y provee de una transferencia de datos relativamente confiable a las capas superiores. Los esquemas ARQ y FEC se utilizan en la capa de enlace para obtener la fiabilidad de los datos y hacer que la conexión inalámbrica se vista por TCP como un enlace confiable, aunque con mayores valores de latencia. Esta solución tiene la ventaja de ser independiente de TCP, por lo que no requiere su modificación. Sin embargo, esa independencia es relativa, pues no se puede independizar los temporizadores de TCP y de ARQ, ni del reordenamiento de paquetes producido por la capa de enlace. Esta interacción puede reducir el rendimiento. Esta mejora puede volverse contraproducente para el tráfico que atraviesa múltiples enlaces. Esto puede generar que la latencia sea variable en el enlace, introduciendo ARQ un grado de Jitter en

función del retraso de la capa física, generando cambios significativos en los valores de RTT causados por las retransmisiones de la capa de enlace.

FEC se utiliza para corregir un número relativamente pequeño de errores. Sin embargo, los códigos de corrección reducen el ancho de banda efectivo del enlace inalámbrico, por la carga adicional transportada. Además, FEC necesita potencia de procesamiento adicional para verificar los errores, lo cual es una restricción en los dispositivos móviles. Adaptive FEC reduce la sobrecarga al elegir códigos FEC dinámicamente dependiendo de la situación de error.

FEC se puede usar en conjunto con otros mecanismos de capa de enlace para mejorar el rendimiento sobre enlaces inalámbricos.

Los esquemas de recuperación local como los que utilizan retransmisiones de capa de enlace y de capa de transporte TCP funcionan bajo el supuesto de que la latencia del enlace inalámbrico es pequeña en comparación con la latencia de toda la conexión. Pero en los casos en que la latencia del enlace inalámbrico sea considerablemente alta, habrá una disputa entre las retransmisiones TCP del remitente y las retransmisiones locales, lo que dará como resultado una reducción del rendimiento.

La retransmisión a nivel de enlace puede causar pérdidas de congestión, ya que la retransmisión significa una reducción en el ancho de banda. Debido a la reducción del ancho de banda, habrá más paquetes en la cola, lo que puede provocar que se activen los mecanismos de control de congestión en el emisor. Esto puede generar que el emisor retransmita el paquete y también lo haga el mecanismo de retransmisión de la capa de enlace. Además, si los paquetes perdidos están demasiado desordenados, se activarán los mecanismos de Fast Retransmit de TCP en el emisor. Esto nuevamente causaría pérdida de ancho de banda debido a retransmisiones duplicadas.

Dentro de las implementaciones de este enfoque, se destaca el protocolo *AIRMAIL (Asymmetric Reliable Mobile Access in Link Layer)* que combina técnicas FEC de corrección de errores y retransmisiones para recuperación de pérdidas de paquetes. La estación base BS envía una ventana completa de datos antes de que el nodo móvil envíe un ACK. La idea de este enfoque no es desperdiciar ancho de banda en ACK y limitar la cantidad de trabajo realizado

por nodo móvil para conservar energía. Otro ejemplo es el protocolo *TULIP (Transport Unaware Link Improvement Protocol)* se basa en la retransmisión automática de los paquetes al detectar una pérdida, introduciendo una aceleración MAC que le permite disminuir el tiempo necesario para la recepción de los ACK sin renegociar el acceso al medio.

Sin embargo, una de las implementaciones más conocidas es el protocolo *Snoop* utiliza mejoras de nivel de enlace compatibles con TCP. La idea principal detrás de este protocolo es almacenar en caché los datos TCP no reconocidos en la estación base y realizar retransmisiones locales, para aliviar los problemas causados por las altas tasas de errores de bits. En este enfoque, se realizan modificaciones en las estaciones base y los nodos móviles. Introduce un módulo, llamado agente Snoop, en la estación base. El agente monitorea cada paquete que pasa a través de la conexión TCP en ambas direcciones y mantiene un caché de los segmentos TCP enviados a través del enlace que no han sido reconocidos (ACK) por el receptor. Las pérdidas de paquetes se detectan tanto por la llegada de un acuse de recibo duplicado (DupACK) o por la expiración de su propio temporizador de retransmisión (local timeout). En esta instancia, el módulo Snoop retransmite el paquete perdido si lo tiene en Cache y suprime los acuses de recibo duplicados. De esta manera, la estación base (BS) oculta la pérdida del paquete, evitando que el emisor invoque innecesariamente al algoritmo de Fast Retransmit y al mecanismo de control de congestión. Esto es muy útil para los errores en ráfagas, cuando más de un paquete se pierde en la misma ventana. El agente Snoop esencialmente oculta las fallas del enlace inalámbrico mediante el uso de retransmisiones locales.

Las soluciones de capa de enlace tradicionalmente funcionan independientemente de la capa de transporte, y esto puede causar problemas a TCP. Algunos esquemas ARQ retransmiten paquetes fuera de orden, lo que puede causar ACK duplicados, la invocación innecesaria de Fast Retransmit y, por lo tanto, degrada el rendimiento. La interacción entre los tiempos de espera de retransmisión de nivel de enlace y los tiempos de espera de nivel de transporte puede producir una importante reducción del rendimiento. Para resolver este problema, los protocolos de la capa de enlace deban tener algún

grado de conocimiento de la conexión TCP. Esto podría permitir que la capa de enlace bloquee ACK duplicados de TCP y evitar que ambas capas inicien retransmisiones. Aunque este enfoque mejora el rendimiento de TCP, el uso de cifrado de la carga útil de IP, por ejemplo, IPSEC, produce que los nodos intermedios no puedan saber que el tráfico que se transporta en la carga útil es TCP.

El protocolo Snoop monitorea y almacena en caché segmentos TCP y realiza retransmisiones locales de los segmentos TCP que requieren que la capa de enlace sea consciente de TCP. Sin embargo, las ideas detrás de TULIP y AIRMAIL son diferentes y ambos dependen de temporizadores locales de retransmisión, ya que la estación base utilizada no necesita ser consciente de TCP. El uso de temporizadores TCP y temporizadores de capa de enlace, puede producir la contención de retransmisión entre las dos capas [92].

La principal ventaja de las propuestas es que no se requieren modificaciones en los puntos finales. Sin embargo, los protocolos de la capa de enlace podrían afectar negativamente el rendimiento del TCP. Mientras que los mecanismos de extremo a extremo no necesitan ningún soporte de la capa de enlace, los enfoques basados en la recuperación de una pérdida inalámbrica en forma local necesitan un soporte sustancial en la capa de enlace.

3.5.4. Split Connection (Conexión Dividida)

La idea principal en el enfoque de los protocolos de Split Connection es la de aislar los problemas relacionados con los enlaces inalámbricos de los protocolos existentes de red. Propone que la conexión entre el nodo fijo y el nodo inalámbrico se separe en dos conexiones independientes. Esto se logra separando la conexión en una cableada entre el host fijo y la estación base, y una conexión inalámbrica entre la estación base y el host móvil. La conexión cableada no necesita ningún cambio en el Software en los Host Fijos y la conexión inalámbrica puede utilizar un protocolo móvil especializado para proveer un mejor rendimiento. Sin embargo, impone una mayor complejidad y dependencia de la estación base. En la siguiente figura (Figura 30) se puede observar la división de la conexión entre el host fijo y el host móvil.

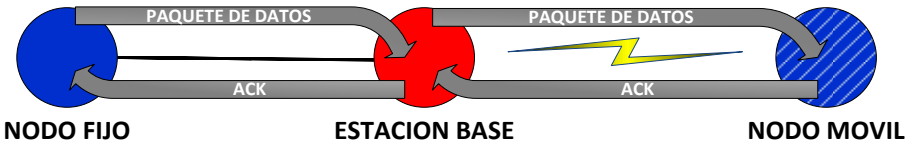

Figura 30: División de la conexión en dos conexiones independientes

El principal inconveniente de este tipo de soluciones es que los ACK recibidos por el emisor no significan, necesariamente, que los paquetes han sido recibidos por el TCP receptor. Por ejemplo, cuando un nodo cambia a la cobertura de otra estación base (handoff) puede ocurrir que se pierdan algunos paquetes y, sin embargo, el emisor puede haber recibido los ACK correspondientes a esos paquetes, aunque puede no haber sido entregados al receptor. Este mecanismo viola claramente la semántica de TCP de extremo a extremo.

Una de las primeras propuestas de este tipo de soluciones es *Indirect TCP* (I-TCP) [93]. La idea es que, dado que se tienen dos clases completamente diferentes de subredes (cableadas e inalámbricas), se puede dividir cada conexión TCP en dos conexiones en el punto donde las dos subredes se encuentran, es decir, en la estación base. El BS mantiene una conexión TCP con el nodo fijo, mientras que al mismo tiempo utiliza otro protocolo diseñado para un mejor rendimiento a través de enlaces inalámbricos con el nodo móvil. Este enfoque tiene en cuenta el hecho de que los nodos móviles MH (Mobile Host) tienen recursos limitados y traslada gran parte de la tarea de red a la BS. De esta manera todos los paquetes destinados a la MH son recibidos, almacenados y reconocidos por la BS. Los paquetes enviados al host móvil se reciben primero y se almacenan temporalmente en la estación base. La estación base acusa recibo (ACK) de los paquetes al host fijo antes de reenviar los paquetes al host móvil a través del enlace inalámbrico. La ventana de congestión se mantiene por separado para las conexiones inalámbrica y fija. Por lo tanto, el nodo fijo FH (Fix Host) está protegido de la característica poco confiable de las conexiones inalámbricas.

Sin embargo, como se describió, los ACK recibidos por el emisor no significa que los paquetes hayan sido recibidos por el destinatario y, además, al ejecutar TCP en forma independiente en ambas conexiones, estos pueden tener

diferentes tasas de envío, lo que puede causar desbordamiento de buffer de la BS.

Otra de las implementaciones de este tipo de solución es *Mobile TCP (M-TCP)* [94] se diseñó con la idea de la perdida de rendimiento de TCP por los efectos de las frecuentes y prolongadas desconexiones y para los enlaces inalámbricos de baja velocidad y ancho de banda variable. También se consideró la escasez de energía en los dispositivos móviles. De esta manera, se diseñó un protocolo que asigna dinámicamente un ancho de banda fijo para cada nodo móvil. Se implementa la recuperación local de errores y se reduce el consumo de energía en el nodo móvil logrando un número bajo de paquetes duplicados. El protocolo también asegura un handoff eficiente y el manejo de los problemas generados por las desconexiones. Si bien utiliza un enfoque basado en conexión dividida, trata de preservar algo de la semántica de extremo a extremo enviando el ACK al emisor solo cuando el receptor reconoce los datos. Además, conserva la estimación del tiempo de espera del emisor en función del RTT completo. En esta solución se utiliza TCP estándar en la red fija o cableada, mientras que M-TCP se utiliza sobre el enlace inalámbrico, que está diseñado para recuperarse rápidamente de pérdidas inalámbricas debido a desconexiones y para eliminar tiempos de espera en serie.

Otra de las variantes de este enfoque es *MTCP (Multiple TCP)*. La idea es introducir un protocolo de Capa de Sesión llamado MHP (Mobile Host Protocol), en la estación base y el Host móvil. Esta capa de sesión está diseñada de tal manera que compensa la falta de fiabilidad y la imprevisibilidad del enlace. Se proponen dos implementaciones para los protocolos de capa de sesión. En la primera, MHP (Mobile Host Protocol), en la que se utiliza TCP para la conexión a través del enlace inalámbrico. En la segunda, se utiliza el protocolo SRP (Selective Repeat Protocol) solo a través del enlace inalámbrico. SRP está diseñado para recuperarse rápidamente de las altas perdidas de paquetes en ráfagas. El receptor devuelve un ACK Selectivo (SACK) cuando recibe un paquete fuera de secuencia. De esta forma puede recuperar más de un paquete en un RTT. La capa de sesión en el host móvil intercepta una solicitud de conexión TCP desde el host móvil al host fijo y establece una conexión TCP con su par en la estación base. La capa de sesión en la estación base

configura un agente de capa de sesión en nombre de la conexión solicitada. Este agente a su vez establece la conexión con el host fijo en nombre del host móvil. Este agente retransmite el tráfico desde la primera conexión a la segunda conexión. En forma similar a I-TCP, la estación base almacena los datos enviados entre host móvil y el host fijo. Segmenta los datos almacenados en segmentos más pequeños y los reenvía al host móvil.

Existen otras propuestas más extremas de Split Connections tal como *Mobile End Transport Protocol (METP)* donde se propone eliminar las capas TCP e IP de los hosts inalámbricos y reemplazarlas por un protocolo de transporte diseñado específicamente para ejecutarse directamente sobre la capa de enlace. Las capas IP y TCP se transfieren desde el dispositivo inalámbrico a la estación base y se implementa METP directamente sobre la capa de enlace, utilizando mecanismos de confiabilidad de capa de enlace y encabezados reducidos para mejorar el rendimiento en el enlace inalámbrico. Por lo tanto, esto requiere modificar la capa de red en la estación base y el dispositivo inalámbrico.

En las soluciones de Split Connections no se conserva la semántica de extremo a extremo. Sin embargo, se logra mejor rendimiento que el TCP estándar cuando las desconexiones no son largas. Si hay traspasos de celdas frecuentes, la sobrecarga involucrada en la transferencia del estado de conexión entre las estaciones bases involucradas, pueden ser importantes y, por lo tanto, aumentar la latencia. La estación base necesita mantener grandes buffers para garantizar que los datos puedan almacenarse durante largas desconexiones. La principal ventaja de estos protocolos es que proporcionan compatibilidad con los protocolos existentes, por lo que no requieren ninguna modificación en los hosts fijos. Sin embargo, violan la semántica de extremo a extremo de TCP y necesitan traducir de un protocolo a otro en la estación base, lo que lleva una sobrecarga significativa.

3.6. Variantes TCP End-To-End para Redes Inalámbricas

Se han propuesto varias soluciones para superar los problemas de TCP a través de enlaces inalámbricos. Se describen a continuación, algunas soluciones basadas en el protocolo de la capa de transporte (TCP) para evitar

la degradación del rendimiento debido a las características del enlace inalámbrico. Son soluciones que mantienen la idea de host-to-host y al mismo tiempo proporcionan cierto nivel de resistencia a las pérdidas de paquetes no relacionadas con la congestión [95].

Algunos algoritmos de control de congestión utilizan la métrica de retraso para fines específicos que no están directamente relacionados como señal de congestión. Estas técnicas utilizan la señal de retardo para calcular un tamaño óptimo de la ventana de congestión después del evento de pérdida de paquetes, pero también para diferenciar entre pérdidas de paquetes relacionadas con la congestión y las que no se deben a la congestión. La técnica de estimación de ancho de banda propuesta como parte de TCP Westwood sentó las bases para que el emisor pudiera deducir entre una pérdida relacionada con la congestión y una pérdida no relacionada (aleatoria) sin ningún soporte de la red. Se identificaron algunas debilidades en este enfoque, tales como, la sobreestimación del ancho de banda, lo que dio lugar a la evolución y los refinamientos de Westwood que intentan mitigar los problemas descubiertos. En la siguiente figura (Figura 31) [18] se observa una parte de dicha evolución.

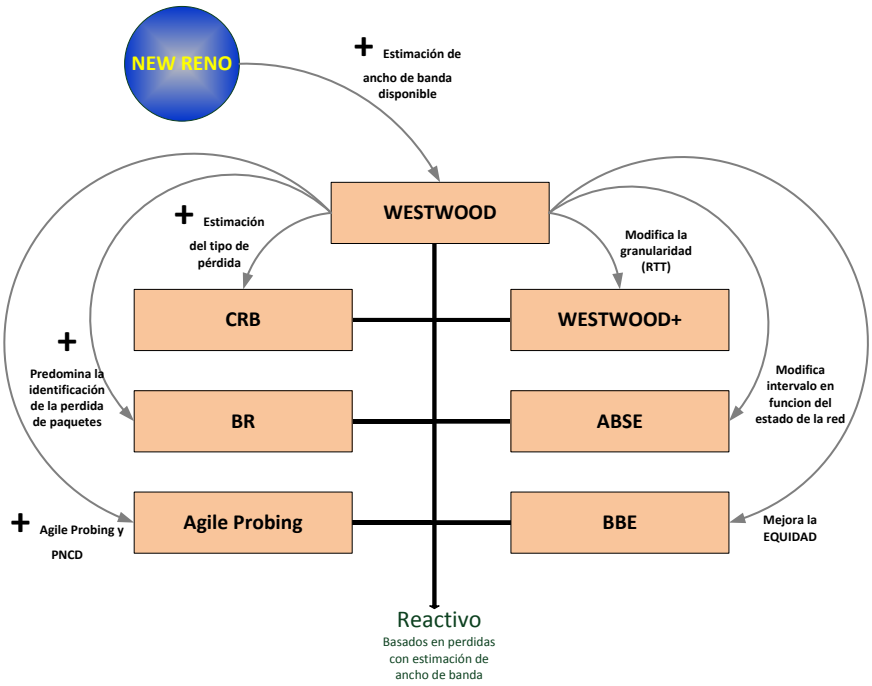

Figura 31: Evolución de las variantes

3.6.1. TCP Westwood/Westwood+

TCP Westwood [96] mantiene el concepto extremo a extremo de TCP e independiente de la red y, al mismo tiempo, puede mejorar significativamente la eficiencia de la transferencia de datos en redes propensas a errores. Modifica el emisor de TCP New Reno para que estime el ancho de banda disponible de la red de manera dinámica, midiendo y promediando la tasa de retorno de ACK recibidos. La dinámica de la ventana de congestión durante las fases Slow Start y Cogestión Avoidance no cambia. La idea general es usar la estimación de ancho de banda (BE) para establecer el tamaño de la ventana de congestión y del umbral de Slow Start (ssthresh) después de un episodio de perdida de paquete, para mejorar el uso del enlace. En lugar de la disminución multiplicativa, implementa un comportamiento de disminución basado en la estimación del ancho de banda disponible en el momento de la congestión. La idea principal es establecer las ventanas de control después de la indicación de pérdida de modo que coincida con el ancho de banda disponible en el momento de la congestión. El ancho de banda disponible se estima a través del flujo de paquetes ACK que llegan al emisor. La idea es mantener el tamaño de la ventana de congestión igual al valor de BDP del camino en la red y, de esta manera, lograr la máxima utilización de los enlaces, con la mínima longitud de cola en los nodos intermedios.

La ventaja del mecanismo propuesto es que el emisor TCP se recupera más rápido después de pérdidas, especialmente en conexiones con RTT elevados, o sobre enlaces inalámbricos donde las pérdidas esporádicas son debidas a problemas en el enlace en vez de a la congestión. Para hacerlo, Westwood reemplaza las acciones ciegas de control de la congestión de New Reno que se desencadenan mediante detección de pérdida o si se reciben tres ACK duplicados, con un procedimiento heurístico para establecer la ventana de congestión en un valor óptimo, logrando una recuperación más rápida. Como óptimo, la heurística considera un valor que corresponde a una velocidad de transferencia de datos observada en el pasado reciente. De hecho, si hay un error aleatorio debido a la interferencia inalámbrica, el óptimo reflejaría la mejor opción para el emisor, es decir, la transmisión sin reducción de velocidad. En otro caso, si se pierde un paquete debido a la congestión en la red, la tasa de

recepción de datos recientemente observada por el receptor es exactamente la velocidad a la que la red puede entregar datos del emisor. Si el emisor continúa la transmisión a una velocidad igual a la observada por el receptor, la cantidad de paquetes transmitidos será igual a la cantidad de paquetes entregados, por lo tanto, las colas no crecerán.

Para que el emisor pueda descubrir, en todos los casos de pérdida, la tasa observada por el receptor, la solución propuesta y posteriormente patentada es realizar una estimación en el emisor de la tasa de entrega real en función del mecanismo de notificación existente (ACK). Es decir, la tasa de ACK observada por el emisor será igual a la velocidad de entrega de datos observada por el receptor. Para calcular el ancho de banda de ruta de ida realmente utilizado, solo tenemos que multiplicar la tasa de ACK por la cantidad de datos reconocidos. El cálculo del ancho de banda se mantiene a largo plazo incluso si el receptor pierde o retrasa algunos ACK; es decir, una disminución en la tasa de ACK será compensada por un aumento en la cantidad de datos reconocidos. De esta manera, el protocolo mejora la performance usando como realimentación la medida extremo a extremo del ancho de banda disponible. Incluso si se pierden algunos ACK o el receptor decide utilizar un mecanismo de delayed ACK, el cálculo de estimación a largo plazo no se verá afectado significativamente, pues el paquete ACK llevará el reconocimiento de una mayor cantidad de datos.

Después, dicha estimación del ancho de banda se utiliza para establecer el valor de la ventana de congestión y el umbral de Slow Start (ssthresh) ante un timeout o la recepción de 3 ACK duplicados. En particular, cuando se reciben tres ACK duplicados, tanto la ventana de congestión (cwnd) como el umbral de Slow Start (ssthresh) se establecen iguales al ancho de banda estimado (BE) multiplicado por el tiempo de ida y vuelta mínimo (RTTmin) medido durante la duración de la conexión. Cuando caduca un tiempo de espera, ssthresh se establece como antes, mientras que cwnd se establece igual a uno. Por lo tanto, cuando se reciben 3 DUPACK, TCP Westwood establece el umbral ssthresh y cwnd de la siguiente manera:

```
if (se reciben n DUPACK)
        if (CWND> ssthresh)     /* Algortimo  Congestion Avoid. */
                ssthresh = BE*RTTmin;
```

```
                CWND = ssthresh;
        endif
        if (CWND< ssthresh)     /* Algoritmo Slow Start */
                ssthresh = BE*RTTmin;
                if (CWND > ssthresh)
                        CWND = ssthresh
                endif
        endif
endif
```

Donde RTTmin es el RTT mínimo experimentado. BE es el ancho de banda estimado. TCP Westwood supone que cuando se reciben 3 DupACK en la fase Congestion Avoidance, el ancho de banda disponible, se utiliza por completo. Por lo tanto, los valores ssthresh y cwnd deben reflejar el ancho de banda estimado.

Así, cuando se reciben n ACK duplicados, ssthresh y cwnd se establecen al valor de BE*RTTmin, si se encontraba en la fase de Congestion Avoidance. Si TCP está en fase de Slow Start, se establece en forma análoga, ssthresh al valor de BE*RTTmin, pero se define cwnd al valor de ssthresh únicamente si cwnd>ssthresh. En otras palabras, durante Slow Start, la ventana de congestión todavía experimenta un crecimiento exponencial.

Si se indica una pérdida de paquete por timeout, TCP Westwood establece el umbral ssthresh y cwnd de la siguiente manera:

```
if (timeout)
        if (cwnd> ssthresh)   /* Algortimo  Congestion Avoid. */
                ssthresh = BE*RTTmin;
                if (ssthresh < 2)
                        ssthresh = 2;
                        cwnd = 1;
                else
                        cwnd = BE*RTTmin;
                endif
        endif
        if (cwnd< ssthresh)     /* Algoritmo Slow Start */
                ssthresh = BE*RTTmin;
                if (ssthresh < 2)
                        ssthresh = 2;
                        cwnd = 1;
                else
                        cwnd = BE*RTTmin;
                endif
        endif
endif
```

Donde RTTmin es el RTT mínimo experimentado y BE es el ancho de banda disponible estimado.

La estimación del ancho de banda se obtiene midiendo la frecuencia de los ACK y las muestras de ancho de banda se calculan como la cantidad de paquetes entregados dividido por el tiempo entre llegadas de dos ACK. Esas estimaciones de ancho de banda se filtran para lograr una estimación precisa y justa. De esta manera, TCP Westwood modifica el Incremento Aditivo y la Disminución Multiplicativa (AIMD) y establece adaptativamente las tasas de transmisión para eliminar el comportamiento oscilatorio de TCP New Reno y maximizar la utilización de los enlaces.

Para mitigar las fluctuaciones, Westwood procesa la estimación de ancho de banda en dos niveles. En el primer nivel, la estimación instantánea se calcula (Ecuación 52) al recibir un paquete ACK de la siguiente manera.

$$b = \frac{d}{\Delta} \qquad \textbf{Ecuación 52}$$

Donde d es la cantidad de datos reconocidos por el ACK y Δ es el tiempo transcurrido desde la recepción del último ACK.

En el segundo nivel, los valores instantáneos calculados se promedian con un filtro especial de tiempo discreto de la siguiente forma (Ecuación 53).

$$BE = \alpha(\Delta).BE^{-1} + [1 - \alpha(\Delta)].(\frac{b + b^{-1}}{2}) \qquad \textbf{Ecuación 53}$$

Donde α(Δ) es el coeficiente promedio, como una función de Δ; b y b^{-1} son muestras actuales y anteriores de la estimación del ancho de banda; y B^{-1} es el valor promedio previamente calculado de la estimación.

Uno de los inconvenientes que presenta, es la correcta estimación de ancho de banda. Si se estima en forma incorrecta, se desempeña en forma poco eficiente. Este cálculo puede ser particularmente incorrecto en ciertas condiciones de la red. Por ejemplo, en presencia del efecto de compresión ACK, cuando los ACK se retrasan y se agrupan de forma diferente debido a la congestión en el camino inverso, el promedio discreto de las muestras de

estimación instantánea del ancho de banda conduce a una sobreestimación sustancial.

Con la intención de corregir alguno de estos problemas, por ejemplo, la compresión de ACK, se modificó en la variante *TCP Westwood+* [97], la estimación para que se calcule con la granularidad de RTT. Es decir, en la fórmula de la Ecuación 52, d es ahora la cantidad de datos reconocidos durante el último RTT y Δ es el valor de RTT. Esta estimación del ancho de banda promedio durante la última RTT se define como promediada a largo plazo utilizando la conocida técnica de suavizado exponencial, con un factor de α = 0,9, modificando la Ecuación 53 de la siguiente forma (Ecuación 54).

$$BE = \alpha \,.\, BE^{-1} + (1-\alpha).b$$ **Ecuación 54**

Aunque la variante TCP Westwood, al igual que TCP Reno no pueda distinguir si la pérdida del paquete se debe a una congestión del enlace o a un error especifico causado por un aumento súbito del BER debido a la atenuación de la señal, ante la pérdida de un paquete vuelve a comenzar con el valor previo del umbral en base a la estimación del ancho de banda. Este comportamiento puede aumentar el rendimiento del protocolo TCP ante un entorno con pérdidas aleatorias, respecto a TCP Reno y, en estas condiciones, no lograr un grado de equidad con las variantes reactivas de TCP [98].

La siguiente ilustración (Figura 32) muestra la evolución de la ventana de la variante TCP Westwood ante los eventos de pérdida.

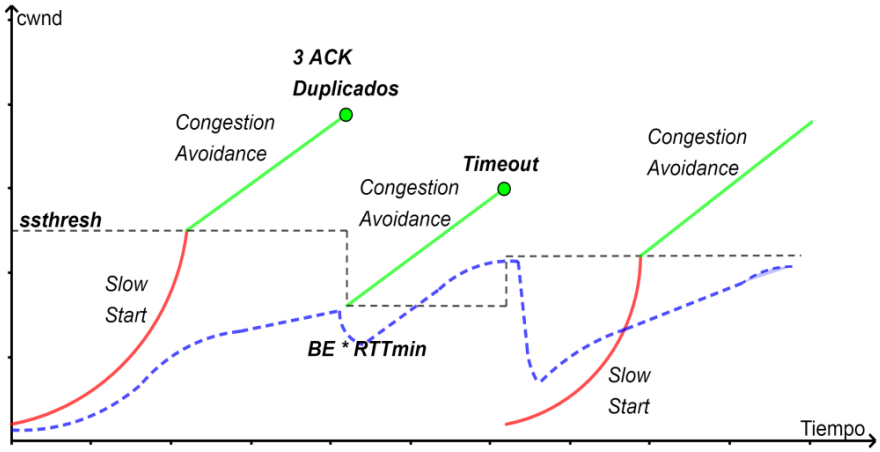

Figura 32: Evolución de la ventana de congestión TCP Westwood

En la Figura 32 se observa que después del timeout, ssthresh no se reduce a la mitad del valor de cwnd, sino que se produce una reducción adaptativa de la ventana y le asigna el valor BE * RTTmin.

3.6.2. TCPW CRB (Combined Rate and Bandwidth estimation)

Como se describió anteriormente, bajo ciertas condiciones de red, la técnica de estimación de ancho de banda arroja resultados muy inexactos [99]. Otra solución se propuso con la variante *TCPW CRB (Westwood with Combined Rate and Bandwidth estimation)*, que refina el algoritmo de estimación complementándolo con una técnica conservadora de cálculo del ancho de banda a largo plazo RE (Rate Estimation). Es similar a la propuesta de Westwood+, pero el período de muestreo es una constante T predefinida, en lugar del valor RTT medido.

Los resultados experimentales muestran que la estimación a largo plazo evita la sobreestimación si una red experimenta congestión. Al mismo tiempo, es probable que subestime el ancho de banda en presencia de errores aleatorios. Para abordar simultáneamente los problemas de subestimación y sobreestimación, CRB mantiene también la estimación de TCP Westwood, BE (Bandwidth Estimation). Al detectar una pérdida de paquete, CRB elige una de las estimaciones según supone el tipo de pérdida: BE para pérdida aleatoria y calcula cwnd como BE * RTTmin, mientras que utiliza la nueva estimación RE (Rate Estimation) para la pérdida por congestión, es decir, establece el tamaño de la ventana de congestión en RE * RTTmin.

De esta forma, TCPW CBR logra una mejor equidad sacrificando en alguna media la eficiencia.

3.6.3. TCPW ABSE (Adaptive Bandwidth Share Estimation)

Se propuso *TCPW ABSE (Westwood con Adaptive Bandwidth Share Estimation)* como una extensión de CRB [100]. ABSE aprovecha la idea de la estimación del ancho de banda dual mediante la introducción de un mecanismo de adaptación del intervalo de muestreo. En otras palabras, en lugar de dos intervalos de muestra predeterminados, ABSE cambia continuamente el

intervalo dependiendo de un estado estimado de la red. De esta manera, la longitud del intervalo de muestreo Δ es controlado por la estimación de estado de red como lo expresa la siguiente ecuación (Ecuación 55).

$$\Delta = max\left(\Delta_{min}, \frac{RTT(VE - RE)}{VE}\right) \qquad \textbf{Ecuación 55}$$

Donde Δ_{min} es un intervalo de muestreo mínimo predefinido, VE es una estimación del estado de las colas similar al de Vegas, y RE es una estimación de ancho de banda promediada exponencialmente con un intervalo de muestreo igual al RTT, similar a Westwood+.

De acuerdo a la Ecuación 55, si el valor actual de RE es significativamente más pequeño que lo predicho por la estimación tipo VE de Vegas, es probable que la red se encuentre en una congestión severa. Por lo tanto, en forma similar a CRB, se calculará un intervalo de muestreo largo, es decir, que Δ se acerca a RTT cuando RE tiende a 0 (RE → 0). En el caso opuesto, cuando VE y RE están cerca, o lo que es lo mismo, cuando un número de paquetes perdidos están cerca de cero y cuando RTT está cerca del valor mínimo, se utilizará el intervalo mínimo de muestreo. Claramente, estas observaciones de los casos fronterizos cumplen con la definición de la heurística CRB. De esta manera, la adaptación del intervalo de muestreo mejora la precisión de la estimación en los períodos de transición.

Además del cálculo adaptativo del intervalo de muestreo, ABSE también define un coeficiente de suavizado exponencial para promediar muestras de la estimación de ancho de banda. La idea básica es aumentar el promedio, si la disponibilidad de los recursos de la red está cambiando de manera muy dinámica y, de esta manera, la nueva muestra debería tener un mayor impacto en el valor promedio, y en el caso contrario menor.

En forma similar a CRB, no cambia sustancialmente el concepto de Westwood de “recuperación más rápida” y mantiene las propiedades interfairness (Interprotocolo).

3.6.4. TCPW BR (Bulk Repeat)

Dado que Westwood no puede manejar de manera eficiente una tasa alta de errores aleatorios [101], se propuso una nueva variante llamada *TCPW BR (Westwood with Bulk Repeat)*. Si bien está basada en Westwood, integra un mecanismo especial de detección de congestión por perdida de paquete. Tras cada detección de pérdida, si se estima que no está relacionada con la congestión, BR aplica políticas de recuperación muy agresivas y altamente optimistas, en lugar de las originales. El mecanismo propuesto de estimación del tipo de pérdida en BR está formado por dos algoritmos: Queuing Delay Estimation Threshold (QDET) y el algoritmo de Rate Gap Threshold.

El algoritmo QDET se basa en el concepto de medición del retardo de puesta en cola. La principal diferencia es que QDET mantiene dos umbrales Ts y Te. Ts representa una condición para ingresar al estado en donde se supone que todas las pérdidas son causadas por la congestión. Te es una condición para volver al estado predeterminado cuando se supone un tipo de pérdida sin congestión.

El algoritmo Rate Gap Threshold se basa en la comparación de la estimación del ancho de banda de Westwood (BE) con una fracción α del rendimiento esperado (VE). Si la estimación de Westwood es menor que una fracción predefinida del rendimiento esperado, se supone que la pérdida se debe a un evento de congestión; de lo contrario, se supone un tipo de pérdida no relacionada con la congestión.

El razonamiento detrás de esta comparación es que cuando no hay congestión, incluso frente a una cantidad sustancial de pérdidas de paquetes, el rendimiento de los datos sigue siendo relativamente cercano al esperado, es decir, RTT está cerca de RTTmin y la cantidad de paquetes de datos entregados durante el último RTT está cerca de cwnd.

La utilización de dos mecanismos independientes de estimación de tipo de perdida aumenta la precisión de la estimación y reduce el número de falsos positivos. Esto es especialmente crucial debido a las políticas de BR al detectar una pérdida no debida a la congestión. Si este es el caso, BR inmediatamente retransmite todos los paquetes de datos que han sido transmitidos y aún no

han sido reconocidos, y no modifica el tamaño de la ventana de congestión. Además, BR modifica el algoritmo de backoff del temporizador de retransmisión al limitar el tiempo de espera a un máximo, durante pérdidas de paquetes no causadas por congestión. Esta decisión mejora el tiempo de recuperación en entornos donde la probabilidad de pérdida es extremadamente alta.

3.6.5. TCPW BBE (Bottleneck and Buffer Estimation)

En algunas circunstancias, los flujos TCP que ejecutan los algoritmos de control de la congestión de Westwood, Westwood+ o ABSE pueden ser altamente injustos [102] con otros flujos TCP que compiten por los recursos de la red (intrafairness). Para resolver este problema, se introdujo en esta variante el algoritmo *BBE (Bottleneck and Buffer Estimation)*, un algoritmo que refina la política de reducción del tamaño de la ventana de congestión de Westwood al detectar una pérdida. Específicamente, BBE complementa la política de reducción de la ventana de congestión, a través de un coeficiente variable adicional μ ≤ 1, de la siguiente manera (Ecuación 56).

$$cwnd = RTT_{min} \,.\, BE \,.\, \mu \qquad \textbf{Ecuación 56}$$

Este coeficiente varía en función del estado de la red, cuando el retardo de cola actual es cercano a un máximo, se considera que la red experimenta congestión y μ tiene un valor de 0,5. De lo contrario, la red es libre de congestión, y μ tiene un valor de 1. El cálculo propuesto del coeficiente μ se define en la siguiente ecuación (Ecuación 57).

$$\mu = \frac{Q_{max}}{(Q + Q_{max})} \qquad \textbf{Ecuación 57}$$

Donde Q es el retardo de cola actual y Q_{max} no es solo el retraso máximo de cola observado durante la vida útil de la conexión, sino muestras de retardo de cola suavizadas exponencialmente obtenidas justo antes de cada evento de detección de pérdida.

Esta técnica permite a BBE adaptarse fácilmente a los cambios de la red. Además, BBE reconoce el problema de sobreestimación en el algoritmo original de Westwood y propone una técnica de estimación híbrida. En particular, BBE calcula la tasa alcanzable como una suma ponderada de dos estimaciones: la

tasa de muestreo variable Bv (por ejemplo, tasa de ACK, como en Westwood original) y una tasa de muestreo constante Bc (por ejemplo, 1 / RTT como en Westwood +). Este cálculo esta expresado en la siguiente ecuación (Ecuación 58).

$$B = \gamma \,.B_V + \,(1 - \gamma\,)\,.B_C$$ **Ecuación 58**

El coeficiente γ, similar al coeficiente de reducción adicional mencionado anteriormente μ, se basa en el concepto de retardo de la cola y esta esperado por la siguiente ecuación (Ecuación 59).

$$\gamma = \frac{1}{e^{\alpha \cdot \frac{RTT}{RTT_{max}}}}$$ **Ecuación 59**

Donde α es una constante positiva.

Como se observa en el cálculo de la estimación B (Ecuación 58), el tiempo de muestreo constante es más preciso cuando la red experimenta congestión, lo que se expresa con el hecho que Bc tiene más peso cuando RTT está cerca de RTT_{max}, y la tasa variable es más precisa cuando la red está libre de congestión, es decir, Bv tiene más peso cuando RTT está lejos de RTT_{max}.

Los resultados han demostrado una buena equidad con los flujos heredados de New Reno, junto con un buen rendimiento de transferencia de datos comparable al algoritmo original de Westwood.

3.6.6. TCPW-A (Agile Probing)

TCPW-A (Westwood with Agile Probing) [103] se diseñó con el objetivo de proporcionar mejor rendimiento en redes con ancho de banda altamente dinámico, con alto valor de BDP, como así también en entornos con pérdidas aleatorias. Para lograr este objetivo, TCPW-A agrega dos mecanismos: Agile Probing (AP) y Persistent Non Congestion Detection (PNCD).

Cuando una conexión comienza o se reinicia después de un timeout, la ventana de congestión se expande exponencialmente hasta el umbral ssthresh preestablecido arbitrariamente, para luego entrar en un aumento lineal de la ventana en la fase de Congestion Avoidance. TCPW-A usa Agile Probing (AP), un mecanismo que restablece repetidamente de forma adaptativa el umbral

ssthresh según la estimación del ancho de banda estimado (BE) y fuerza el crecimiento exponencial del valor de cwnd. Cada vez que el ssthresh se restablece a un valor más alto que el actual, la ventana de congestión sube exponencialmente al nuevo valor. El resultado es una convergencia rápida a un valor de ssthresh apropiado. Este mecanismo también se invoca después de que se detecta ancho de banda adicional.

Cuando se reciben 3 ACK duplicados, TCPW-A restablece los valores de ssthresh y de cwnd, a través de la fase Agile Probing (AP) de la siguiente forma.

```
if (se reciben 3 DUPACKS)
        switch to Congestion Avoidance phase;
else (se recibe un ACK)
        if (ssthresh < BE*RTTmin)
                ssthresh =BE*RTTmin);
        endif
        if (cwnd >ssthresh)            /*mini linear increase phase*/
                cwnd = cwnd +1/cwnd;
        else
        if (cwnd <ssthresh)     /*mini exponential increase phase*/
                cwnd = cwnd +1;
        endif
endif
```

El mecanismo PNCD se ocupa de detectar ancho de banda disponible que no está siendo utilizado. Mientras se encuentra en la fase de Congestion Avoidance, PNCD identifica la disponibilidad de ancho de banda adicional, invoca al mecanismo de AP.

Similar a TCP Westwood, todos estos algoritmos modificados pueden sufrir problemas de equidad en diferentes grados si el RTT_{min} es una sobreestimación de RTT_{base}.

CAPITULO 4 - Simulaciones

4.1. Introducción

Para evaluar el rendimiento de un sistema o una red, se puede utilizar un modelo analítico, la medición directa del sistema o red, o bien, realizar una simulación. Cada una de estas tres técnicas tiene sus fortalezas y debilidades.

El modelado analítico es una técnica teórica donde se analiza un sistema investigando modelos matemáticos. La ventaja de esta técnica es que puede realizarse en etapas tempranas del proceso de desarrollo, antes de implementar realmente el sistema, mientras que la principal desventaja es que se basa en gran medida en simplificaciones y suposiciones, ya que el sistema real no está involucrado.

Otra de las formas de realizar la evaluación del sistema, es tomar medidas del sistema real, lo que requiere que al menos partes del sistema estén implementadas. Las pruebas en el sistema real, por supuesto, darán resultados muy precisos, pero al costo de tener que implementarlo antes de poder evaluarlo. Encontrar fallas en el diseño en este punto es costoso.

Las simulaciones son más cercanas a la realidad que los modelos analíticos, pero aun así poseen detalles abstractos y se pueden realizar a lo largo de todo el proceso de desarrollo. Al principio, pueden dar una buena indicación de qué tan bien funcionará el sistema, pero también se pueden usar en etapas posteriores del proceso para encontrar fallas en el diseño. Otro punto fuerte de la simulación es que es fácil realizar pruebas a gran escala que de otro modo serían difíciles de implementar en un banco de pruebas o en el propio sistema.

Las simulaciones utilizan modelos. Un modelo es un objeto que se esfuerza por imitar algo del mundo real. Este modelo no puede recrear completamente el objeto del mundo real, y tendrá que abstraer algunos de sus detalles.

Un tipo muy común de simulador de red, un simulador de eventos discretos (DES, por sus siglas en inglés), es un buen ejemplo de simulador dinámico. Estos son simuladores donde los eventos se programan dinámicamente a medida que pasa el tiempo. Consisten en una serie de eventos que ocurren en momentos discretos de la ejecución de la simulación. Discreto significa que el simulador se define para un conjunto de valores finitos. A pesar de esto, pueden ocurrir eventos al mismo tiempo. La mayoría de los simuladores de redes son de este tipo.

La simulación de redes es la metodología por la cual se implementa una red en una computadora para su posterior evaluación. Permite probar escenarios que son difíciles o costosos de simular en entornos reales. La realización de estas pruebas en redes con equipos reales tiene como límite la capacidad del laboratorio, acotado en general, en la cantidad y variedad de equipos, lo que reduce la posibilidad de modificar el diseño y fuerza a trabajar con una sola topología. Ahora, si bien un simulador no puede sustituir el trabajo directo con equipos, puede proveer en cambio: facilidad de acceso, manejo de diversas topologías, equipos y protocolos, rapidez en el armado, trabajo con diferentes tipos de escenarios, algunos de estos escenarios pueden ser configurados y modificados si tener que cambiar hardware. Aunque no son reales, imitan de cerca la realidad. De esta manera, para cambiar de prueba solo se tiene que cambiar ciertos parámetros en el software del simulador, sin tener que rearmar una topología que, además del tiempo necesario, puede requerir equipos que no están disponibles.

El objetivo principal de la simulación de red es observar las características y el comportamiento de una red donde se puede simular, emular y analizar los resultados finales de las simulaciones de red.

La gran influencia de los entornos de la simulación a día de hoy se pone de manifiesto, entre otras cosas, por la cifra incalculable de publicaciones científicas que en algún punto de su desarrollo han utilizado estas plataformas.

Actualmente, existen gran cantidad de simuladores de red, cada uno de ellos con características y capacidades diferentes.

A continuación [104], se describen brevemente algunas de las herramientas más conocidas y utilizadas para la simulación de redes, que cuentan con módulos para simular el comportamiento de escenarios basados en redes inalámbricas.

Network Simulator 2: [105] Sus posibilidades de uso, disponibilidad al público y las características del software, permiten que gran cantidad de usuarios desplieguen implementaciones sobre esta plataforma, convirtiéndola en una de las herramientas preferidas por la comunidad investigativa. La disponibilidad del código fuente tanto para su inspección, modificación y la libertad para la aplicación de cualquier usuario, ha promovido generación de módulos entre de la comunidad, que permiten la creación de nuevos protocolos y sistemas para ser simulados. Es una fuerte ventaja en el ámbito de la investigación que el código fuente de la herramienta se encuentra disponible. Sin embargo, el uso simultáneo de dos lenguajes de programación implica la necesidad de un gran conocimiento previo para el uso de la herramienta. De esta manera, es necesario el uso de la herramienta por largos periodos de tiempo para lograr su implementación eficaz.

El simulador NS2 fue diseñado para investigación en redes y para proveer soporte para simulaciones TCP, y protocolos de envío múltiple sobre redes cableadas e inalámbricas. Aunque el software no contiene una herramienta que retorne gráficas de una manera eficiente, existen diversas herramientas que permiten hacerlo, lo cual minimiza el problema y, permite aprovechar todos los valores que arroja el simulador.

Sin duda alguna, se trata del simulador de código libre dominante (aunque dejó de tener actualizaciones en el 2009), habiendo contado con una gran cantidad de desarrolladores y contribuciones externas. Como elementos negativos, se puede destacar una clara falta de modularidad y una lenta curva de aprendizaje.

Network Simulator 3: [106] En 2005, varios investigadores comenzaron a desarrollar un nuevo simulador de red para reemplazar el simulador de red NS-

2. Varios de sus desarrolladores más importantes deciden crear un nuevo entorno de simulación especializado en el análisis de redes. Una de las razones principales para utilizar C++ es que la mayoría de las implementaciones de pila de protocolos se implementan en C, lo que facilita su incorporación en las simulaciones NS-3. Se trata de un proyecto muy activo que recientemente se ha visto ampliado con módulos de análisis de tecnologías de acceso inalámbricas. Esta herramienta también es un simulador de eventos discretos de red, cuyos objetivos principales son lograr un mayor énfasis en los niveles 2 y 4 del modelo OSI. En un principio la compatibilidad con NS-2 no es un objetivo del proyecto, por lo que NS-3 no es una actualización de NS-2 sino un proyecto diferente. Esta herramienta también es de código abierto. Se enfoca en el sector educativo como herramienta de simulación para enseñanza e investigación. Además, como factor diferenciador, permite realizar un amplio conjunto de simulaciones en redes basadas en IP y no IP. También, cuenta con una gran comunidad de desarrolladores que constantemente liberan versiones estables, específicamente cada tres meses, lo cual implica la inclusión de nuevos modelos, documentados, validados y con soporte para realizar simulaciones en escenarios actualizados y con los últimos estándares de telecomunicaciones.

Es un software de distribución gratuita bajo la licencia GNU GPLV2, el núcleo de su operación está en lenguaje C++ y las instrucciones se generan en Python.

EstiNet: [107] es un simulador y emulador de red, que se originó de NCTUns (National Chiao Tung University (NCTU) network simulator), el cual fue usado ampliamente en investigaciones relacionadas con redes de comunicaciones desde el año 2002 hasta el 2011. NCTuns se volvió un software comercial en 2011, y se renombro como EstiNet. Esta nueva versión del software comercial incluye la capa física, de control de acceso, de red, de transporte y de aplicación. Cuenta con una interfaz gráfica amigable y con tres características que lo diferencian ampliamente de los demás simuladores: (i) integra la pila de protocolos del núcleo de Linux, lo cual permite obtener un comportamiento real de los protocolos de las capas tres y cuatro simuladas; (ii) es compatible con aplicaciones de red reales del sistema operativo Linux, lo cual permite

desarrollar y probar aplicaciones y funcionalidades en entornos simulados garantizando un igual comportamiento a cuando se despliegue la aplicación en un dispositivo de red real con sistema operativo Linux; y finalmente, (iii) interactúa con dispositivos de red reales, mediante la función de emulación un dispositivo o red simulada puede interactuar con una red o un dispositivo del mundo real.

Gracias al gran desarrollo que ha tenido la herramienta de simulación EstiNet, el software tiene la posibilidad de simular, características y elementos de redes cableadas e inalámbricas como routers, switchs, y dispositivos finales; protocolos de red como Spanning Tree, algoritmos de ruteo como OSPF y RIP; también permite simular puntos de acceso inalámbricos, bajo los protocolos 802.11a/g/n, modelos de canales, entre otros.

Opnet Modeler: Esta herramienta de modelado de redes cuenta con dos factores diferenciales importantes: un motor de simulación de eventos con alto desempeño, lo cual permite ejecutar rápidos análisis de impacto y comportamiento en modelos de redes, y cuenta con una amplia biblioteca de modelos de red compatibles y listos para ser usados, o modificados para generar modelos de aplicaciones y protocolos personalizados, lo cual podría resultar en una reducción de los tiempos de simulación. Además de esto, no se centra en un solo sector de aplicación, en este caso, este simulador puede ser usado en la industria, ya que permite simular comportamientos de extremo a extremo en diferentes diseños tecnológicos, también es usado en la academia para investigación y desarrollo de redes, así como la generación de tecnologías o protocolos de comunicación inalámbricos.

Gracias a las numerosas ventajas y debido a la poderosa interfaz gráfica y estadística que maneja, el uso por parte de grupos académicos es alto, puesto que la manera de simular es muy intuitiva, pero necesita de gran cantidad de conocimientos previos en redes y programación para enfrentarse a la herramienta, por lo cual la curva de aprendizaje de OPNET es alta. De otro lado, aunque se debe aclarar que la licencia del simulador es comercial, existe una versión de este software que no genera cargos económicos para utilizarlo en actividades netamente académicas. Puesto que OPNET proporciona los

mecanismos necesarios para el desarrollo fluido de una simulación, permitiendo arrastrar componentes para conformar topologías de red.

OMNet++: [108] no es simulador de red en sí mismo, sino una herramienta basada en eventos discretos de propósito más general. Es una arquitectura genérica que se puede utilizar en varios dominios de problemas, por ejemplo: modelado de redes, modelado de protocolos, modelado de multiprocesadores y validación de arquitecturas de hardware, entre otros. OMNeT ++ proporciona infraestructura y herramientas para escribir diferentes tipos de simulaciones. También proporciona una serie de utilidades para usar junto con la simulación. Mientras que la funcionalidad del módulo está escrita en C ++, la estructura del módulo, y sus secciones, se describen en el lenguaje NED de OMNeT ++. Este es un lenguaje de script muy simple. Como punto negativo se destaca su escaso soporte, con una comunidad de usuarios menos activa. Esta herramienta se conforma por módulos escritos en C++ que se comunican entre sí por paso de mensajes, donde módulos simples, pueden conformar módulos compuestos y los niveles jerárquicos no tienen límites. Se debe tener en cuenta, que a pesar de que la herramienta de simulación presenta muchos beneficios, los módulos que la conforman no están del todo desarrollados, lo cual implica que los programadores deban modificar el código existente o realizar implementaciones de módulos nuevos para cubrir componentes de red aún no especificados dentro del paquete. Esto hace que la curva de aprendizaje de la herramienta sea alta ya que modificar el código fuente de los componentes de red no es una tarea sencilla.

JSim: [109] Conocido formalmente como JavaSim, se trata de un simulador composicional, centrado en el desarrollo a través una jerarquía de componentes autónomos, que utiliza Java como lenguaje principal de desarrollo. JSIM fue desarrollado por un equipo del laboratorio de computación distribuida en tiempo real (DRCL). En un ambiente de simulación y animación basado en Java que soporta la animación basada en Web. Este proyecto fue patrocinado por la Fundación Nacional para la Ciencia (National Science Foundation, NSF), el programa de modelado y simulación de redes DARPA, la Universidad del Estado de Ohio y la Universidad de Illinois en Urbana-Champaign. Su código está disponible en Internet y es de libre uso. Este

simulador ha sido usado para emular ambientes de red inalámbricas con múltiples especificaciones y gran capacidad para establecer detalles. Inicialmente JSim contaba con su propia herramienta de lenguaje por comandos (Tool Command Language, TCL), en la cual mediante comandos se podían diseñar, simular y analizar los diferentes escenarios. Según la documentación oficial, JSim puede integrarse con gEditor, un editor gráfico para diseñar y simular escenarios de red, este editor usa XML como lenguaje para modelar los diferentes componentes de red. Lo anterior deja ver que su curva de aprendizaje es alta, ya que se requieren conocimientos avanzados en lenguaje TCL, Java, redes y programación para poder interactuar con la herramienta.

Finalmente, como elemento diferenciador JSim resalta que es un simulador que no requiere alto poder computacional y dado que se ejecuta en la máquina virtual de Java puede ejecutar diferentes simulaciones con un menor uso de memoria y mayor capacidad para escalar. Además de esto, dado que es un software de código abierto y en constante reforma por su comunidad, no tiene un foco principal de aplicación y deja el camino para que los mismos usuarios puedan generar módulos de procesamiento y análisis.

Gns3: [110] Software gráfico de simulación de red que permite la emulación de redes complejas, es de libre descarga, pero necesita de las imágenes de los dispositivos Cisco para emular el comportamiento de estos y dichas imágenes se deben adquirir directamente con el fabricante. La interfaz gráfica de la herramienta es considerada intuitiva para el usuario. Como principal característica y diferenciador de esta herramienta se resalta la capacidad de simular y emular dispositivos de red reales, lo cual permite configurar y desplegar redes de comunicaciones en un computador. Dado que estos dispositivos ejecutan los sistemas operativos de los dispositivos reales, una de las limitantes a la hora de usar GNS3 es la cantidad de recursos de hardware disponibles en el equipo de simulación.

Netsim Tetcos: [111] es un software para modelado y simulación de redes y protocolos enfocado a la investigación y desarrollo en redes, también usado en aplicaciones de defensa. Actualmente es comercial. La interfaz gráfica permite configurar los diferentes escenarios, ejecutar la simulación de eventos

discretos, controlar y visualizar las simulaciones de forma animada, y analizar los resultados obtenidos en las simulaciones. Por otro lado, su consola de comandos de línea (CLI) permite establecer el escenario de simulación mediante archivos de configuración XML y ejecutar simulaciones de eventos discretos. Sus capacidades han permitido lograr simulaciones bastante precisas y con resultados importantes en el ámbito científico. Es importante resaltar el valor agregado y el uso o enfoque que se le da a este simulador. Cuenta con una interfaz gráfica que permite diseñar y modelar entornos de red con gran nivel de detalle, este simulador ofrece tres tipos de licencia, las cuales ofrecen diferentes protocolos y aplicaciones para labores académicas o investigativas de simulación. Un elemento importante a resaltar es la gran cantidad de aplicaciones disponibles en su versión profesional, las cuales son redes celulares, de internet de las cosas, de sensores inalámbricos, de área personal, de radio cognitiva, de 5G, vehiculares y enlaces de radio de uso militar.

4.2. Elección del simulador

A partir del estudio de las distintas alternativas de simulación [112], una condición necesaria debe ser que el software sea de código abierto (Open Source) y gratuito. Dentro de las muchas ventajas que tiene el software open source, resultan de peso para la selección en este trabajo, la posibilidad de compartir, modificar y estudiar el código fuente y, por otro lado, promueve la colaboración entre usuarios. Esta característica supone el desarrollo rápido y variado de multitud de herramientas y no depender del vendedor para las actualizaciones. Otro de los aspectos, que no es trivial, es que no tiene costo de licencia.

De esta manera se descartaron las alternativas comerciales o de código propietario. Es por esta razón, que se evaluaron los simuladores de eventos discretos NS-2 y NS-3.

En base a lo evaluado se seleccionó a NS-2 como herramienta de simulación para el presente proyecto, por lo que toda implementación está sujeta a su arquitectura.

Dentro de los principales motivos de esta elección, en primer lugar, es que se trata de una plataforma gratuita y de código abierto, lo que otorga la flexibilidad necesaria para utilizar y modificar libremente los módulos existentes a conveniencia, así como añadir nuevos elementos.

En segundo lugar, se destaca la gran popularidad de NS-2, pues es ampliamente utilizado en el campo académico e investigativo siendo el entorno de simulación favorito por la comunidad científica, contando con un porcentaje de publicaciones muy superior al resto.

En tercer lugar, cumple con las necesidades de este trabajo pues soporta el protocolo TCP con distintas variantes de su control de congestión, tanto sobre redes cableadas como inalámbricas (802.11).

En cuarto lugar, si bien el aprendizaje de la herramienta puede resultar complejo, existe gran cantidad de información disponible que puede contribuir a una mejor comprensión y aprendizaje de la herramienta. Existe una importante variedad de bibliografía, tutoriales y desarrollos de scripts realizados por terceros disponible en bibliotecas de uso público. Además, se tiene acceso para la consulta a otros grupos de investigación que ya venían trabajando con este simulador, lo que puede acelerar el proceso de aprendizaje.

En quinto lugar, a pesar de que una de las principales desventajas de NS-2, es que no posee la capacidad de representar gráficamente los resultados, estos datos pueden ser leídos, procesados y representados con herramientas tales como Perl, AWK [113], XGRAPH [114], NAM que se pueden integrar al entorno.

4.3. Network Simulator 2 (NS-2)

NS-2 es el simulador de eventos discretos de código abierto más utilizado en la investigación y el desarrollo de las comunicaciones. Debido a su flexibilidad y naturaleza modular, se convirtió rápidamente en un éxito en la comunidad de investigación de redes después de su lanzamiento en 1989. Durante muchos años, la herramienta de simulación de redes NS-2 ha sido el estándar de facto para la investigación académica sobre protocolos de redes. Se ha distribuido como software libre y de código abierto, lo que ha permitido que estudiantes,

ingenieros e investigadores aporten scripts que permiten que este siempre en proceso de desarrollo.

NS-2 cubre una gran cantidad de aplicaciones de red, protocolos, tipos de redes, elementos de red y modelos de tráfico.

Desarrollado inicialmente en los laboratorios de investigación de la universidad californiana de Berkeley (LBNL, Lawrence Berkeley National Laboratory), en 1989 como variante del simulador REAL, está concebido para el estudio del comportamiento de una red cualquiera de telecomunicaciones. Se basa en el protocolo IP, implementando protocolos de transporte como el TCP, RTP, SRM y el UDP, protocolos de las capas de aplicación (HTTP, FTP, CBR, etc.), protocolos para la gestión de la congestión como RAP y TFRC, mecanismos y procedimientos para la gestión de las colas como DropTail, RED (Random Early Detection), ECN (Explicit Congestion Notification) y CBQ, además de algoritmos de ruteo, de multicasting y finalmente, algunos protocolos MAC para la simulación de redes LAN, WLAN, tanto para redes cableadas como inalámbricas.

NS-2 está basado en dos lenguajes de programación, C++ (a nivel de datos) y Tool Command Language (OTcl) lenguaje de script, Tcl con extensión orientada a objetos desarrollado en MIT (a nivel de control). El núcleo se basa en C ++ y la interfaz se basa en OTcl. Cada módulo está escrito en C ++, mientras que OTcl se usa para ensamblar y configurar módulos, así como para programar eventos. Al estar escrito en C++, agiliza la simulación y el lenguaje OTcl se utiliza para actuar sobre el simulador.

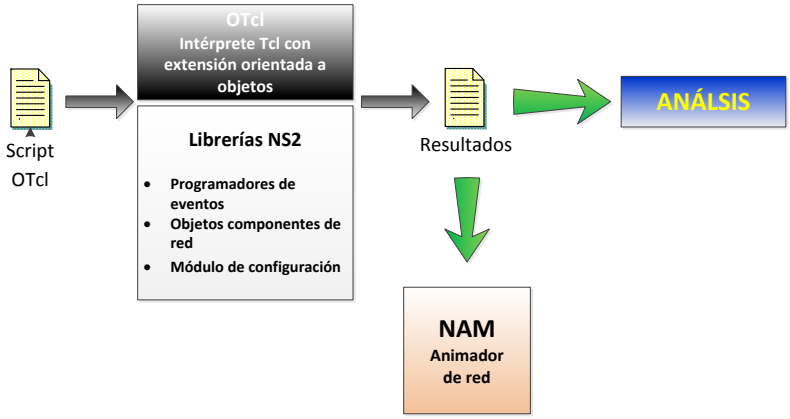

Figura 33: Secuencia de simulación de NS-2

La utilización de estos dos lenguajes permite combinar la velocidad de ejecución propia de un programa compilado con la sencillez de OTcl. El OTcl es un lenguaje interpretado, simple e intuitivo, generalmente es utilizado en la redacción de los scripts por la programación de los acontecimientos que definen los escenarios de simulación.

La Figura 33 representa la secuencia de simulación de NS-2 desde el punto de vista del usuario. Como se observa, se inicia con un script OTcl que es procesado por NS2, posteriormente los resultados de la simulación se almacenan en archivos de registro que genera NS-2. Cada paquete transmitido durante la simulación genera una línea en el archivo de registro donde se describen varios parámetros del paquete, teniendo la posibilidad de guardar datos específicos de cada nodo o de toda la red. También permite generar un archivo de registro especial, el cual es interpretado por NAM (Network Animator), que ofrece una visualización básica de la simulación.

El C++ es el lenguaje compilado con el que se implementa en NS la mayor parte de los modelos de los objetos de red utilizados durante las simulaciones. Siendo un lenguaje compilado, tiene la ventaja de ser más veloz y eficiente, aunque precisa de una mayor complejidad de escritura. El script OTcl define el escenario de red a simular. Cuando se ejecuta la simulación, los comandos en OTcl inicializan los objetos internos en C++ que realizan la acción requerida. De esta forma cuando se realiza una simulación el código OTcl se vincula con las clases de C++ para poder ejecutar las funciones implementadas en C++. En la siguiente figura (Figura 34) se observa la arquitectura del simulador.

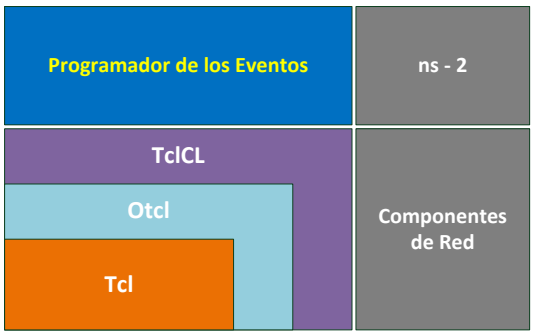

Figura 34: Arquitectura de NS-2

Hay dos clases de jerarquías: el compilado en C++ y la jerarquía interpretada OTcl, con correspondencia uno a uno entre ellos. Durante una simulación se

inicializan y trabajan, al mismo tiempo las dos jerarquías de objetos. Esto permite separar la implementación de las funciones relativas a la simulación de los datos en la red, de la implementación de la lógica de control y configuración de la red. Es decir, donde en C++ define el mecanismo interno de la simulación y el OTCL pone a punto la simulación, ensamblando y configurando los objetos en una programación discreta de eventos. La mayor parte de los componentes de red están implementadas en C++ y están disponibles para OTcl a través de un vinculador OTcl (OTcl linkage) que se implementa usando la interfaz TclCL.

Mediante el lenguaje C++ los usuarios pueden crear nuevas clases [115], en las cuales se implementa la funcionalidad para los diferentes objetos de red. El usuario también puede construir sus propios protocolos y modificar según sus necesidades los ya existentes, por tanto, es necesario utilizar el programa fuente del simulador y luego volver a compilar y vincular todos los módulos para crear un nuevo ejecutable de NS-2. Tcl se utiliza principalmente para el desarrollo rápido de prototipos, aplicaciones "script", interfaces gráficas y pruebas. Es un lenguaje interpretado, y su código puede ser creado y modificado dinámicamente.

NS-2 utiliza el lenguaje de programación OTcl para desarrollar los scripts de simulación, en los que se configuran las características de los escenarios (topología, parámetros de los enlaces, protocolos, modelos de tráfico), así como también para establecer la planificación de los eventos. En el script de OTcl proporcionado por el usuario, se puede definir una topología de red en particular, los protocolos y aplicaciones específicas que se desea y la forma de la salida que se desea obtener desde el simulador [116].

Network Simulator utiliza una variedad de herramientas para manipular objetos y flujos de datos y también proporciona herramientas que permiten interpretar y analizar los resultados obtenidos de la simulación de una manera gráfica. Estas herramientas son NAM, Network AniMator y XGraph que crean una representación a partir de los ficheros de traza generados por NS-2. NAM es una herramienta que permite visualizar trazas de simulación de red que utiliza el archivo de resultados generado por el simulador. XGRAPH se utiliza para graficar datos y trazar los datos de archivos en un mismo gráfico y tiene la capacidad de manejar conjuntos de datos de tamaño ilimitado. Se basa en la

lectura de archivos de trazas con formato de columnas x-y, obtenidos como resultado de la ejecución de scripts escritos en algún lenguaje, como por ejemplo TCL. XGRAPH tiene la capacidad de personalizar los gráficos en cuanto a colores, títulos, nombre de los ejes; además tiene la posibilidad de exportar los resultados en distintos formatos de tal manera que puedan ser utilizados posteriormente en reportes u otros documentos

Los ficheros de traza reflejan todos los eventos que han sucedido durante la simulación de la red. Cada una de las trazas incluidas en estos ficheros contiene información sobre el instante de tiempo en que ha ocurrido, sobre los nodos implicados, información sobre la trama MAC, sobre el paquete IP, e información adicional que depende del evento que se ha producido o del tipo de paquete del que se trata.

CATEGORIA	Modelo y Tecnología
ROUTING	AODV, AODV-UV, ZRP, AOMDV, IS-IS, RCDS, DLSR, DMCR,DYMO, UM-OLSR, ATM, HWMP
Cableado, Inalámbrico y Movilidad	ARP, HDLC, GAF, MPLS, LDP MAC: CSMA, Satellite Aloha, Queuing: Drop Tail, RED, RIO, SRR, WFQ, REM, IEEE 802.11b, IEEE802.15.4, IEEE 802.11 support, IEEE 802.11 PHY-MAC design and implementation, IEEE 802.11 PCF, IEEE 802.11 PSM, IEEE 802.11e EDSA and CFB simulation model, IEEE 802.11e HCCA module, IEEE 802.15.4, IEEE 802.16 model, IEEE 802.16 model MIRACLE framework, IEEE 802.16 wireless mesh networks, NS2-emulation extension (optimized for wireless networks, IR-UWB, TDMA DAMA satellite support, WiMAX, CRCN, UCBT Bluetooth, SUNSET underwater networking, VANAT, CanuMobiSim, EURANE extensions, BonnMotion, a java mobility scenario generator and analyzer, GPRS, BlueHoc, a bluetooth extension, CIMS
Transporte	TCP Pacing, UDP, DCCP for wired and wireless networks, Linux TCP Congestion Control for 12 different congestion control algorithms (BIC, CUBIC, HighSpeed TCP, H-TCP, TCP-HYBLA, NewReno, Scalable TCP, Vegas, Westwood, TCP Veno, TCP Compound and TCP Low-Priority), Network Simulation Cradle, TCP Westwood, Extensions to RTP code, Freeze-TCP, Multipath TCP, Data Center TCP (DCTCP), TCP ex Machina, SCTP, TCP Rate-Halving Algorithm, MFTP, SNACK
Otros modelos y tecnologías	Satellite networks, Topology and traffic generation, Differentiated services, Integrated services, Scheduling and queue management, Multicast, DTN, Application layer

Tabla 3: Implementado en NS 2.35

Además de estas herramientas para visualizar los resultados, también se pueden extraer métricas cuantitativas de los resultados de la simulación

utilizando lenguajes como PERL [117] o AWK para filtrar los ficheros que contienen las trazas mediante un post-procesado de los mismos.

AWK, hace referencia a los nombres de sus autores es un lenguaje de programación diseñado para procesar datos basados en archivos de texto, su función básica es buscar en los archivos líneas o secuencias de caracteres que contengan ciertos patrones, cuando existe una coincidencia con uno de los patrones, AWK realiza acciones específicas. En resumen, este lenguaje es ampliamente utilizado para realizar análisis de gran cantidad de datos obtenidos como resultado de simulaciones, en el presente proyecto ha sido muy útil para extraer desde el archivo de trazas entregado por NS-2, información de rendimiento, retardo, Jitter, así como paquetes enviados, recibidos, perdidos y la tasa de entrega de paquetes, entre otras [118].

En la Tabla 3 se observan los modelos y tecnologías más importantes implementadas en esta versión de NS-2.

4.4. Simulaciones

Los ensayos que se presentan en este capítulo, corresponden a una secuencia de simulaciones que fueron acompañando la curva de aprendizajes sobre el tema a lo largo de los últimos años, desarrollados dentro del marco de los proyectos de investigación de la UNPA.

Todas las simulaciones de este trabajo se realizaron con el simulador NS-2 en su versión ns-2.35 released Nov 4 2011 [119]. Para implementar los modelos se utilizó el lenguaje OTcl [120]. El procesamiento de datos se hizo mediante scripts de awk, perl y python. Para la presentación visual se utilizó gnuplot. En la siguiente figura (Figura 35) se observan las diferentes etapas de la simulación.

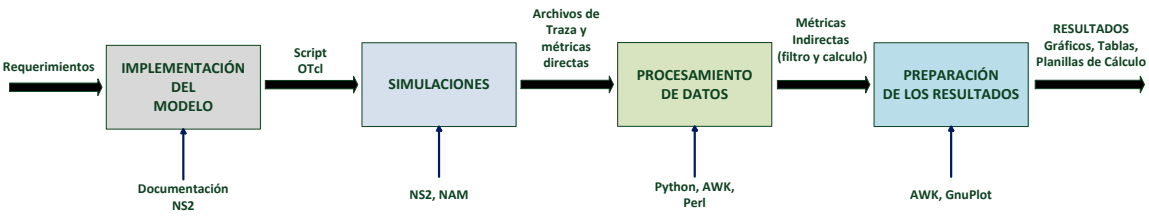

Figura 35: Proceso de la simulación

El proceso de simulación (Figura 35) consiste en la implementación de un modelo a simular, la generación de pruebas, el procesamiento de los datos obtenidos y la presentación de los mismos.

Para el estudio de rendimiento de TCP sobre diferentes topologías, se crearon scripts en Otcl para simular redes cableadas, redes móviles y redes híbridas en NS-2 [121]. En ellos se podían ajustar variables que definían tipo de tráfico, cantidad de nodos, formato de los archivos de salida, etc.

Algunos de los parámetros útiles para el análisis de rendimiento podían obtenerse directamente del simulador, al igual que los archivos de traza. Entre ellos se encuentran la Ventana de congestión, Round Trip Time y número de secuencia. Análogamente, se utilizaron scripts en AWK y Perl para filtrar y calcular: throughput, goodput, delay, packet loss ratio, packet delivery ratio, y jitter. Los lenguajes AWK y Perl son lenguajes de programación que se utilizaron para procesar archivos de texto. Luego se utilizó Gnuplot lo que permite una representación gráfica de los datos procesados. Gnuplot es un software para trazar gráficos 2D y 3D a partir de datos y funciones. Gnuplot admite muchos formatos de salida, puede producir resultados directamente en la pantalla, o en muchos formatos de archivos de gráficos, incluyendo Portable Network Graphics (PNG), PostScript encapsulado (EPS), Graphics Vector Graphics (SVG), JPEG y muchos otros.

Con respecto a los modelos que se utilizaron, se implementaron modelos sencillos de manera de poder controlar cada uno de los efectos por separado y queden en evidencia en forma más simple, los efectos de lo que se propuso estudiar en cada uno. Los modelos emulan una red heterogénea con acceso inalámbrico.

El TCP Vegas, como se verá más adelante es un tema distinto. Dado su control de congestión proactivo se observará que es incapaz de conseguir una parte justa de los recursos cuando compite con controles de congestión reactivos, más agresivos. Por esta razón, se analizó en la mayoría de los casos a la variante Vegas en un modelo de 3 nodos y con un solo flujo para estudiar su comportamiento.

En cuanto a las simulaciones que se analizan a continuación, la primera parte estudia los efectos de las desconexiones de distinta duración, en un modelo simple, para distintas variantes de control de congestión. La segunda tanda de simulaciones, corresponden a un modelo sencillo, de solo 3 nodos y un solo flujo TCP que permitió estudiar el comportamiento de distintas variantes de control de congestión sin interferencias de otros flujos ni nodos, sometiéndolo a las pérdidas de paquetes en forma de ráfagas de distintas longitudes, y así probar su efecto en el rendimiento. Ante lo observado en los ensayos anteriores, se realizó una tercera tanda de simulaciones para explorar el comportamiento del TCP Vegas, y sus parámetros característicos α (alfa) y β (beta). Resulta interesante el análisis de esta variante dado que posee un algoritmo de control de congestión proactivo.

Por último, para analizar la contienda de las variantes de TCP, se utilizó un modelo de hasta 17 nodos, lo que permitió estudiar la interrelación de dos o más flujos. En el último ensayo este modelo se lleva a la contienda de más de 8 flujos TCP compartiendo y compitiendo por parte de los recursos del camino común de la red.

Resulta interesante destacar que, como resultado adicional del análisis y el estudio de las simulaciones realizadas en estos temas de investigación, se utilizaron parcialmente para la publicación y exposición de diferentes artículos en congresos como el CACIC, CACIDI e INCISCOS, y la publicación de artículos en JCS&T y CCIS de Springer. Un detalle de los mismos, se puede observar en el Anexo B - Contribuciones.

4.5. Métricas

Tanto la Comisión de Estudio 12 (CE12) [122] del UIT-T (Unión Internacional de Telecomunicaciones UIT) [123] como el Grupo de trabajo de métricas de rendimiento de IP (IPPM - IP Performance Measurement) [124] del IETF (Internet Engineering Task Force) [2] han producido definiciones de métricas estándar, aunque cada una con un énfasis diferente, estrechamente relacionado con los antecedentes históricos de ambas organizaciones. La UIT tiene sus orígenes en la telefonía, mientras que el IETF tiene una formación en redes de datos. La UIT enfatiza la evaluación de un servicio y su calidad, el

IETF mide la red y desea proporcionar a la comunidad una comprensión y medición precisas del rendimiento y la fiabilidad de Internet.

Las métricas del UIT-T buscan proporcionar un lenguaje común para que los proveedores se comuniquen sobre el rendimiento, por lo que las métricas del UIT-T no se concentran en el rendimiento dentro de una sola red. Mientras que el IETF se centra en los protocolos de medición del rendimiento y la implementación.

Debido a sus antecedentes respectivos, la UIT generalmente produce métricas estadísticas orientadas a una representación cuantitativa de la experiencia completa del usuario de extremo a extremo, mientras que el grupo de trabajo IPPM de IETF se centra principalmente en métricas estadísticas que proporcionan una vista técnica detallada de diferentes aspectos de la calidad de transmisión, a lo largo de la ruta de la red.

Las recomendaciones UIT-T abarcan las terminales, redes y servicios desde las redes de conmutación de circuitos hasta las redes inalámbricas. Las normas de la CE12 están destinadas a lograr los niveles de funcionamiento extremo a extremo requeridos para propiciar una calidad de servicio adecuada en un entorno IP caracterizado por una gran diversidad de aplicaciones de usuario.

El trabajo de normalización de la CE12 se orienta a los aspectos prácticos de la calidad de funcionamiento, la calidad de servicio y la calidad percibida. Presta especial atención a la calidad de extremo a extremo, tal y como es percibida por el usuario.

Uno de los éxitos recientes de la CE12 fue la finalización de una norma que trata sobre la calidad de servicio en las redes móviles (Recomendación UIT-T E.804), un método que aborda los parámetros de la calidad de servicio desde el punto de vista del usuario. En la norma se definen parámetros de calidad de servicio y el cálculo de los mismos para servicios populares de las redes móviles, como son el correo o la transmisión de vídeo y voz; describe asimismo los procedimientos de medición necesarios para llevar a cabo la medición de los parámetros de calidad de servicio.

Por otra parte, el objetivo del IPPM WG es desarrollar un conjunto de métricas estándar que se puedan aplicar a la calidad, el rendimiento y la confiabilidad de los servicios de entrega de datos de Internet y las aplicaciones que se ejecutan sobre los protocolos de la capa de transporte (por ejemplo, TCP, UDP) sobre IP. También desarrolla y mantiene metodologías y protocolos para la medición de estas métricas.

Estas métricas se diseñan de manera tal que puedan ser realizadas por operadores de red, usuarios finales o grupos de prueba independientes. Es importante que las métricas no representen un juicio de valor, sino que proporcionen medidas cuantitativas imparciales de desempeño.

En la siguiente tabla (Tabla 4) se observan algunos de los RFC del grupo de trabajo IPPM de IETF y las recomendaciones UIT-T equivalentes.

	IETF RFC (IPPM)	ITU-T Recommendations (SG12)
Framework	RFC2330	Y.1540, sections 1 thru 5
Loss	RFC2680	Y.1540, section 5.5.6 G.1020
Delay	RFC2679 (One-way) RFC2681 (Round Trip)	Y.1540, section 6.2 G.1020G.114 (one-way)
Delay Variation	RFC3393	Y.1540, section 6.2.2 G.1020
Connectivity / Availability	RFC2678	Y.1540, section 7
Loss Patterns	RFC3357	G.1020
Packet Reordering / Packet Duplication	RFC4737	Y.1540 sections 5.5.8.1 and 6.6 Y.1540 sections 5.5.8.3, 5.5.8.4, 6.8 and 6.9
Link/Path Bandwidth Capacity - Link Utilization - Available Capacity	RFC5136	
Bulk Transport Capacity	RFC3148 - RFC5136	

Tabla 4: IETF RFC y UIT-T Recomendaciones

En la Tabla 4, se puede observar dentro de las Recomendaciones de la UIT-T, la Y.1540: "Internet protocol data communication service - IP packet transfer and availability performance parameters" [125] y G.1020 (07/06): "Performance parameter definitions for quality of speech and other voiceband applications utilizing IP networks" [126].

Además, en la misma tabla (Tabla 4), se puede observar que IPPM ha definido un marco para las métricas de rendimiento de la red en RFC2330 "Framework for IP Performance Metrics" [127]. También desarrolló definiciones para varias métricas de rendimiento específicas, tales como RFC2680: "A One-way Packet

Loss Metric for IPPM" [128], [129] para pérdidas de paquetes, RFC2679: "A One-way Delay Metric for IPPM" [130] y RFC2681: "A Round-Trip Delay Metric for IPPM" [131] para latencia, RFC3393: "IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)" [132] para la variación de la latencia. En la Tabla 4, se destancan RFC2678: "IPPM Metrics for Measuring Connectivity" [133], RFC3357: "One-way Loss Pattern Sample Metrics" [134], RFC4737: "Packet Reordering Metrics" [135], RFC5136: "Defining Network Capacity" [136] y RFC3148: "A Framework for Defining Empirical Bulk Transfer Capacity Metrics" [137].

De acuerdo a lo establecido anteriormente, las definiciones del Grupo de Trabajo IPPM de IETF se ajustan mejor que las de la Comisión de Estudio 12 de UIT-T, para la realización del presente trabajo. Sin embargo, no fueron adoptadas en su totalidad, en algunos casos porque no fue posible, mientras que en otros se consideró que se podría analizar más claramente los resultados de otra forma [138].

Para evaluar y comparar los rendimientos [139] de las distintas variantes del protocolo TCP en los escenarios y condiciones propuestas, se obtuvo una serie de métricas para cada una de las simulaciones [140]. A continuación, se definen dichas métricas y se describe la forma en que se obtuvieron a partir de la simulación [141]. Para mayor detalle, en Anexo A - Scripts se incluye parte del código utilizado.

4.5.1. Throughput

IPPM IEFT define el *Throughput TCP* en RFC6349 [142]. En este documento se establece una metodología para probar el rendimiento de la capa de transporte TCP. También define al throughput como la cantidad de datos por unidad de tiempo que transporta TCP en estado de equilibrio. Para el cálculo del rendimiento del throughput de TCP, utiliza un conjunto de variables tales como el ancho de banda del cuello de botella (Bottleneck Bandwidth), el Round Trip Time (RTT) y el path MTU (Maximum Transmission Unit) entre otros. Para obtener el Path MTU, el método de la búsqueda que sigue es el de RFC4821 [143]. Para la búsqueda del cuello de botella, puede realizarse una prueba de Capa 2/3 del RFC2544 [144].

Sin embargo, para la realización del presente trabajo se definió el *throughput* o *throughput instantáneo* como la medida de la cantidad de datos enviados por el emisor por unidad de tiempo, medida en bits por segundo.

La selección de esta forma de obtener la métrica obedece a que se consideró que, dada la particularidad de los escenarios bajo estudio, resultan más evidentes las acciones del control de congestión de las distintas variantes ensayadas, pues refleja de manera directa la forma en que el emisor TCP inyecta paquetes en la red, sin contradecir lo propuesto por la RFC.

El cálculo de esta métrica se realiza mediante un script escrito en AWK (Anexo A - Scripts), aplicado sobre el archivo de traza generado durante la simulación. El script diferencia el formato de la traza según el tipo de tráfico utilizado (cableado, inalámbrico o híbrido) y asigna variables a los campos que indican el tipo de evento, tiempo, id del flujo, protocolo utilizado, origen, destino, etc.

4.5.2. Throughput Promedio

Throughput Promedio se define como la totalidad de bits enviados por el emisor dividido por el tiempo total que se necesita para que todos los datos sean enviados y reconocidos. Las unidades de medida están expresadas en bits por segundo (bits/s)

$$\text{Thoughput Promedio} = \frac{\text{Cantidad total de bits enviados}}{\text{Tiempo Total de Transmisión}} \qquad \textbf{Ecuación 60}$$

El throughput promedio se obtiene mediante un script de AWK que calcula el promedio sobre los datos almacenados para el throughput instantáneo. Simplemente suma los valores y los divide entre la cantidad de entradas disponibles.

4.5.3. Goodput

Es una métrica relacionada, que puede ser considera como rendimiento efectivo, que excluye de los cálculos a los paquetes descartados, duplicados y todo dato no útil de la comunicación. La cantidad de datos considerados excluye los bits de los encabezados de los protocolos. Esto se relaciona con la cantidad de tiempo desde el primer bit del primer paquete enviado y entregado,

hasta que se entrega el último bit del último paquete recibido y reconocido. También, se lo puede pensar como el rendimiento de la capa de aplicación. Se calcula de la siguiente manera (Ecuación 61).

$$\text{Goodput} = \frac{\text{Cantidad Total de bits útiles (aplicación)}}{\text{Tiempo Total de Transmisión}} \qquad \textbf{Ecuación 61}$$

Para el presente trabajo, el cálculo realizado para el goodput es muy similar al del throughput, con la salvedad de que se utilizan los bits transmitidos con éxito únicamente. En este caso es posible utilizar valores contenidos en variables de estado de la simulación (Anexo A - Scripts), consiste en aprovechar la existencia de una variable global de la simulación que registra los bytes recibidos por agentes de tipo sink (sumidero) para tráfico TCP.

Si bien esta métrica se calculó en todas las simulaciones, por la forma en que se implementó, no resultó información relevante para el análisis de los distintos comportamientos.

4.5.4. Tiempo total de transmisión

El tiempo total de transmisión, es el tiempo que requiere un flujo determinado, para trasmitir una determinada cantidad de datos. De esta manera, es el tiempo medido desde que se comienza a trasmitir el primer paquete hasta que se recibe el ACK del último.

Esta métrica se obtuvo mediante un script de AWK (Anexo A - Scripts), a partir de los archivos de traza de la simulación. Para cada flujo se registran las entradas correspondientes al primer y último paquete transmitido y, por lo tanto, la diferencia de tiempo entre ellos constituirá el intervalo de transmisión para dicho flujo.

4.5.5. Latencia (End to End Delay)

El grupo de trabajo IPPM del IETF define en RFC2679 la definición de la métrica One Way delay u OWD. La latencia de un paquete o trama unidireccional para un servicio, lo define como el intervalo entre el momento en que el primer bit del paquete ingresa a la interfaz de envío y el momento en que el último bit del paquete correspondiente llega a la interfaz de recepción. El

atributo puede ser aplicado solo a los paquetes que se entregan a la interfaz receptora, lo que significa que los paquetes perdidos se excluyen de la definición. Dado que los diferentes paquetes enviados generalmente experimentan diferentes tipos de retraso, el atributo puede especificarse utilizando un valor estadístico como el retraso medio del paquete. Dado que el retardo de propagación es un componente del retardo de paquetes, el atributo puede especificar diferentes valores para diferentes pares de interfaces de envío y recepción. La demora se utiliza para medir el tiempo esperado para que un paquete IP atraviese la red de un host a otro. El retraso se aplica a la calidad del servicio para los protocolos sensibles a la latencia.

Para el presente trabajo, se definió la latencia como el intervalo de tiempo requerido entre la generación de los datos en el emisor y la recepción completa del segmento en el receptor. De esta forma, incluye los tiempos de propagación, de procesamiento en los distintos nodos y los tiempos de cola.

Para obtener esta métrica se utilizó un script de AWK (Anexo A - Scripts), a partir de los archivos de traza obtenidos de la simulación. Al recorrer el archivo de traza, se registran los tiempos correspondientes a la transmisión y recepción de cada paquete, y se almacenan. Luego de registrar todos los valores, se guardan las diferencias de tiempo de cada paquete.

La latencia inducida por la red a menudo tiene un impacto notable en el rendimiento. Al estudiar el rendimiento de extremo a extremo, generalmente resulta más interesante estudiar las métricas que se derivan de esta, tal como el Tiempo de ida y vuelta (RTT) y la Variación de la latencia o Jitter. Las aplicaciones interactivas en tiempo real, como la telefonía por Internet, es decir, VoIP (Voz sobre IP), videoconferencia, entornos VR (Realidad Virtual) y juegos de red multijugador son muy sensibles a los retrasos y aplican restricciones estrictas en la transferencia de datos en el tiempo. Los altos retrasos destruyen gravemente el rendimiento de estas aplicaciones. Si bien en el presente trabajo solo se utilizó tráfico FTP en las simulaciones, esta métrica resulto interesante como medida indirecta del estado de las colas y del grado de congestión de los nodos intermedios.

4.5.6. Latencia Promedio (Average Delay)

Es el promedio del valor métrica anterior (Latencia) de todos los paquetes de la simulación. De esta manera, el retardo extremo a extremo promedio es la sumatoria de todos los tiempos dividido por la cantidad de paquetes entregados exitosamente.

La métrica se obtuvo mediante un script de AWK, a partir de los archivos de Latencia, en forma análoga a las métricas de throughput promedio y goodput promedio.

4.5.7. Jitter

El grupo de trabajo IPPM de la IETF, define la métrica IP Packet Delay Variation (IPDV) o Jitter en la RFC3393. Esta métrica en particular solo compara los retrasos experimentados por paquetes de igual tamaño, ya que el retraso depende naturalmente del tamaño del paquete. IPDV se presenta como la diferencia OWD entre paquetes consecutivos de una secuencia de paquetes. De esta manera, se deriva de la métrica de retardo unidireccional.

El Jitter es la variación de la latencia entre las llegadas de paquetes subsiguientes. Es la dispersión del delay extremo a extremo o latencia. Es una medida de la variabilidad de la latencia de una comunicación. El valor de latencia u OWD no es constante y la diferencia se denomina "variación de retardo" o jitter. Puede ser causada por la contienda con otros flujos (es decir, la cola de espera), o por la contención de los recursos de procesamiento en la red y, en general, un gran valor de jitter es sintomático de una red muy cargada.

En la simulación, el jitter se calcula a partir de los valores obtenidos para la latencia. De forma secuencial, para cada par de entradas en archivo, se calcula el valor medio entre ellos y el resultado es almacenado.

Esta métrica se calculó en todas las simulaciones realizadas para el presente trabajo. Sin embargo, no fue de mayor utilidad para el análisis propuesto debido a que es un parámetro importante en las trasferencias de datos de tiempo real, voz y video, no así para los tráficos FTP utilizados.

4.5.8. Average Jitter

Es el promedio de la métrica Jitter para una simulación completa. Se calcula de forma análoga a los promedios anteriores, mediante un script de AWK que calcula el promedio sobre los datos contenidos en el archivo de valores de jitter.

De la misma manera que la métrica anterior, no tuvo mayor utilidad en el presente trabajo.

4.5.9. Packet Loos Ratio (PLR)

El grupo de trabajo IPPM define en RFC2680, la definición para la métrica One Way Packet Loss. La pérdida de paquetes se determina como la probabilidad de que un paquete se pierda en tránsito desde una fuente A un destino B. También relaciona la definición One-way-Packet-Loss con One-way-Delay. Además, el documento describe la necesidad de que la metodología de medición incluya una forma de distinguir entre una pérdida de paquetes y un retraso muy grande, pero finito. Al igual que con las mediciones de retraso, el tiempo umbral para considerar la pérdida de un paquete tardío debe informarse como parte de los resultados de la medición.

Para el presente trabajo, se define la métrica Packet Loss Ratio (PLR) como la tasa de pérdida de paquetes que se expresa como una fracción de los paquetes enviados, de acuerdo a la siguiente ecuación (Ecuación 62).

$$\mathrm{PLR} = \frac{\sum \text{de paquetes enviados} - \sum \text{de paquetes recibidos}}{\sum \text{de paquetes enviados}} \cdot 100 \qquad \textbf{Ecuación 62}$$

Es la relación entre el total de paquetes perdidos y el total de paquetes transmitidos. Es una métrica que se puede calcular para toda la red o para un flujo determinado y puede usar como una indicación directa de la eficiencia del entorno.

En las simulaciones, el valor de PLR se obtiene a partir del archivo de traza, mediante un script de AWK que registra, para cada flujo, la cantidad de paquetes enviados, recibidos y perdidos. Al final del mismo, se emplea la fórmula definida en la Ecuación 62.

4.5.10. Packet Delivery Ratio (PDR)

PDR es el porcentaje del total de paquetes enviados por el nodo emisor, que son recibidos con éxito por el nodo receptor. Es decir, es la proporción de entrega de paquetes recibidos por el receptor y la cantidad de paquetes que envía el emisor.

$$\text{PDR} = \frac{\sum \text{de paquetes recibidos}}{\sum \text{de paquetes enviados}} . 100 \qquad \textbf{Ecuación 63}$$

4.5.11. Ventana de Congestión (CWND)

La métrica es la variación del tamaño de la ventana de congestión en función del tiempo. Puede representarse en varias unidades, bytes, segmentos, paquetes. En el caso de este trabajo esta expresada en paquetes. La evolución del tamaño de la ventana de congestión afecta directamente el rendimiento de TCP y su evolución permite establecer el funcionamiento de los distintos mecanismos de control de congestión de cada una de las variantes.

En las simulaciones, el valor del tamaño de la ventana de congestión se obtiene a partir de una variable global que registra el valor de la ventana de congestión en cada instante, para un determinado flujo TCP. Se utiliza una función que se ejecuta periódicamente (Anexo A - Scripts) para comprobar el valor de la variable y almacenarla en un archivo.

Es una de las métricas más utilizadas en el presente trabajo, pues permite ver en los gráficos de su valor en función del tiempo como trabajan las distintas variantes de control de congestión, ajustando la tasa de envío a determinados estímulos.

4.5.12. Número de Secuencia

Es el valor del número de secuencia que se encuentra en el encabezado del protocolo TCP. Se grafica su evolución en función del tiempo de simulación.

En las simulaciones, el número de secuencia se obtiene a partir de una variable global de la simulación que registra el número de secuencia siendo utilizado en cada instante de la transmisión, para un determinado flujo. Se

utiliza una función que se ejecuta periódicamente para comprobar el valor de la variable y almacenarla en un archivo. Para simplificar los gráficos, se le resta el primer valor de manera que el primer número de secuencia siempre sea 1.

Es una métrica que también se utilizó en forma reiterada en forma de gráficos, porque permite observar claramente la tasa de envío del emisor, y como se recupera esa tasa después de algún evento que dispare el control de congestión.

4.5.13. RTT

El grupo de trabajo IPPM define en RFC2681, Round Trip Delay (RTD) para un solo datagrama transmitido y, en base a eso, define cómo se puede extender esta métrica para producir una serie de valores de retraso a un flujo distribuido de Poisson de paquetes enviados. En el documento, la IETF también evalúa los posibles errores de medición y enfatiza la necesidad de tener en cuenta el nivel de incertidumbre junto con los resultados. En las definiciones de IETF, RTD denota solo el componente del tiempo de propagación del tiempo total requerido para enviar un paquete de ida y vuelta. En contraste, el tiempo completo de ida y vuelta (RTT) también incluye el retardo del sistema final.

Sin embargo, para este trabajo se utilizó el valor de RTT Round Trip Time, que hace referencia al tiempo que tarda un paquete en salir del nodo emisor, llegar al nodo destino y recibir el correspondiente reconocimiento ACK en forma correcta. Es decir, es la suma de los retrasos unidireccionales de emisor a receptor y de receptor a emisor, y del tiempo que tarda el receptor en procesar la respuesta al paquete original. El valor de RTT no solo influye en el rendimiento alcanzable, ya que solo puede haber un valor de datos no reconocidos de una ventana en la red, sino también, lo utilizan algunas variantes de control de congestión para realizar estimaciones.

En las simulaciones, el valor de RTT se obtiene a partir de una variable global de la simulación que registra el valor de RTT en cada instante de la transmisión, para un determinado flujo. Se utiliza una función que se ejecuta periódicamente para comprobar el valor de la variable y almacenarla en un archivo (Anexo A - Scripts).

4.6. Simulaciones en NS-2

A partir de las variantes de TCP analizadas en este trabajo, y considerando los resultados preliminares de sus simulaciones, solo se incluyen a continuación aquellas que se consideran que sus resultados son significativos para la tesis, dada las características del ambiente y condiciones que se establecieron para los ensayos. Asimismo, se consideraron las variantes más utilizadas tales como Compound, CUBIC, New Reno, además de TCP Westwood que fue desarrollada para los ambientes con enlaces inalámbricos y TCP Reno como respuesta de referencia. También se incluye TCP Vegas como un caso particular, tanto por el mecanismo de ajuste de cwnd, como por sus respuestas en los escenarios analizados [15].

4.6.1. Desconexiones de distinta duración en un escenario simple

En una conexión que incluye enlaces inalámbricos y por cable, el enlace inalámbrico sufre frecuentemente desconexiones debido a la movilidad, crisis de energía o variaciones en el entorno, como se ha descripto anteriormente. Este tipo de desconexiones puede tener distinto tiempo de duración, aunque suele durar más que un RTO y es más corto que la vida útil de una conexión TCP. Un problema grave causado por esta desconexión temporal es que la caducidad de los temporizadores de retransmisión (timeout) en serie en el emisor que produce múltiples y consecutivas retransmisiones de un mismo paquete que intenta enviar al destino que esta desconectado.

Es por esta razón que resulta interesante analizar el comportamiento de una red WLAN de acceso ante la presencia de desconexiones de diferentes tiempos de duración. Se simula un modelo sencillo de solo 3 nodos y un solo flujo para poder visualizar el efecto en el rendimiento que producen estas desconexiones. Al ser un modelo sencillo y con la presencia de un solo flujo, se puede aislar el efecto de las desconexiones de las otras causas de perdida de rendimiento de TCP, lo que permite analizar cómo se comportan los distintos mecanismos de control de congestión de las diferentes variantes de TCP.

Para explorar el efecto de las desconexiones se utiliza el siguiente modelo simple (Figura 36). El modelo de la siguiente figura representa un escenario

heterogéneo como una red de acceso inalámbrica. Se implementó en el simulador de redes de eventos discretos (NS-2, Network Simulator 2), en su versión 2.35 (released nov. 4 2011). La selección de este modelo es una aproximación a un escenario Inalámbrico con un nodo fijo (nodo 0), una estación base (nodo 1) y un nodo móvil (nodo 2), con la simplificación práctica que el enlace inalámbrico no presenta errores y solo tiene desconexiones de distinta duración.

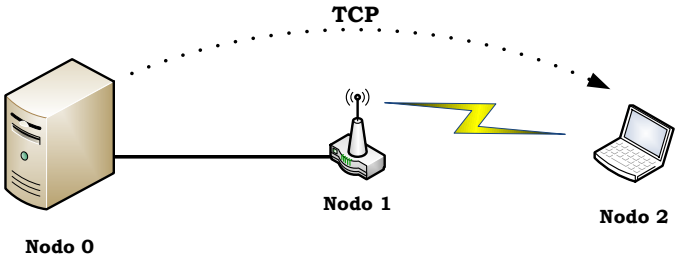

Figura 36: Modelo simple, flujo TCP y desconexiones

Los nodos 0 y 1 están vinculados por un enlace cableado que se configuró como dúplex, con un ancho de banda 10 Mb/s, retardo de propagación 2 ms. y política de servicio de las colas DropTail. El enlace entre los nodos 1 y 2 es inalámbrico y se configuró como modo de propagación TwoRayGround, la capa física WirelessPhy, MAC 802.11, la antena OmniAntenna, 2Mb/s con MAC 802.11. El nodo inalámbrico no posee movimiento.

El flujo TCP se define para cada una de las simulaciones entre el nodo 0 (emisor) y el nodo 2 (receptor). El flujo comienza a transmitir y las desconexiones comienzan a partir del paquete número 1000. En cada simulación se transmitieron 3.000 paquetes de TCP, de 1.000 Bytes cada uno. Las simulaciones fueron realizadas en forma independiente para cada variante del protocolo y para cada una de los tiempos de desconexión. Las desconexiones ensayadas tuvieron una duración de 0,05, 0,1, 0,5, 1, 2, 5, 10 y 20 segundos.

Resultados

Para poder analizar los efectos de las desconexiones de distinta duración sobre variantes de TCP a continuación, se presentan distintas métricas tales como Throughput vs. Tiempo, Throughput Promedio y el Tiempo Total de Transmisión en función de la duración de la desconexión, la evolución del

tamaño de la ventana de congestión y del número de secuencia del segmento TCP en función del tiempo.

THROUGHPUT

En las siguientes figuras (Figura 37 a Figura 66) se observa el valor del throughput en función del tiempo de la simulación, para los ensayos de desconexiones con distinto tiempo de duración para las distintas variantes de TCP.

En todos los casos, la secuencia se presenta para realizar una observación cualitativa, dado que difieren en la escala del eje de tiempos. Esto se debe a que el tiempo total de transmisión para esos datos, se incrementa a medida que aumenta la duración de la desconexión.

Las figuras a continuación (Figura 37 a Figura 42) representan el throughput instantáneo de *TCP Reno* para los ensayos con desconexiones de 0,05, 0,5, 1, 2, 5 y 10 segundos de duración.

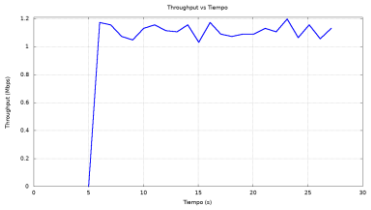

Figura 37: Throughput vs Tiempo TCP Reno – Desconexión 0,05s

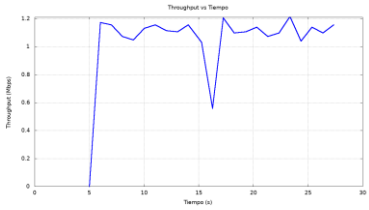

Figura 38: Throughput vs Tiempo TCP Reno - Desconexión 0,5s

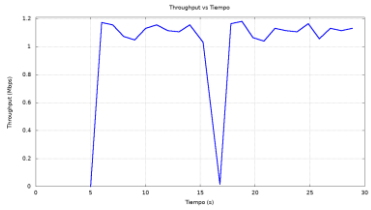

Figura 39: Throughput vs Tiempo TCP Reno - Desconexión 1s

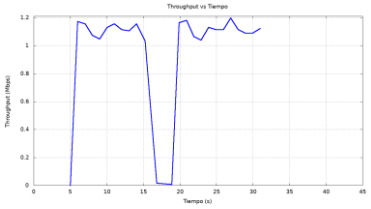

Figura 40: Throughput vs Tiempo TCP Reno - Desconexión 2s

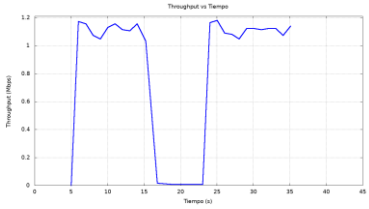

Figura 41: Throughput vs Tiempo TCP Reno - Desconexión 5s

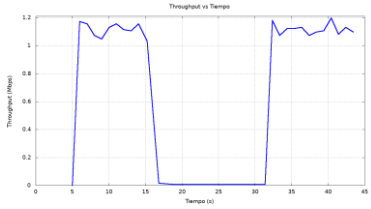

Figura 42: Throughput vs Tiempo TCP Reno - Desconexión 10s

En la anterior serie de gráficos, se puede ver el comportamiento del throughput a medida que transcurre el tiempo de la simulación. Cada una de las figuras representa el valor instantáneo para cada una de los tiempos de desconexión ensayados. Se puede observar que el rendimiento decae en mayor grado a medida que la duración del corte se incrementa.

Las siguientes figuras (Figura 43 a Figura 48) representan el throughput instantáneo de la variante del protocolo *TCP CUBIC* para desconexiones de distinta duración. En las figuras se puede observar un comportamiento similar al Reno. Se observa como el aumento del tiempo de desconexión degrada el uso del ancho de banda.

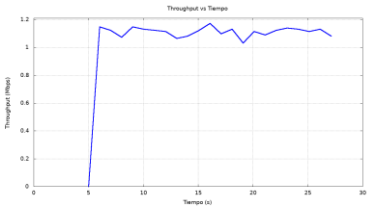

Figura 43: Throughput vs Tiempo TCP CUBIC – Desconexión 0,05s

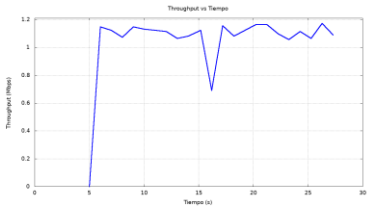

Figura 44: Throughput vs Tiempo TCP CUBIC – Desconexión 0,5s

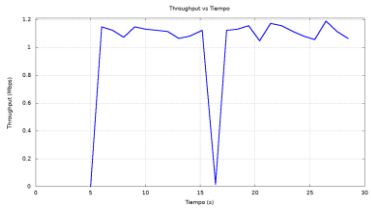

Figura 45: Throughput vs Tiempo TCP CUBIC – Desconexión 1s

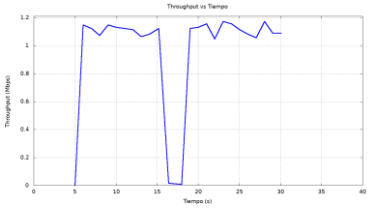

Figura 46: Throughput vs Tiempo TCP CUBIC – Desconexión 2s

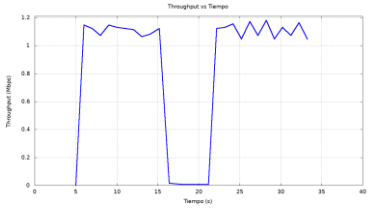

Figura 47: Throughput vs Tiempo TCP CUBIC – Desconexión 5s

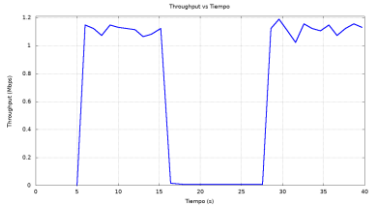

Figura 48: Throughput vs Tiempo TCP CUBIC – Desconexión 10s

En las siguientes figuras (Figura 49 a Figura 54) se puede observar el valor del throughput instantáneo para la variante de *TCP Westwood* en función del tiempo, para desconexiones de distintos valores de duración.

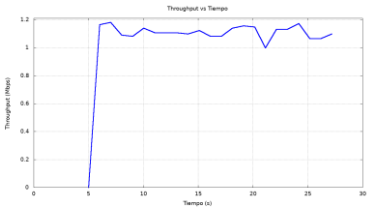

Figura 49: Throughput vs Tiempo TCP Westwood–Desconexión 0,05s

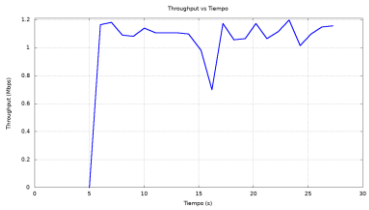

Figura 50: Throughput vs Tiempo TCP Westwood– Desconexión 0,5s

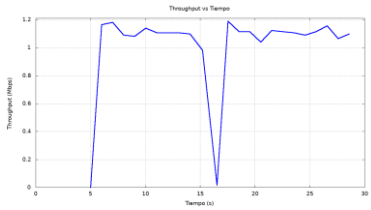

Figura 51: Throughput vs Tiempo TCP Westwood – Desconexión 1s

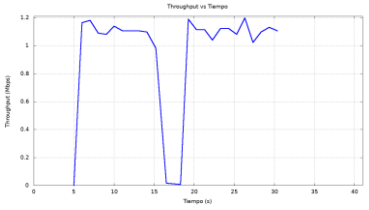

Figura 52: Throughput vs Tiempo TCP Westwood – Desconexión 2s

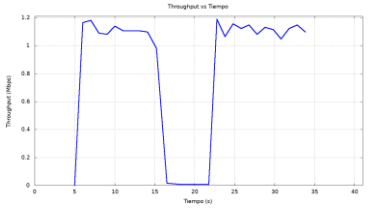

Figura 53: Throughput vs Tiempo TCP Westwood – Desconexión 5s

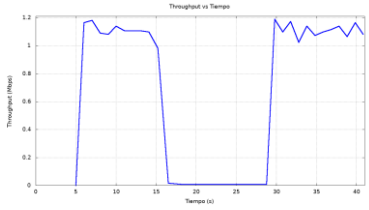

Figura 54: Throughput vs Tiempo TCP Westwood – Desconexión 10s

En la secuencia anterior se observa que el TCP Westwood tiene un comportamiento similar al TCP Reno para este modelo y estas condiciones. Sin

embargo, se puede observar que la reducción del valor del throughput comienza un instante antes.

La serie de gráficas a continuación (Figura 55 a Figura 60) representan el throughput vs tiempo para *TCP Vegas* para distintos tiempos de desconexiones.

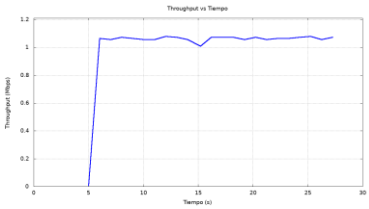

Figura 55: Throughput vs Tiempo TCP Vegas – Desconexión 0,05s

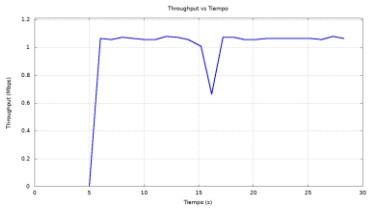

Figura 56: Throughput vs Tiempo TCP Vegas – Desconexión 0,5s

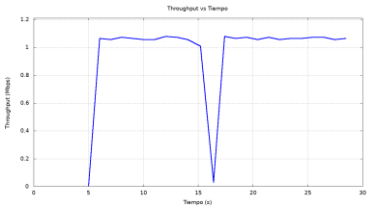

Figura 57: Throughput vs Tiempo TCP Vegas – Desconexión 1s

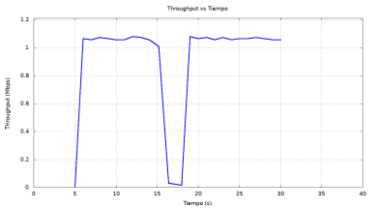

Figura 58: Throughput vs Tiempo TCP Vegas – Desconexión 2s

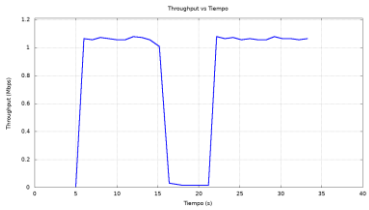

Figura 59: Throughput vs Tiempo TCP Vegas – Desconexión 5s

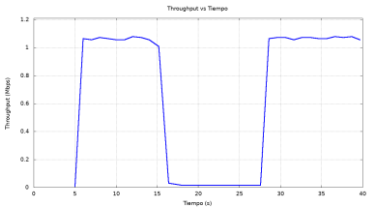

Figura 60: Throughput vs Tiempo TCP Vegas – Desconexión 10s

La serie de gráficos anteriores, muestra que esta variante empieza a mostrar secuelas en su rendimiento desde tiempos más cortos de desconexión. De las variantes analizadas es la más susceptible, desde el punto de vista del rendimiento, al efecto de las desconexiones.

A continuación (Figura 61 a Figura 66) la secuencia de valor del throughput instantáneo para la variante *TCP Compound*.

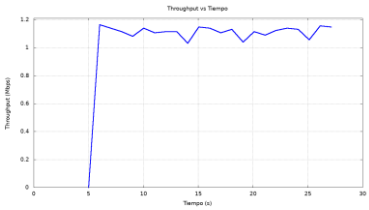

Figura 61: Throughput vs Tiempo TCP Compound–Desconexión 0,05s

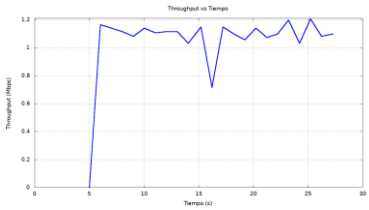

Figura 62: Throughput vs Tiempo TCP Compound–Desconexión 0,5s

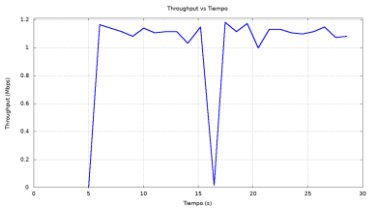

Figura 63: Throughput vs Tiempo TCP Compound – Desconexión 1s

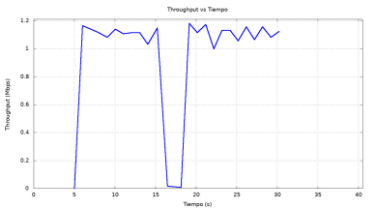

Figura 64: Throughput vs Tiempo TCP Compound – Desconexión 2s

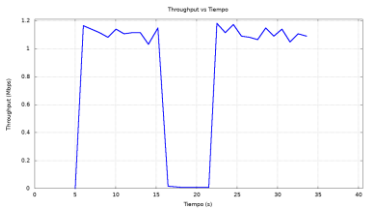

Figura 65: Throughput vs Tiempo TCP Compound – Desconexión 5s

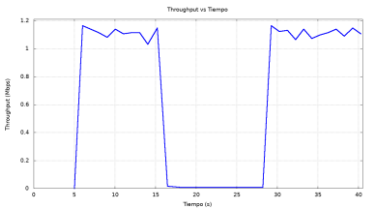

Figura 66: Throughput vs Tiempo TCP Compound – Desconexión 10s

En la serie de figuras anterior, se puede observar que el TCP Compound es el que tiene el comportamiento más parecido a TCP Reno en el modelo ensayado con un solo flujo TCP.

THROUGHPUT PROMEDIO

En la siguiente figura (Figura 67) se observa el valor del throughput promedio expresados en Mbit/s de cada una de las variantes de TCP simuladas, para cada uno de los tiempos de desconexión ensayados en estas pruebas. Se observa un patrón común para todas las variantes ensayadas: el throughput promedio disminuye a medida que el tiempo de desconexión aumenta. Más allá de alguna pequeña variación en la disminución relativa entre ellas, es siempre TCP Vegas el que tiene el menor valor de throughput promedio para este modelo. Observando las figuras anteriores de los throughput (Figura 55 a Figura 60) se puede ver las causas de este comportamiento de los valores promedios.

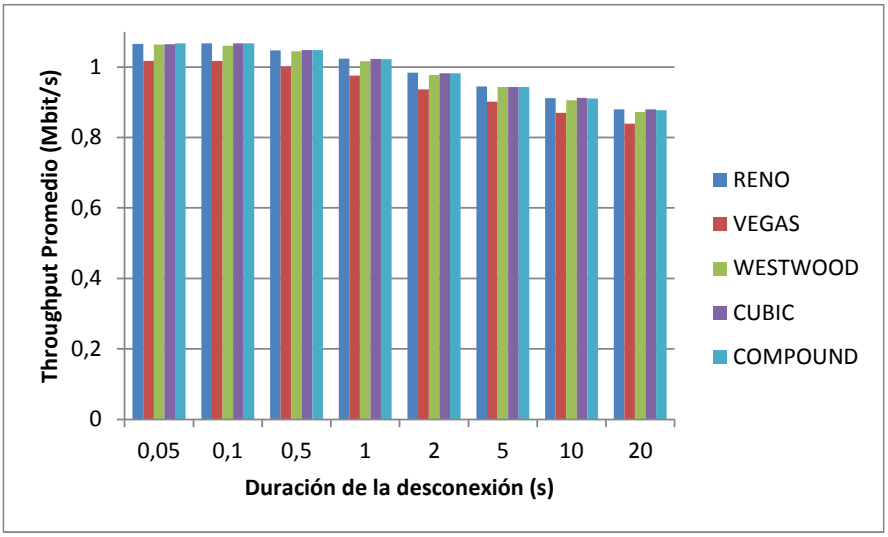

Figura 67: Throughput promedio vs duración de las desconexiones

TIEMPO TOTAL DE TRANSMISIÓN

Dado que las simulaciones se realizan transmitiendo siempre 3000 paquetes de 1000 bytes, la cantidad de información a trasmitir es constante. A partir de esto, resulta interesante analizar el tiempo que demora cada una de las variantes estudiadas en transmitir esa misma cantidad de datos. Esta medida

tiene directa relación con el throughput, lo que permitirá observar cómo afecta esas desconexiones a cada una de las variantes analizadas desde otra perspectiva.

En la siguiente figura (Figura 68) se observa el Tiempo Total de Transmisión en función del tiempo para cada una de las variantes ensayadas y para cada uno de los tiempos de desconexión. Se puede observar que a medida que el tiempo de desconexión crece, el tiempo necesario para trasmitir todos los datos también lo hace. Además, para desconexiones de baja duración el tiempo necesario para transmitir todos los paquetes es prácticamente del mismo orden, aunque el de Vegas es levemente superior. Sin embargo, a medida que aumenta el tiempo, empiezan a diferenciarse distintos valores de tiempo total, siendo Reno y Westwood quienes más tiempo necesitan para concluir la transmisión de los datos.

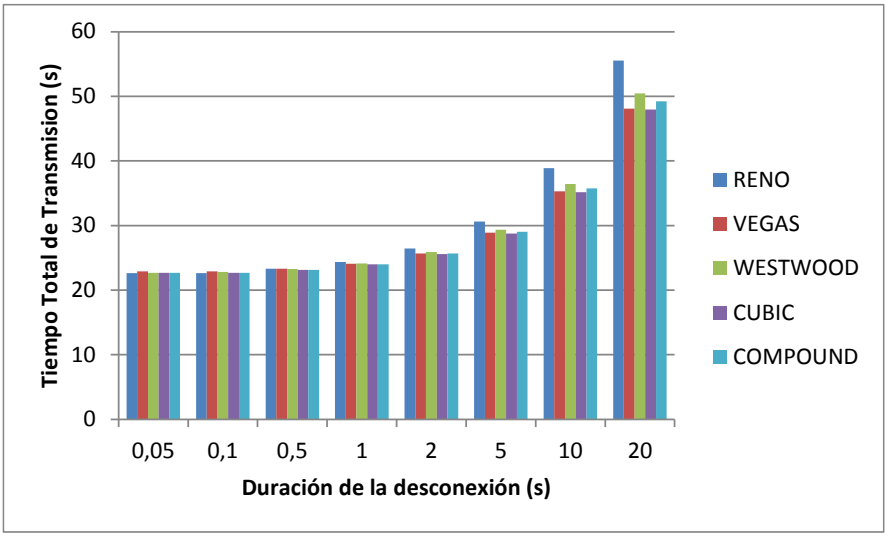

Figura 68: Tiempo total de transmisión vs duración de las desconexiones

Cabe resaltar que el tiempo necesario para transmitir todos los datos de la variante TCP Vegas es levemente mayor en los cortes de menor duración, aunque del mismo orden de magnitud para los valores medios. Para los cortes de mayor duración es junto con CUBIC, de las variantes que menos tiempo requieren para completar la transferencia en este modelo en particular y en estas condiciones.

NUMERO DE SECUENCIA

Como análisis complementario al tiempo total de transmisión, resulta interesante observar la evolución del número de secuencia del segmento TCP. Al ser un valor instantáneo permite observar la dinámica de la transmisión.

A continuación, se observan la secuencia de figuras (Figura 69 a Figura 80) que representan la evolución del número de secuencia del segmento TCP en función del tiempo de simulación. La serie muestra a las variantes Reno, Vegas y Westwood para desconexiones de 0.05, 1, 5 y 20 segundos de duración. En todos los casos, la secuencia se presenta para realizar una observación cualitativa, dado que difieren en la escala del eje de tiempos. Esto se debe a que el tiempo total de transmisión para esos datos, se incrementa a medida que aumenta la duración de la desconexión. Estas figuras permiten observar con mayor claridad el comportamiento de las variantes que se observaba en la Figura 68, tiempo total de transmisión vs tiempo de desconexión.

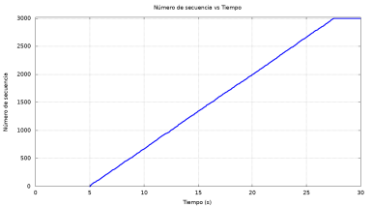

Figura 69: Nº de Sec. vs. Tiempo TCP Reno - Desconexión 0.05s

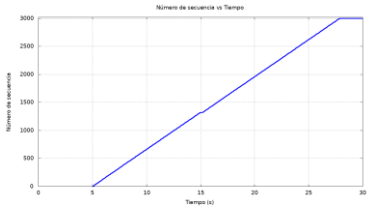

Figura 70: Nº de Sec. vs. Tiempo TCP Vegas - Desconexión 0,05s

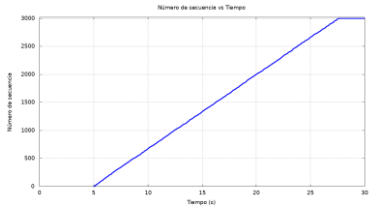

Figura 71: Nº de Sec. vs. Tiempo TCP Westwood-Desconexión 0.05s

Las figuras anteriores (Figura 69 a Figura 71) corresponden a las simulaciones con desconexiones de menor duración. Se puede observar que mientras que para los TCP Westwood y Reno no parece haber un punto de inflexión importante en la secuencia, si es apreciable en TCP Vegas (Figura 70). Esto se corresponde con lo visto en la Figura 68, para el tiempo de desconexión de 0,05 segundos.

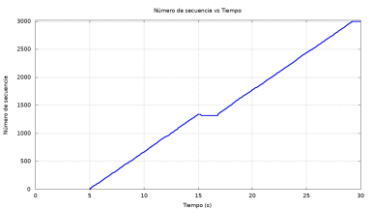

Figura 72: Nº de Sec. vs. Tiempo TCP Reno - Desconexión 1s

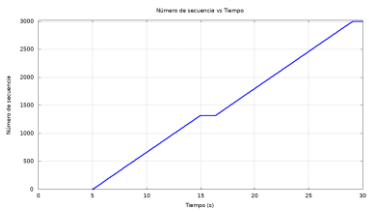

Figura 73: Nº de Sec. vs. Tiempo TCP Vegas - Desconexión 1s

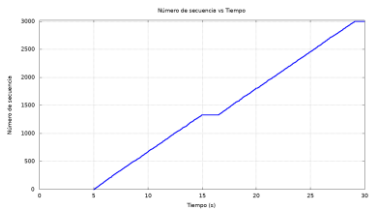

Figura 74: Nº de Sec. vs. Tiempo TCP Westwood - Desconexión 1s

En las figuras anteriores (Figura 72, Figura 73 y Figura 74) se puede observar que la desconexión de 1 segundo de duración afecta en forma similar a las 3 variantes. En la Figura 68 se observa este detalle donde los tiempos totales son similares.

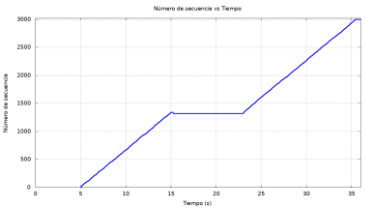

Figura 75: Nº de Sec. vs. Tiempo TCP Reno - Desconexión 5s

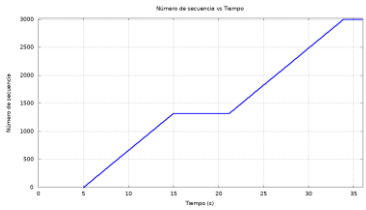

Figura 76: Nº de Sec. vs. Tiempo TCP Vegas - Desconexión 5s

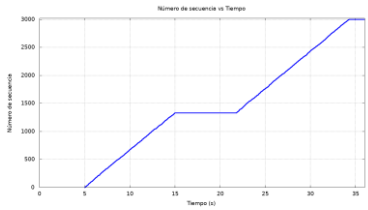

Figura 77: Nº de Sec. vs. Tiempo TCP Westwood - Desconexión 5s

En las figuras anteriores (Figura 75, Figura 76 y Figura 77) se comienza a observar cómo crece el tiempo en que el número de secuencia no avanza de la variante Reno (Figura 75) y, aunque en menor medida, Westwood (Figura 77), comparándolos con TCP Vegas (Figura 76). Esto es apreciable en la Figura 68 con el crecimiento de los tiempos totales de transmisión.

Esta tendencia se confirma y se acentúa en la siguiente serie de gráficos (Figura 78, Figura 79 y Figura 80) donde los tiempos en donde no avanza el número de secuencia de las variantes Reno y Westwood se hacen más largos.

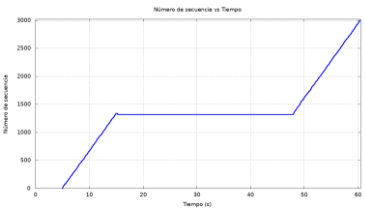

Figura 78: Nº de Sec. vs. Tiempo TCP Reno - Desconexión 20s

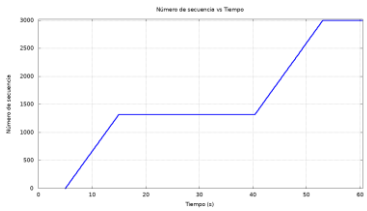

Figura 79: Nº de Sec. vs. Tiempo TCP Vegas - Desconexión 20s

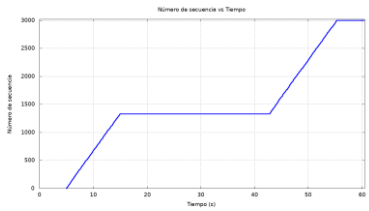

Figura 80: Nº de Sec. vs. Tiempo TCP Westwood - Desconexión 20s

En el caso de TCP Reno (Figura 78) el crecimiento se hace muy evidente, lo que repercute directamente en el tiempo total de transmisión (Figura 68).

VENTANA DE CONGESTIÓN

Una de las métricas de gran interés de análisis es la evolución del tamaño de la ventana de congestión en función del tiempo de la simulación, pues tiene directa relación con el rendimiento de TCP.

En las siguientes figuras (Figura 81 a Figura 110) se puede observar la evolución del tamaño de la ventana de congestión en función del tiempo. Cada una de las series se presenta para las distintas variantes de TCP. Estas series están compuestas una figura para cada uno de los valores de duración de las desconexiones, de manera de observar, en forma cualitativa, su variación en el tiempo. Debido a la introducción de las desconexiones de distintos tiempos de duración, las escalas de ambos ejes pueden variar y no permitir comparar sus valores. Sin embargo, permiten observar con mayor detalle el comportamiento de los controles de congestión de las distintas variantes ensayadas.

En la siguiente secuencia (Figura 81 a Figura 86) se observa la evolución de la ventana de congestión en función del tiempo para la variante *TCP Reno* y para desconexiones de 0.05, 0.5, 1, 2, 5 y 10 segundos.

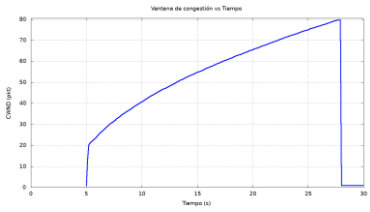

Figura 81: CWND vs. Tiempo
TCP Reno - Desconexión 0,05s

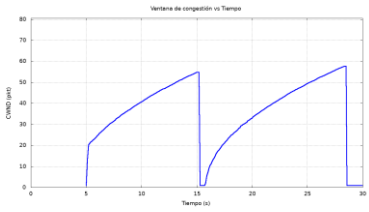

Figura 82: CWND vs. Tiempo
TCP Reno - Desconexión 0,5s

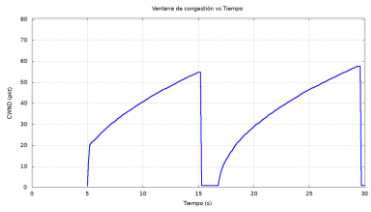

Figura 83: CWND vs. Tiempo
TCP Reno - Desconexión 1s

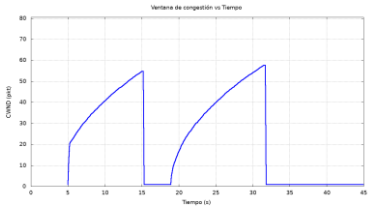

Figura 84: CWND vs. Tiempo
TCP Reno - Desconexión 2s

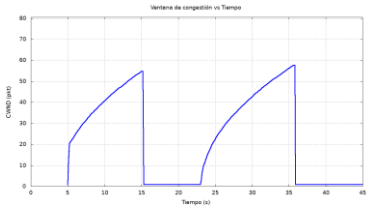

Figura 85: CWND vs. Tiempo
TCP Reno - Desconexión 5s

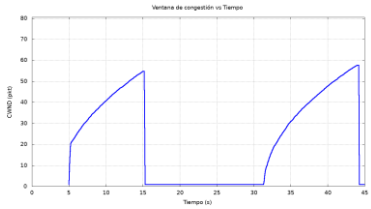

Figura 86: CWND vs. Tiempo
TCP Reno - Desconexión 10s

Se puede observar que el tamaño de la ventana que alcanza en la desconexión de 0,05 segundos de aproximadamente de 80 paquetes (Figura 81), valor que no puede volver a alcanzar en ninguna de las otras simulaciones de desconexiones de mayor tiempo. En los siguientes ensayos (Figura 82 a Figura 86) el valor máximo no llega a los 60 paquetes. El valor de la ventana desciende a 1 durante la desconexión.

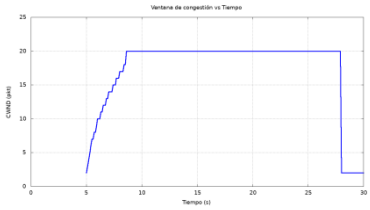

Figura 87: CWND vs. Tiempo TCP CUBIC - Desconexión 0,05s

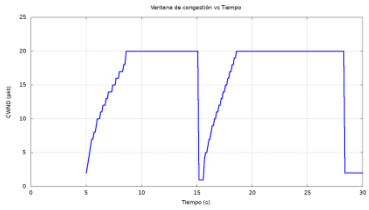

Figura 88: CWND vs. Tiempo TCP CUBIC - Desconexión 0,5s

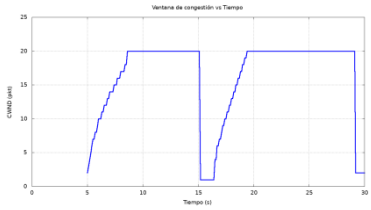

Figura 89: CWND vs. Tiempo TCP CUBIC - Desconexión 1s

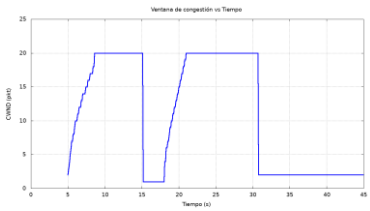

Figura 90: CWND vs. Tiempo TCP CUBIC - Desconexión 2s

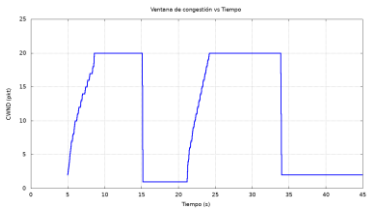

Figura 91: CWND vs. Tiempo TCP CUBIC - Desconexión 5s

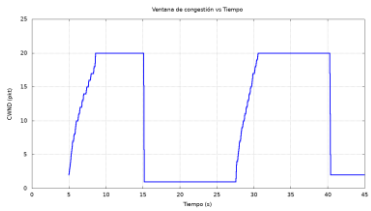

Figura 92: CWND vs. Tiempo TCP CUBIC - Desconexión 10s

En las figuras anteriores (Figura 92 a Figura 97) se observa la ventana de congestión vs el tiempo de la variante *TCP CUBIC*, para las desconexiones de distinto tiempo de duración. Se puede observar, en el crecimiento del tamaño de la ventana, la influencia de la función cúbica. En todos los cortes, la ventana de congestión se reduce a 1, excepto para el corte de 0,05 que no se observa que se modifique el tamaño de la ventana.

Las siguientes figuras (Figura 93 a Figura 98) muestran la evolución de la ventana de congestión de la variante *TCP Westwood*, para desconexiones de distintos tiempos de duración.

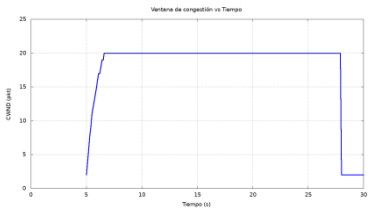

Figura 93: CWND vs. Tiempo TCP Westwood- Desconexión 0,05s

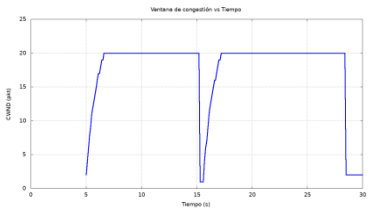

Figura 94: CWND vs. Tiempo TCP Westwood - Desconexión 0,5s

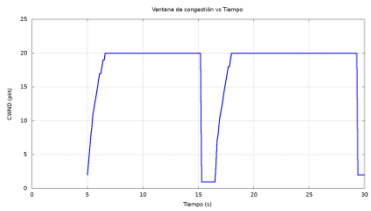

Figura 95: CWND vs. Tiempo TCP Westwood - Desconexión 1s

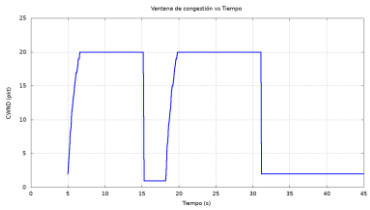

Figura 96: CWND vs. Tiempo TCP Westwood - Desconexión 2s

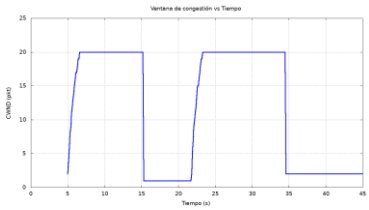

Figura 97: CWND vs. Tiempo TCP Westwood - Desconexión 5s

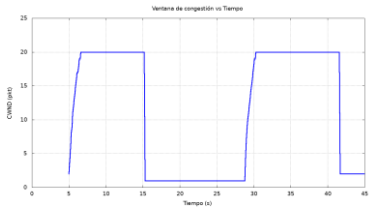

Figura 98: CWND vs. Tiempo TCP Westwood - Desconexión 10s

En la serie de figuras anterior se observa un comportamiento similar a las variantes anteriores, con la diferencia de la forma en la que crece la ventana. Así mismo, el tamaño máximo alcanzado es de 20 paquetes que se reduce a 1 en el periodo de desconexión.

En las siguientes figuras (Figura 99 a Figura 104) se observa la variación de la ventana de congestión en función del tiempo para *TCP Vegas*.

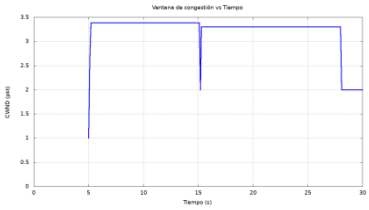

Figura 99: CWND vs. Tiempo TCP Vegas - Desconexión 0,05s

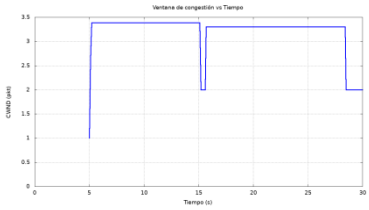

Figura 100: CWND vs. Tiempo TCP Vegas - Desconexión 0,5s

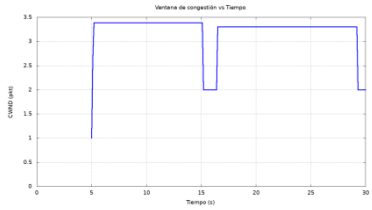

Figura 101: CWND vs. Tiempo TCP Vegas - Desconexión 1s

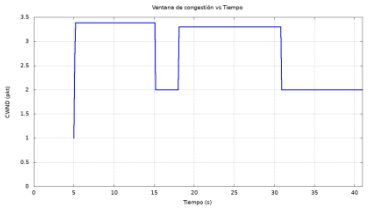

Figura 102: CWND vs. Tiempo TCP Vegas - Desconexión 2s

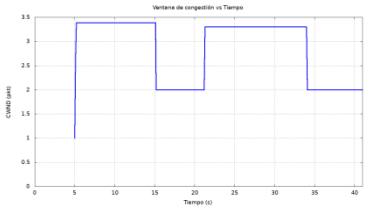

Figura 103: CWND vs. Tiempo TCP Vegas - Desconexión 5s

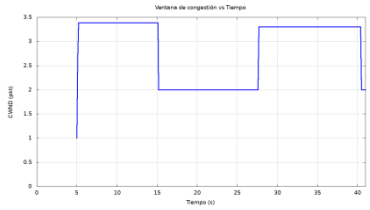

Figura 104: CWND vs. Tiempo TCP Vegas - Desconexión 10s

En las secuencias de figuras anteriores, se puede observar que el único control de congestión de los ensayados en este escenario, que toma alguna acción apreciable, con respecto al tamaño de la ventana de congestión para una desconexión de 0,05 segundos de duración es TCP Vegas (Figura 99). Reduce su tamaño por una fracción de tiempo. Sin embargo, el tamaño de la ventana de congestión es prácticamente constante y de valores pico similares para todos los cortes. El tamaño al que se reduce la ventana de congestión durante las desconexiones es dos paquetes.

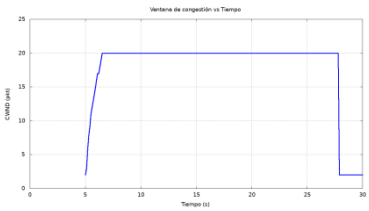

Figura 105: CWND vs. Tiempo TCP Compound-Desconexión 0,05s

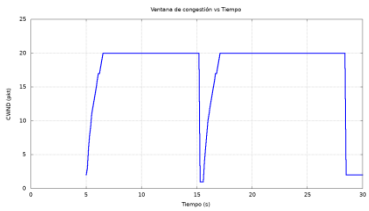

Figura 106: CWND vs. Tiempo TCP Compound-Desconexión 0,5s

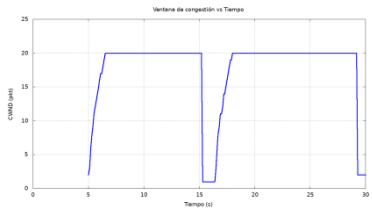

Figura 107: CWND vs. Tiempo TCP Compound-Desconexión 1s

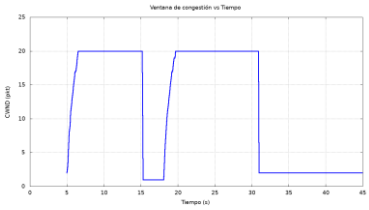

Figura 108: CWND vs. Tiempo TCP Compound-Desconexión 2s

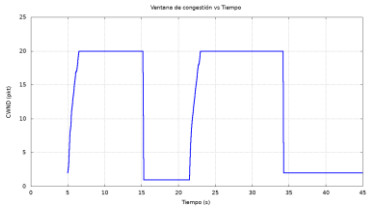

Figura 109: CWND vs. Tiempo TCP Compound-Desconexión 5s

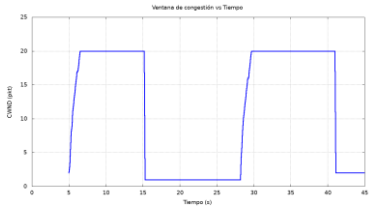

Figura 110: CWND vs. Tiempo TCP Compound-Desconexión 10s

La evolución del tamaño de la ventana de congestión de la variante TCP Compound, en función del tiempo, se puede observar en las figuras anteriores (Figura 105 a Figura 110), para las distintas longitudes de cortes.

Si se analizan las figuras anteriores, se observa que el comportamiento de la ventana de congestión en función del tiempo es similar a TCP Westwood en este escenario, con la diferencia que el tamaño se reduce a 1 durante el período de desconexión.

La presente tanda de simulaciones permite corroborar el efecto nocivo de las desconexiones frecuentes de los enlaces inalámbricos producen sobre el rendimiento del TCP.

Cuando se produce la desconexión, el flujo de datos y de ACK se corta. Al no recibir ACK de ningún tipo no se activa la fase de Fast Retransmit. En el emisor ocurrirá un timeout y en consecuencia reaccionará retransmitiendo el segmento aplicando el algoritmo de backoff exponencial e iniciando los algoritmos de control de congestión. Sin embargo, nada de esto funcionará hasta que se restablezca la conexión. De esta manera el algoritmo de backoff irá espaciando las retransmisiones duplicando el valor del temporizador de retransmisión. Una vez que se restablece la conexión y es nuevamente posible la trasmisión de los datos, lo deseable es que el paquete sea enviado inmediatamente y la tasa de envió sea similar a la que existía con anterioridad a la desconexión. Sin embargo, no es lo que sucede. El valor de RTO creció exponencialmente y puede darse el caso en que a pesar de que el enlace esta nuevamente disponible, el emisor este esperando el ACK de un segmento que nunca llego al receptor porque el enlace estaba desconectado. Agotado el tiempo de este temporizador, reenvía nuevamente el segmento y, después de un tiempo recibe el ACK correspondiente. Además de haber un tiempo de espera sin utilización del enlace, cuando comienza a enviar datos nuevamente, lo hace a

una tasa mucho menor de la que traía antes de la desconexión, pues los algoritmos de control de congestión disminuyen la tasa de inyección de paquetes, para evitar una congestión que nunca ocurrió. El comportamiento deseado en esta situación es, TCP debió retransmitir los paquetes perdidos ni bien se reconectará el destinatario y una vez recibido el ultimo ACK volver al tamaño de la ventana anterior al momento de la detección de la primera pérdida, o lo que es lo mismo, seguir inyectando datos a la red a la misma tasa que lo hacía antes de la desconexión del receptor.

En los gráficos de throughput (Figura 37 a Figura 66) y de evolución del tamaño de la ventana de congestión (Figura 81 a Figura 110), se puede observar como las distintas variantes despliegan sus mecanismos de control de congestión en forma innecesaria. Además, resulta evidente en el análisis de los gráficos de número de secuencia en función del tiempo de simulación (Figura 69 a Figura 80) cómo las desconexiones producen estos timeout en serie, lo que hace crecer exponencialmente el valor de RTO, lo que produce que, aun después de que el nodo se conecte nuevamente, no haya avance del número de secuencia y no hay utilización de la red por un período de tiempo importante.

4.6.2. Errores en ráfaga de distinta longitud en un escenario simple

Como se describió anteriormente, los errores en ráfaga, característicos de las redes con enlaces inalámbricos, tienen como consecuencia un deterioro en el rendimiento de las comunicaciones. Por lo que resulta interesante analizar cómo el protocolo se recupera después de sufrir este tipo de eventos [145]. Con este fin, en este grupo de simulaciones se realiza un análisis de cómo se ven afectadas las transmisiones de datos al sufrir pérdidas de paquetes consecutivas en ráfagas de distinta longitud. Basado en esto, se propone mediante la configuración de un modelo simple implementado en la herramienta de simulación NS-2.

Para analizar el efecto de estos errores, en una red de acceso inalámbrica, se realizaron una serie de simulaciones sobre el modelo simple de la Figura 36, de manera de aislar los efectos de la perdida de paquetes consecutivos de distinta longitud y evitar otro tipo de influencias. La selección de este modelo como una

aproximación a un escenario WLAN, con la simplificación que el enlace inalámbrico no presenta desconexiones y solo presenta este tipo de errores.

En forma similar al grupo de simulaciones anterior, la topología de 3 nodos consta de un enlace cableado full dúplex, ancho de banda 2 Mb/s, retardo de propagación 2 ms. y política de servicio de las colas DropTail. El enlace inalámbrico se configuró como modo de propagación TwoRayGround, la capa física WirelessPhy, MAC 802.11, la antena OmniAntenna y el nodo inalámbrico sin movilidad. El nodo 0 se definió como emisor y el nodo 2 como receptor. A este enlace se asoció un tráfico FTP (File Transfer Protocol) como único flujo de datos. Análogamente, la trasmisión de datos comienza a los 5 segundos de iniciada las simulaciones y están condicionadas a la transmisión de 3.000 paquetes de TCP, de 1.000 bytes c/u, independientemente de la longitud de la ráfaga de error. Estas ráfagas siempre comenzaron después de trasmitidos los primeros 999 paquetes y el ensayo concluyó al terminarse de transmitir los 3.000 paquetes.

Se realizaron simulaciones independientes sobre las implementaciones de las distintas variantes de TCP. Para cada una de ellas, se generaron simulaciones introduciendo distintas longitudes de los errores en ráfagas, con tamaños que fueron desde una prueba sin errores (0), pasando a pruebas con ráfagas de error de 5, 10, 15 y 20 paquetes perdidos continuos. Los agentes TCP que se utilizaron fueron Vegas, CUBIC, Reno y Westwood, tal como están designados e implementados en esta versión de NS-2. En el caso de TCP Vegas los valores de sus parámetros son los que utiliza el simulador por defecto, es decir $\alpha=1$ y $\beta=3$.

Resultados

Se presentan los resultados de las distintas simulaciones, para cada una de las variantes analizadas y para cada una de las longitudes de ráfaga de errores medida en cantidad de paquetes. En los resultados se observan las métricas el throughput en función del tiempo de simulación (Figura 111 a Figura 134), el número de secuencia del segmento TCP en función del tiempo (Figura 135 a Figura 140) y el tamaño de la ventana de congestión en función del tiempo (Figura 141 a Figura 146).

THROUGHPUT

Las siguientes figuras (Figura 111 a Figura 116) representan el valor del throughput en función del tiempo para el *TCP Reno*, para las distintas longitudes de errores en ráfaga ensayadas.

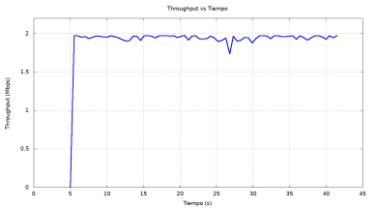

Figura 111: Throughput vs. Tiempo TCP Reno – 0 paquetes perdidos

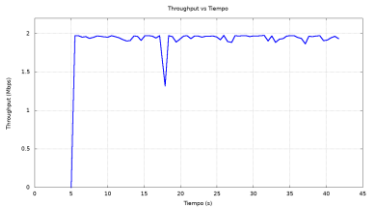

Figura 112: Throughput vs. Tiempo TCP Reno – 5 paquetes perdidos

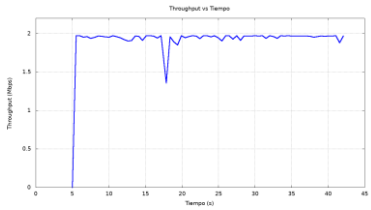

Figura 113: Throughput vs. Tiempo TCP Reno – 10 paquetes perdidos

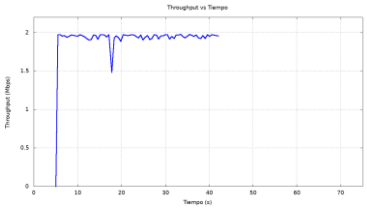

Figura 114: Throughput vs. Tiempo TCP Reno – 15 paquetes perdidos

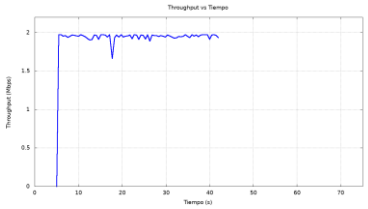

Figura 115: Throughput vs. Tiempo TCP Reno – 20 paquetes perdidos

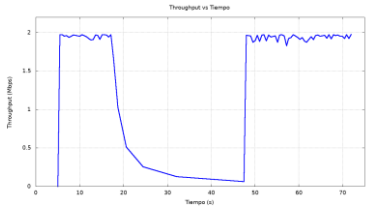

Figura 116: Throughput vs. Tiempo TCP Reno – 25 paquetes perdidos

En la secuencia de figuras anteriores se observa que el efecto de los errores en ráfaga se manifiesta en mayor medida en la longitud de 25 paquetes perdidos. Si bien produce una leve reducción del throughput para los errores de menor longitud, la recuperación al estado anterior se realiza en forma rápida.

Las siguientes figuras (Figura 117 a Figura 122) representan el valor del throughput en función del tiempo para la variante *TCP CUBIC* para las distintas longitudes de errores. Se puede observar que TCP CUBIC se ve afectado en menor medida que TCP Reno en este escenario. Si bien cuando la longitud supera los 20 paquetes perdidos el rendimiento se degrada significativamente, el tiempo de recuperación es menor que el TCP Reno, en estas condiciones.

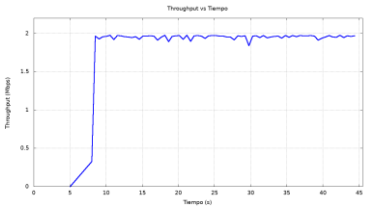

Figura 117: Throughput vs. Tiempo TCP CUBIC – 0 paquetes perdidos

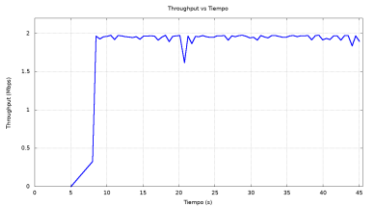

Figura 118: Throughput vs. Tiempo TCP CUBIC – 5 paquetes perdidos

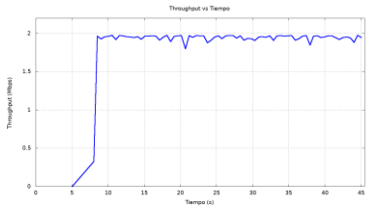

Figura 119: Throughput vs. Tiempo TCP CUBIC – 10 paquetes perdidos

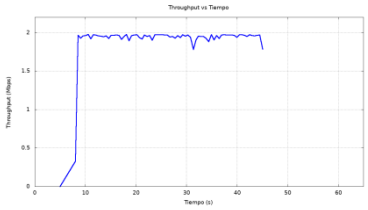

Figura 120: Throughput vs. Tiempo TCP CUBIC – 15 paquetes perdidos

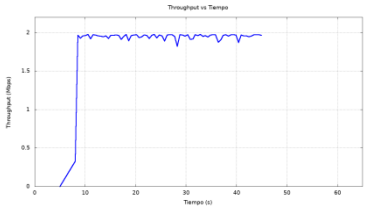

Figura 121: Throughput vs. Tiempo TCP CUBIC – 20 paquetes perdidos

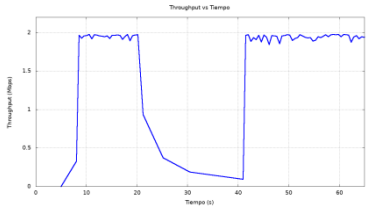

Figura 122: Throughput vs. Tiempo TCP CUBIC – 25 paquetes perdidos

En las siguientes figuras (Figura 123 a Figura 128) se puede observar el Throughput vs Tiempo de la variante *TCP Westwood* para cada uno de los errores en ráfaga ensayados en las simulaciones.

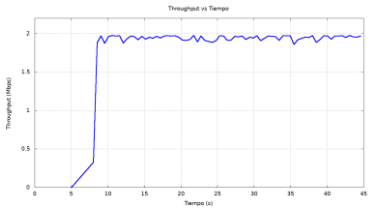

Figura 123: Throughput vs. Tiempo TCP Westwood – 0 paq. perdidos

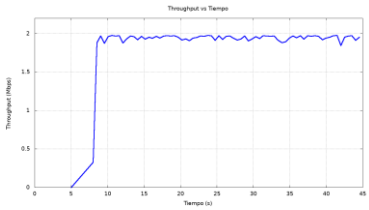

Figura 124: Throughput vs. Tiempo TCP Westwood – 5 paq. perdidos

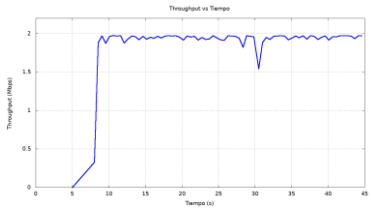

Figura 125: Throughput vs. Tiempo TCP Westwood – 10 paq. perdidos

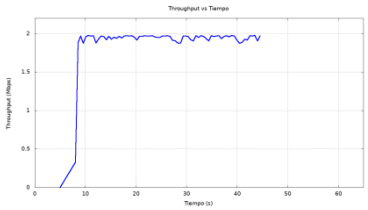

Figura 126: Throughput vs. Tiempo TCP Westwood – 15 paq. perdidos

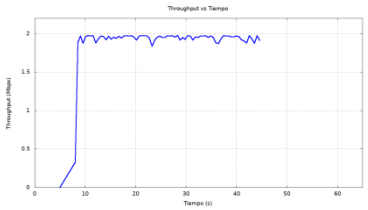

Figura 127: Throughput vs. Tiempo TCP Westwood – 20 paq. perdidos

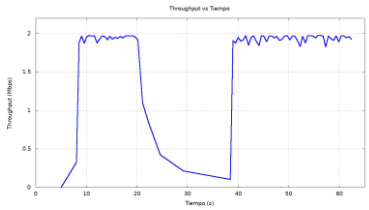

Figura 128: Throughput vs. Tiempo TCP Westwood – 25 paq. Perdidos

En las figuras anteriores se observa que TCP Westwood es menos susceptible, desde el punto de vista del valor de throughput, que Reno e inclusive que CUBIC. Para el caso del error de 25 paquetes se observa que recupera la tasa de envío en menos tiempo que las dos variantes anteriormente detalladas.

En las siguientes figuras (Figura 129 a Figura 134) se observa el valor de throughput para la variante *TCP Vegas* para las distintas ráfagas de error.

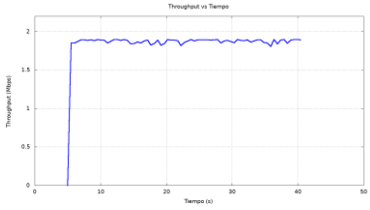

Figura 129: Throughput vs. Tiempo TCP Vegas – 0 paquetes perdidos

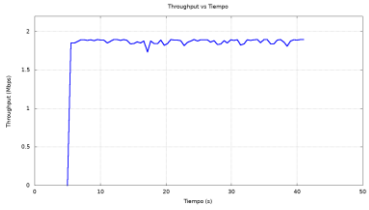

Figura 130: Throughput vs. Tiempo TCP Vegas – 5 paquetes perdidos

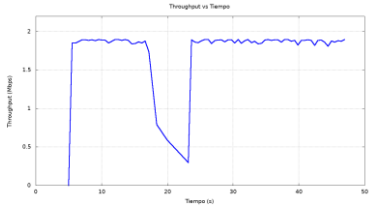

Figura 131: Throughput vs. Tiempo TCP Vegas – 10 paquetes perdidos

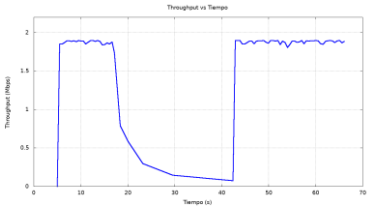

Figura 132: Throughput vs. Tiempo TCP Vegas – 15 paquetes perdidos

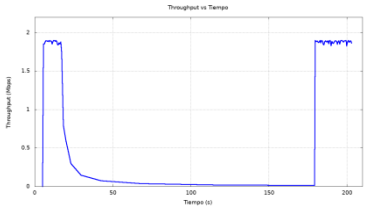

Figura 133: Throughput vs. Tiempo TCP Vegas – 20 paquetes perdidos

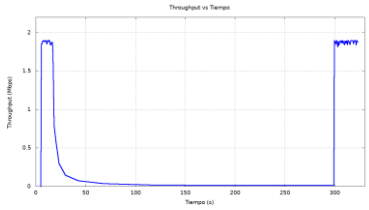

Figura 134: Throughput vs. Tiempo TCP Vegas – 25 paquetes perdidos

Cuando se observa las figuras anteriores, queda en evidencia que TCP Vegas reduce su tasa de envío de manera significativa desde los 10 paquetes perdidos consecutivos. Se observa también, que el tiempo de recuperación de esta tasa de envío se hace más largo a medida que la longitud de la ráfaga de errores crece. Comparado con Reno, CUBIC y Westwood (Figura 111 a Figura 128) es la variante más susceptible en este escenario (Figura 36).

NUMERO DE SECUENCIA

Para poder poner en evidencia este fenómeno, se presentan las figuras a continuación (Figura 135 a Figura 140) que representan la evolución del número de secuencia del segmento TCP en función del tiempo de la simulación, superpuestos en un mismo gráfico todas las variantes ensayadas, para cada longitud de errores en ráfaga.

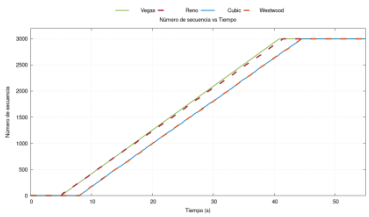

Figura 135: Nº de Sec. vs. Tiempo 0 paquetes perdidos

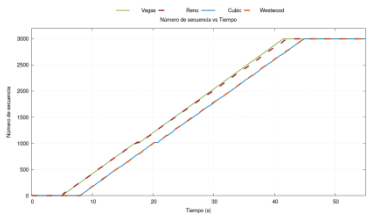

Figura 136: Nº de Sec. vs. Tiempo 5 paquetes perdidos

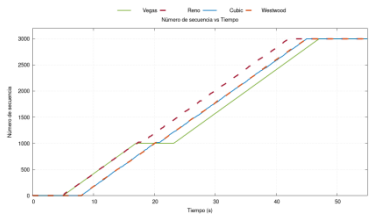

Figura 137: Nº de Sec. vs. Tiempo 10 paquetes perdidos

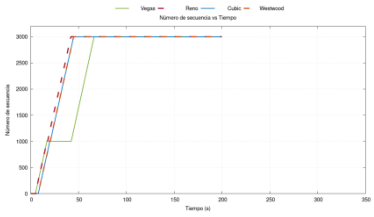

Figura 138: Nº de Sec. vs. Tiempo 15 paquetes perdidos

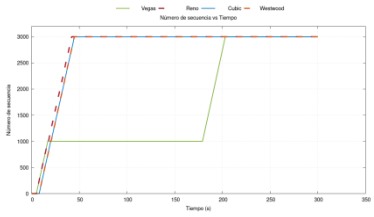

Figura 139: Nº de Sec. vs. Tiempo 20 paquetes perdidos

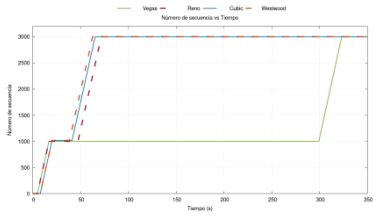

Figura 140: Nº de Sec. vs. Tiempo 25 paquetes perdidos

En la secuencia de figuras anterior, se observa que el crecimiento del número de secuencia es similar en los casos de Reno, CUBIC y Westwood más allá de alguna pequeña diferencia. Sin embargo, TCP Vegas sufre una demora

significativa en este escenario comparado con las otras variantes ensayadas. Se observa cómo transcurre el tiempo y, mientras las otras variantes continúan avanzando en el número de secuencia transmitido, TCP Vegas se queda estancado. Si bien este fenómeno es visible en ráfagas de corta longitud, en las de 15 paquetes consecutivos perdidos (Figura 138) es donde se comienza a ser mucho más evidente.

VENTANA DE CONGESTIÓN

Para analizar cómo el comportamiento de los diferentes algoritmos de control de congestión, resulta interesante observar cómo varía el tamaño de la ventana de congestión en función del tiempo.

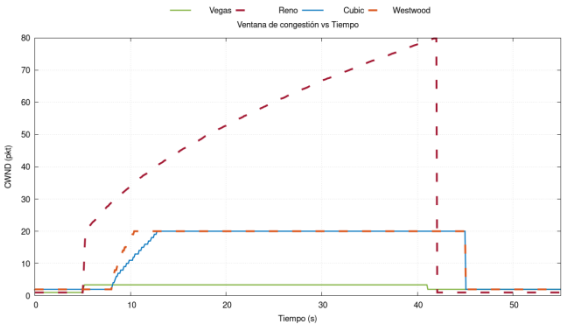

Figura 141: CWND vs. Tiempo

0 paquetes perdidos

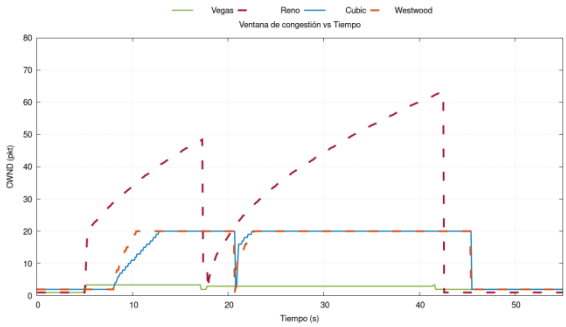

Figura 142: CWND vs. Tiempo

5 paquetes perdidos

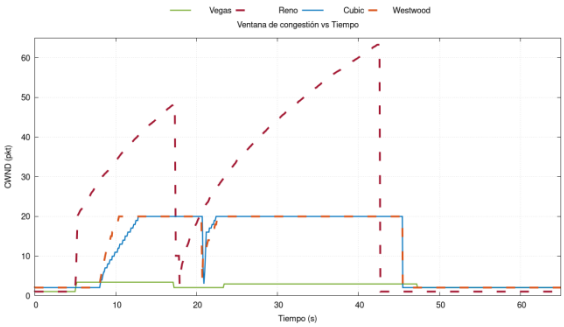

Figura 143: CWND vs. Tiempo

10 paquetes perdidos

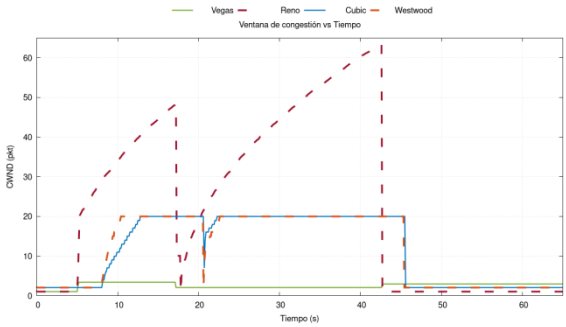

Figura 144: CWND vs. Tiempo

15 paquetes perdidos

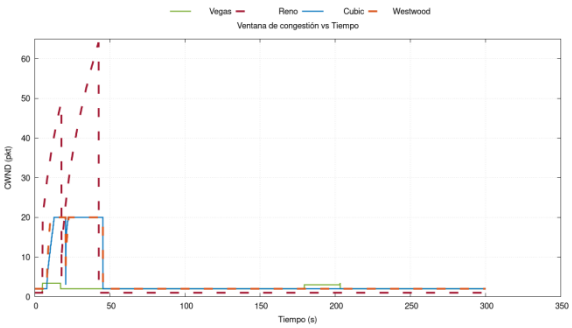

Figura 145: CWND vs. Tiempo

20 paquetes perdidos

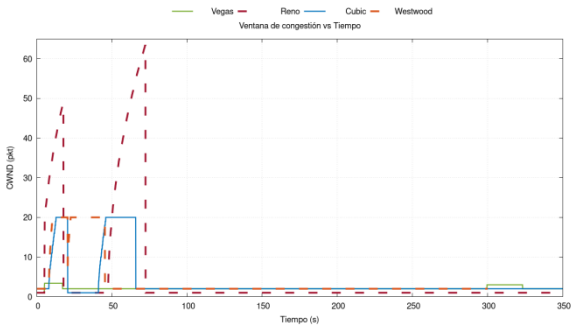

Figura 146: CWND vs. Tiempo

25 paquetes perdidos

En las figuras anteriores (Figura 141 a Figura 146) se presenta la evolución de las ventanas de las variantes superpuestas en el mismo gráfico para cada una de las longitudes de errores ensayadas. Se puede observar cómo los algoritmos de control de congestión reducen el tamaño de las ventanas y, por lo tanto, la tasa de envío, aun cuando la red no presenta congestión alguna. Esto degrada el rendimiento de todas las variantes.

De la misma serie de gráficos resulta interesante observar el tiempo necesario que requieren, cada una de las estrategias de control de congestión, para recuperar la tasa de envío al valor anterior a la ráfaga de errores.

RÁFAGA DE 15 PAQUETES

A partir de los 15 paquetes consecutivos perdidos, se empieza a observar diferencias significativas en las variantes de TCP ensayadas, por lo que resulta interesante detenerse a explorar el comportamiento para esta longitud.

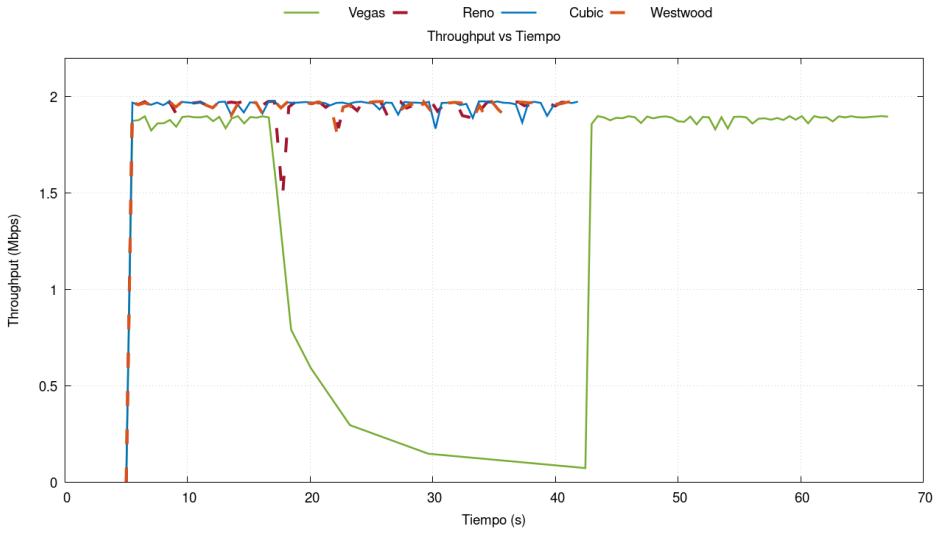

Figura 147: Throughput vs Tiempo - 15 paquetes perdidos

En la figura anterior (Figura 147) se puede observar el throughput en función del tiempo de las cuatro variantes ensayadas en el mismo gráfico, para un error en ráfaga de 15 paquetes de longitud. De ella se observa con mayor claridad cuanto se reduce el rendimiento de TCP Vegas comparándolo con las otras variantes.

En la siguiente figura (Figura 148) se observa el crecimiento del número de secuencia en función del tiempo de las variantes ensayadas para 15 paquetes

perdidos. Se observa que todas las variantes tienen un quiebre en la línea de crecimiento del número de secuencia. Sin embargo, es Vegas quien demora más de 20 segundos en retomar el envío.

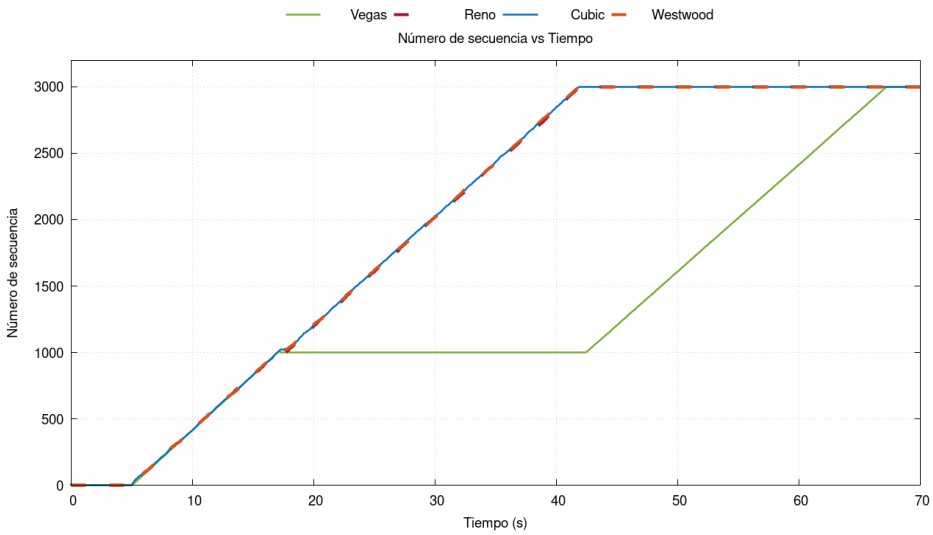

Figura 148: Nº de secuencia vs Tiempo - 15 paquetes perdidos

En la siguiente figura (Figura 149) se observa la evolución del tamaño de la ventana de congestión para las variantes de TCP ensayadas para una longitud de ráfaga de 15 paquetes.

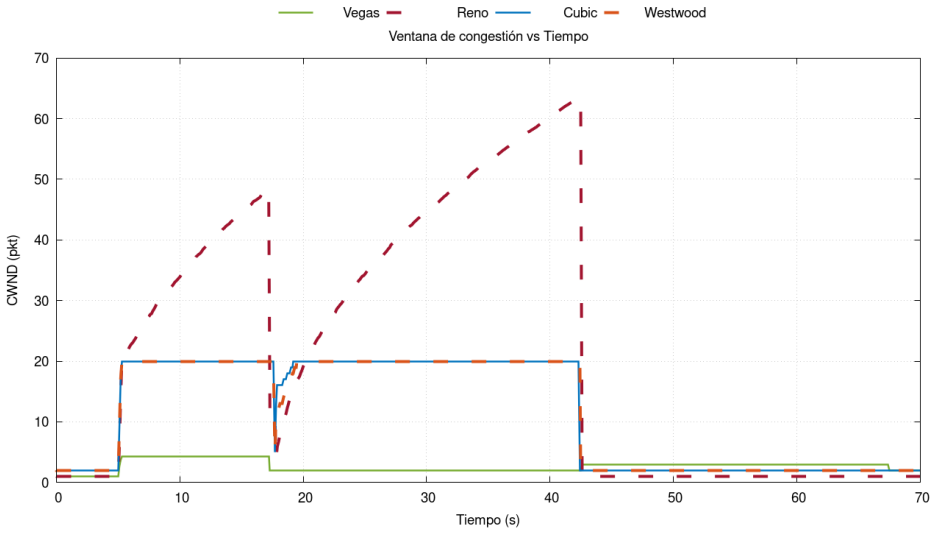

Figura 149: CWND vs Tiempo - 15 paquetes perdidos

En la figura anterior se observa con mayor detalle las estrategias de los mecanismos de control de congestión. Se observa como TCP Vegas demora

más de 20 segundos en inflar la ventana de congestión, y continuar con el envío de paquetes.

A partir de lo que se puede observar en la serie de figuras (Figura 147, Figura 148 y Figura 149), se puede concluir que para el modelo definido en la Figura 36, cuando la longitud de la ráfaga de paquetes perdidos es igual a 15, TCP Vegas comienza a mostrar signos de perdida de rendimiento importantes, mientras que en el resto de las variantes ensayadas sólo comienzan a ser significativo con mayores longitudes de errores.

A partir de lo anteriormente expuesto, resulta evidente que es la variante TCP Vegas la que es más susceptible a los errores en ráfaga. Comparando las gráficas del throughput instantáneo de Reno, Westwood y Vegas (Figura 112 a Figura 134) se puede observar que es esta última variante, la que empieza a degradar su tasa de envío anticipadamente. A partir de los 15 paquetes consecutivos perdidos, esa mayor susceptibilidad se hace muy evidente, hecho que se pone en evidencia observando los gráficos de la evolución del número de secuencia en función del tiempo (Figura 135 a Figura 140).

OTRAS VARIANTES DE TCP

A partir de lo anteriormente observado, resulta interesante ampliar a otras variantes de TCP en este modelo, el ensayo para una longitud de ráfaga de 15 paquetes perdidos y analizar su comportamiento, en contraste con las anteriores.

En las siguientes figuras (Figura 150 a Figura 155) se observan los valores del throughput y del tamaño de la ventana de congestión en función del tiempo para las variantes *SACK*, *FACK* y *New Reno*.

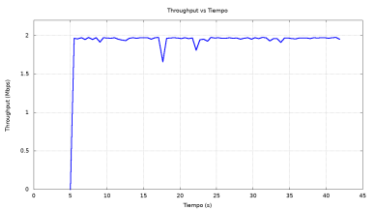

Figura 150: Throughput vs. Tiempo TCP SACK – 15 paq. perdidos

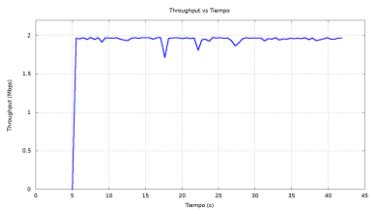

Figura 151 Throughput vs. Tiempo TCP FACK – 15 paq. perdidos

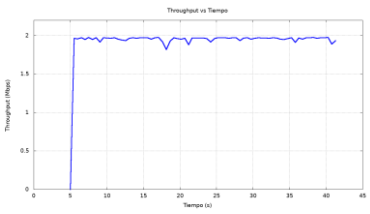

Figura 152: Throughput vs. Tiempo TCP New Reno – 15 paq. perdidos

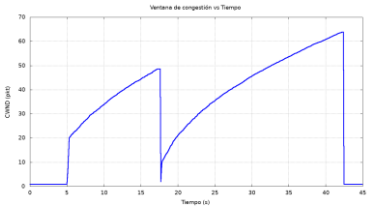

Figura 153: CWND vs. Tiempo
TCP SACK – 15 paq. perdidos

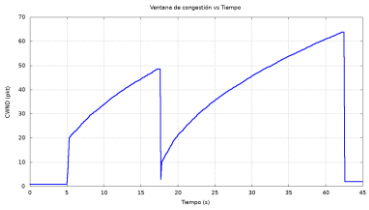

Figura 154: CWND vs. Tiempo
TCP FACK – 15 paq. perdidos

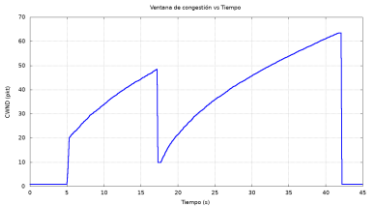

Figura 155: CWND vs. Tiempo
TCP New Reno – 15 paq. perdidos

Como se observa en los anteriores gráficos, la evolución de la ventana de congestión es similar a la de la variante TCP Reno, debido a que implementan modificaciones sobre el control de congestión de este último. En el caso de SACK y FACK estas modificaciones no mejoran la respuesta, lo que resulta en una graficas de throughput instantáneo muy similar a la de Reno para este escenario y esta longitud de ráfaga. En el caso de New Reno, en el momento de los errores, reduce en menor medida la ventana de congestión, lo que permite una mejor respuesta del throughput instantáneo.

Las siguientes figuras (Figura 156 y Figura 157) representan el throughput y la Ventana de Congestión en función del tiempo de *TCP Veno* para errores en ráfaga de 15 paquetes de longitud.

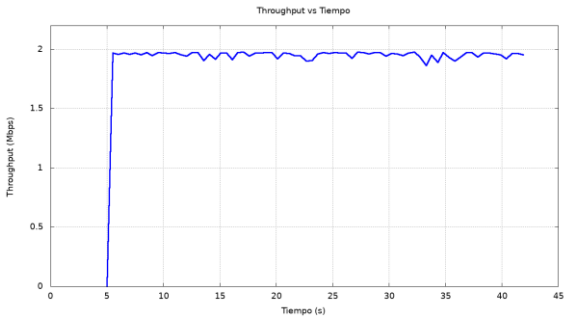

Figura 156: Throughput vs. Tiempo
TCP Veno – 15 paquetes perdidos

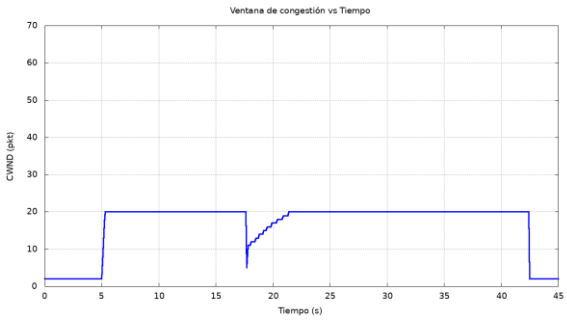

Figura 157: CWND vs. Tiempo
TCP Veno – 15 paquetes perdidos

Se observa que el throughput instantáneo (Figura 156) prácticamente no se resiente, lo que se puede comprender analizando la gráfica de la ventana de congestión de TCP Veno (Figura 157). Se puede observar que la reducción de la ventana es menor (se reduce a 5 paquetes) y el tiempo necesario para la recuperación al tamaño anterior es sustancialmente menor con respecto a otras variantes. Este comportamiento se puede observar también en las siguientes

figuras (Figura 158 a Figura 163) para las variantes *TCP Compound*, *TCP Illinois* y *HS-TCP (HighSpeed*).

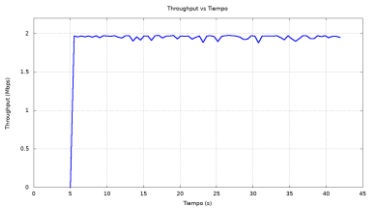

Figura 158: Throughput vs. Tiempo
TCP Compound – 15 paq. perdidos

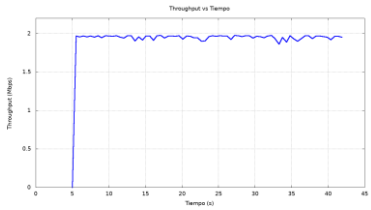

Figura 159: Throughput vs. Tiempo
TCP Illinois – 15 paq. perdidos

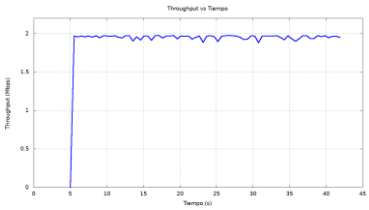

Figura 160: Throughput vs. Tiempo
TCP HS-TCP – 15 paq. perdidos

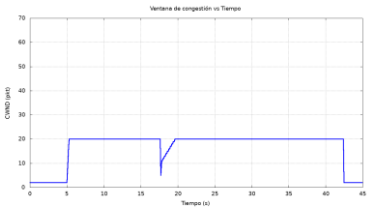

Figura 161: CWND vs. Tiempo
TCP Compound – 15 paq. perdidos

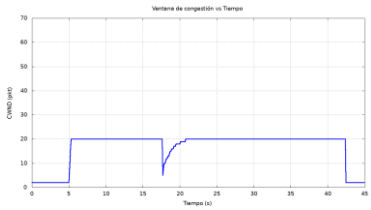

Figura 162: CWND vs. Tiempo
TCP Illinois – 15 paq. perdidos

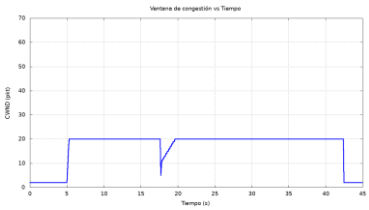

Figura 163: CWND vs. Tiempo
TCP HS-TCP – 15 paq. perdidos

Como se puede observar, ninguna de las variantes resiente el throughput de manera apreciable. También se observa, en la evolución de la ventana de congestión, una reducción a un valor más alto y una recuperación más rápida similar a TCP Veno. Sin embargo, se puede observar diferencias en el crecimiento posterior a los errores, definido por los distintos algoritmos de control de congestión.

A continuación (Figura 164 y Figura 165) los gráficos de Throughput vs. Tiempo y Ventana de Congestión vs. Tiempo de *TCP Hybla* para una ráfaga de 15 paquetes.

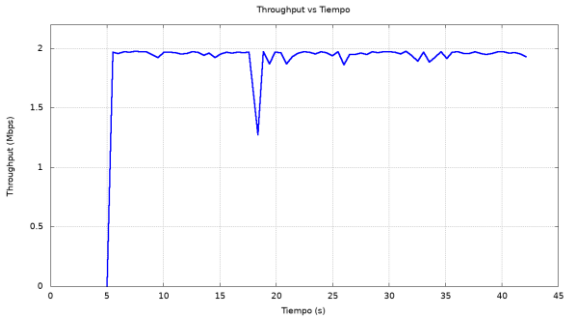

Figura 164: Throughput vs. Tiempo
TCP Hybla – 15 paquetes perdidos

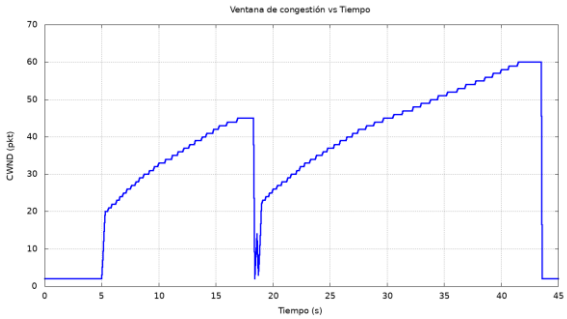

Figura 165: CWND vs. Tiempo
TCP Hybla – 15 paquetes perdidos

El comportamiento observado del throughput instantáneo en la figura anterior (Figura 164) es similar a Reno (Figura 114), debido a que es una variante que modifica parte de los algoritmos de New Reno para independizarlos del valor de RTT. Al no estar en un escenario con valores altos de RTT la evolución de su ventana de congestión (Figura 165) es similar.

Como se observa en la prueba, TCP Vegas revela una sensibilidad mayor que las otras variantes de TCP, mostrando una demora en el reinicio de la trasmisión. En las secuencias de la Figura 129 a la Figura 134, se puede apreciar cómo las ráfagas afectan de manera progresiva su comportamiento, analizando el throughput en función del tiempo para los valores de error en ráfaga correspondiente a 5, 10, 15, 20 y 25 paquetes, donde se observa cómo el protocolo extiende progresivamente la demora en iniciar nuevamente la transmisión de los datos, resultando de esta manera sensible a este tipo de errores desde un primer un momento. Estas dificultades son más moderaras en las demás variantes ensayadas hasta la pérdida de 15 paquetes consecutivos. Como resulta evidente en los distintos gráficos, cada una de las variantes ensayadas reacciona al evento de los errores disparando sus mecanismos de control de congestión, interpretando que una situación de este tipo se ha producido, cuando en realidad lo que se presenta es un problema de pérdida de paquetes por errores en ráfaga, inherentes al canal inalámbrico.

4.6.3. TCP Vegas: Errores en ráfaga de distinta longitud

En virtud de lo observado de la variante TCP Vegas en el conjunto de las simulaciones anteriores (*4.6.2 Errores en ráfaga de distinta longitud en un escenario simple*) y, para poder analizar con mayor detalle el comportamiento bajo condiciones similares, se propone este conjunto de ensayos en un escenario sencillo para distintos valores de sus parámetros característicos α (alfa) y β (beta) de este protocolo, a fin de verificar su respuesta ante los errores en ráfaga de longitud variable [146].

Para este estudio, se realizaron una serie de ensayos, sobre el modelo de tres nodos de la Figura 36, incorporando además de los errores en ráfaga de distintas longitudes, distintas combinaciones de valores de α y β. Resulta interesante analizar de qué manera las variaciones de estos parámetros

afectan el desempeño del protocolo TCP Vegas ante la presencia de este tipo de errores. Dado el carácter proactivo de los algoritmos de control de congestión de TCP Vegas, se comprobará más adelante en este capítulo, que en la contienda con la mayoría de las otras versiones no puede obtener una porción justa de los recursos de la subred que comparten. Es por esta razón, que el modelo simple y de un solo flujo resulta útil para el estudio de esta variante. Para estas simulaciones se generó un flujo FTP con la variante TCP Vegas.

Se realizaron simulaciones independientes para las distintas longitudes de los errores en ráfagas, con longitudes que varían de 0 a 20 paquetes consecutivos perdidos. Para cada una de estas longitudes se ensayaron los valores de α (de 1 a 10) y de β (de 3 a 12) de forma tal que α siempre fuese menor que β (β>α).

En forma análoga a las simulaciones anteriores, en cada una se transmitieron 3.000 paquetes de TCP, de 1.000 Bytes cada uno. La transmisión comenzó a los 5 segundos, y las ráfagas sucesivas se introdujeron al alcanzar el número de secuencia 999.

Resultados

En los ensayos anteriores (*4.6.2 Errores en ráfaga de distinta longitud en un escenario simple*), se pudo observar cómo TCP Vegas comenzaba a reducir su rendimiento en forma significativa, en comparación a otras variantes, a partir de las ráfagas de 15 paquetes perdidos. A partir de ello, resulta interesante profundizar este estudio, evaluando, en principio, el comportamiento de esta variante para esta longitud de errores al modificar sistemáticamente los valores de los parámetros α y β.

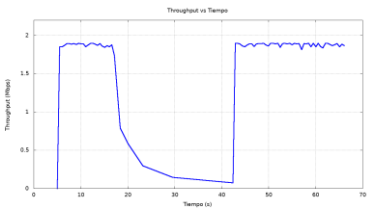

Figura 166:Throughput vs. Tiempo TCP Vegas(1,3)–15 paq. perdidos

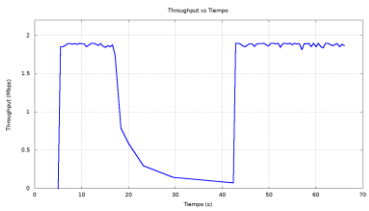

Figura 167:Throughput vs. Tiempo TCP Vegas(1,5)–15 paq. perdidos

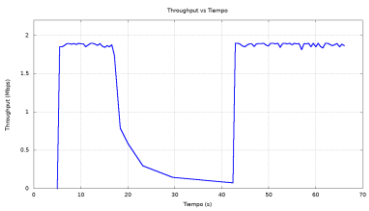

Figura 168:Throughput vs. Tiempo TCP Vegas(1,7)–15 paq. perdidos

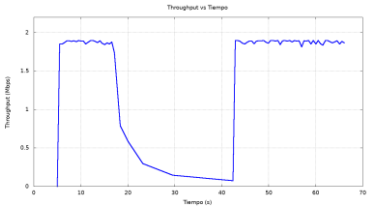

Figura 169:Throughput vs. Tiempo TCP Vegas(1,9)–15 paq. perdidos

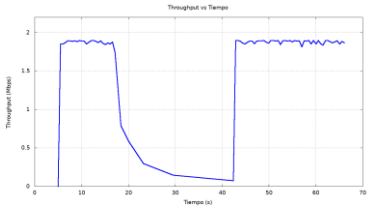

Figura 170:Throughput vs. Tiempo TCP Vegas(1,11)–15 paq. perdidos

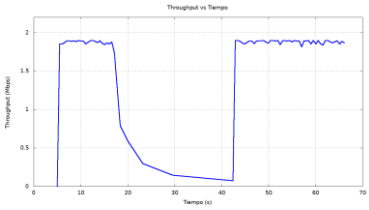

Figura 171:Throughput vs. Tiempo TCP Vegas(1,12)–15 paq. perdidos

En las figuras anteriores (Figura 166 a Figura 171), se observa el valor del throughput en función del tiempo para un valor de α fijo (α=1) y una secuencia de β de 3, 5, 7, 9, 11 y 12, para una ráfaga de 15 paquetes pedidos. Lo que se puede inferir de las figuras anteriores es que el parámetro β, que define uno de los umbrales de este control proactivo, no influye en el rendimiento en este escenario, con un solo flujo. El gráfico es prácticamente idéntico en todas las figuras de la serie anterior, para el rango de valores ensayados de β.

Análogamente, las siguientes figuras (Figura 172 a Figura 177) representan el valor del throughput en función del tiempo para un valor de β fijo (β=12) y una serie de valores para α de 1, 2, 5, 7, 9 y 11.

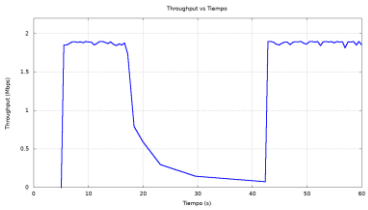

Figura 172:Throughput vs. Tiempo TCP Vegas(1,12)–15 paq. perdidos

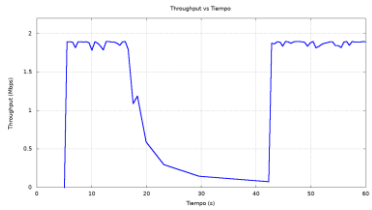

Figura 173:Throughput vs. Tiempo TCP Vegas(2,12)–15 paq. perdidos

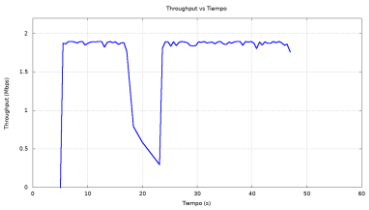

Figura 174:Throughput vs. Tiempo TCP Vegas(5,12)–15 paq. perdidos

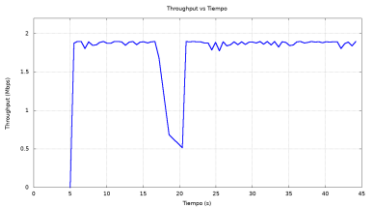

Figura 175:Throughput vs. Tiempo TCP Vegas(7,12)–15 paq. perdidos

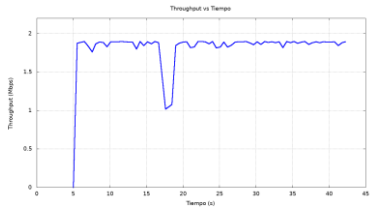

Figura 176:Throughput vs. Tiempo TCP Vegas(9,12)–15 paq. perdidos

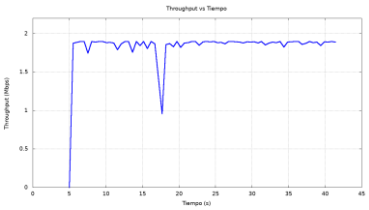

Figura 177:Throughput vs. Tiempo TCP Vegas(11,12)-15 paq. perdidos

En la secuencia de figuras anterior, se puede observar que la variación del parámetro α mejora la respuesta de esta variante, en este escenario y para una pérdida de 15 paquetes consecutivos.

En la siguiente figura (Figura 178) se observan los valores de throughput promedio como función de la longitud de la ráfaga de paquetes perdidos,

donde, cada una de las curvas representa los valores de la métrica para un valor determinado de α y un conjunto de valores de β (con β > α).

Como se puede desprender las figuras anteriores de Throughput vs Tiempo (Figura 166 a Figura 177), la influencia del parámetro β no es significativo para el modelo utilizado, debido a su simpleza y a la presencia de solo un flujo TCP. De esta manera, en la Figura 178, las curvas representan el valor de α para todos los valores de β ensayados en esta simulación. Por otro lado, se observa que el comportamiento de estas curvas es similar en la mayoría de los casos.

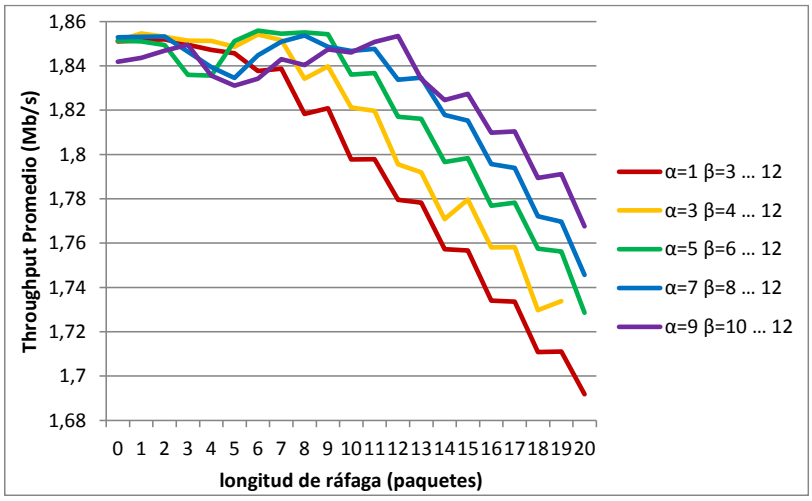

Figura 178: Throughput Promedio vs. Longitud de Ráfaga, para distintas combinaciones de α y β

Comienza con una zona de transición, donde los valores no difieren más del 1,5% del máximo, para a continuación comenzar una etapa de caída constante en el tiempo, acompañados de leves repuntes.

La siguiente figura (Figura 179) representa el valor de latencia promedio en función de la cantidad de paquetes perdidos en forma consecutiva. Para el caso de la figura, se observa un comportamiento estable, con variaciones que no presentan un desvío significativo. En estas curvas, a medida que el valor de α crece, el valor de latencia aumenta. Una de las causas del comportamiento de la latencia promedio, se debe a que a medida que el valor de α crece en esta simulación, también lo hace el throughput instantáneo lo que produce más carga en la red, lo que hace que las colas y los buffers estén más llenos. Por lo tanto, más utilización de los recursos, mejor rendimiento, más latencia.

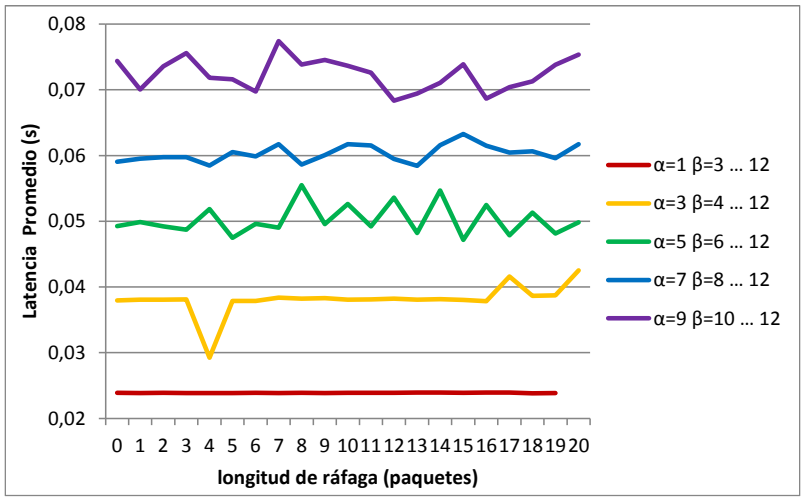

Figura 179: Latencia Promedio vs. Longitud de Ráfaga, para distintas combinaciones de α y β

La siguiente figura (Figura 180) representa el tiempo total de transmisión en función de la cantidad de paquetes perdidos en forma consecutiva. Cada una de las gráficas de la figura representa un valor de α y combinado con los valores de β ensayados.

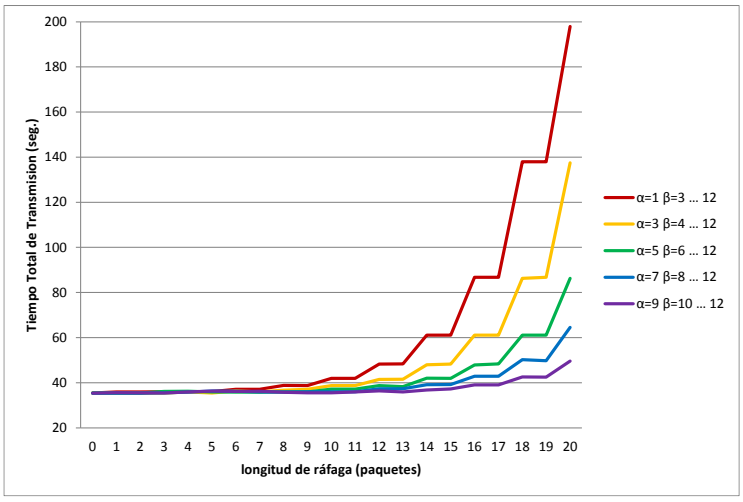

Figura 180: Tiempo Total de Transmisión vs. Longitud de la ráfaga, para distintas combinaciones de α y β

Lo que se observa en la figura anterior, es la inversa de la Figura 178, los tiempos totales comienzan a crecer a medida que crece la cantidad de paquetes perdidos. Así mismo, otra vez se observa la casi nula influencia del parámetro β, mientras que las diferentes curvas correspondientes a distintos valores de α muestran menores tiempos de transmisión para mayores valores, lo que es coherente con la Figura 178. Asimismo, es coherente el crecimiento de los valores de latencia promedio de la Figura 179 a medida que aumenta el

valor de α. Al haber un único flujo de datos en el modelo, el tráfico puede disponer de todo el ancho de banda disponible. A la luz de los resultados obtenidos, se observa que en el rango de errores introducidos en ráfagas y los valores de α y β considerados, α domina la escena y condiciona la respuesta.

Para verificarlo se observa, a continuación, una secuencia de gráficos en 3 dimensiones (Figura 181 a Figura 192) que muestran el throughput promedio y la latencia promedio, para los errores en ráfaga de distinta longitud, en función de valores de α y β (α < β).

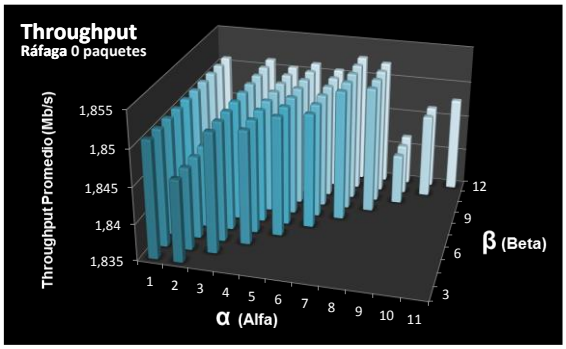

Figura 181: Throughput Promedio en función de α y β
0 paquetes perdidos

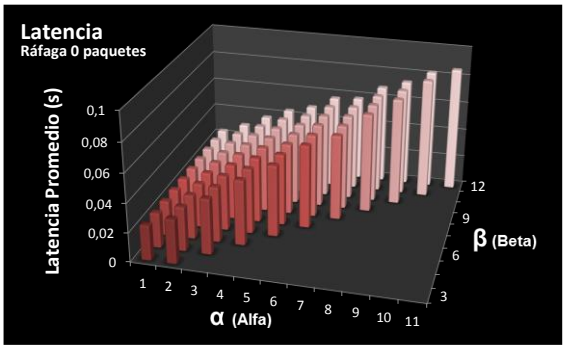

Figura 182: Latencia Promedio en función de α y β
0 paquetes perdidos

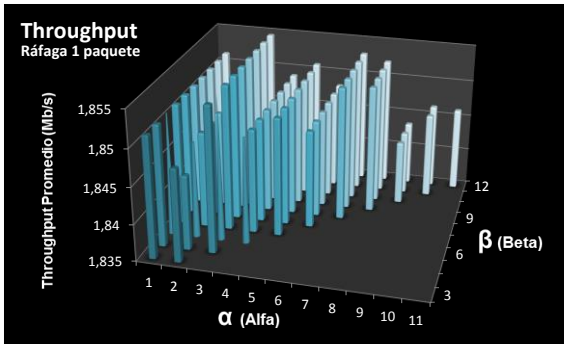

Figura 183: Throughput Promedio en función de α y β
1 paquete perdido

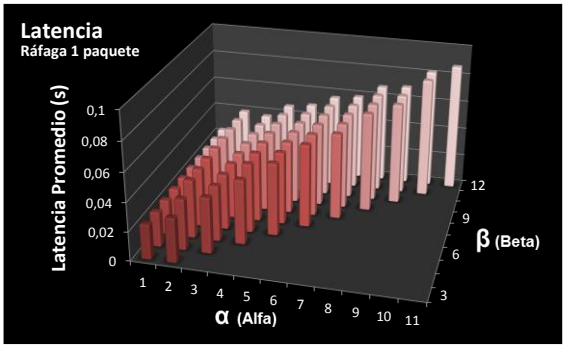

Figura 184: Latencia Promedio en función de α y β
1 paquete perdido

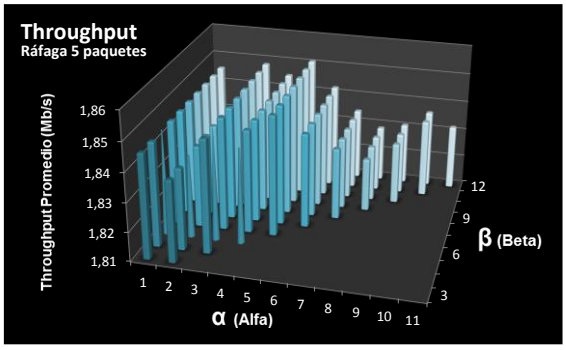

Figura 185: Throughput Promedio en función de α y β
5 paquetes perdidos

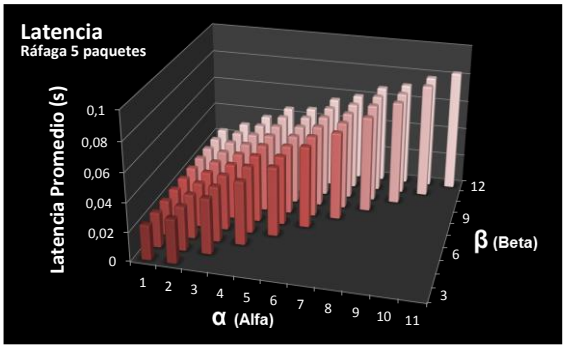

Figura 186: Latencia Promedio en función de α y β
5 paquetes perdidos

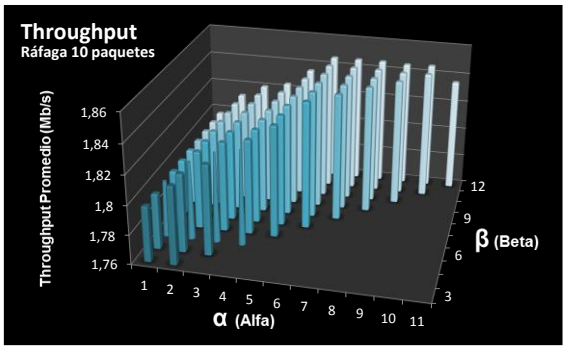

Figura 187: Throughput Promedio en función de α y β
10 paquetes perdidos

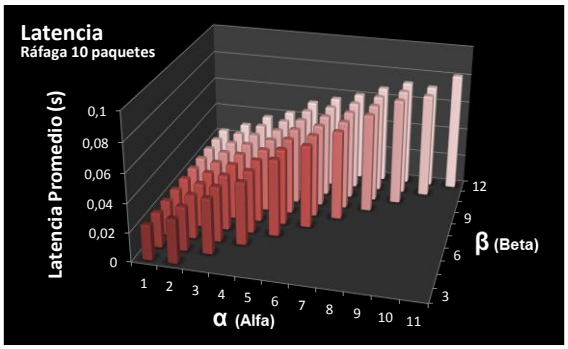

Figura 188: Latencia Promedio en función de α y β
10 paquetes perdidos

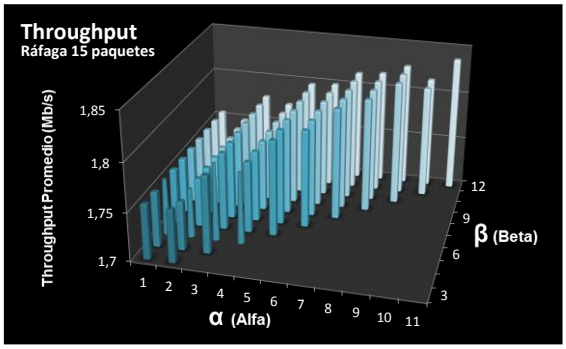

Figura 189: Throughput Promedio en función de α y β
15 paquetes perdidos

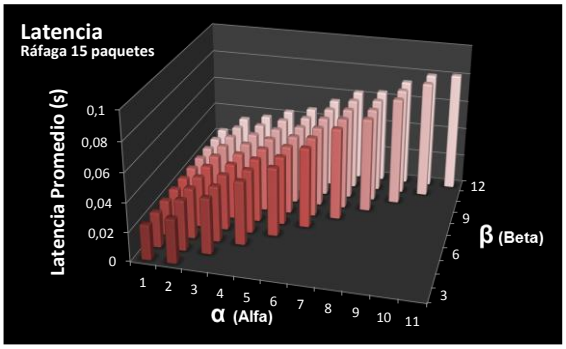

Figura 190: Latencia Promedio en función de α y β
15 paquetes perdidos

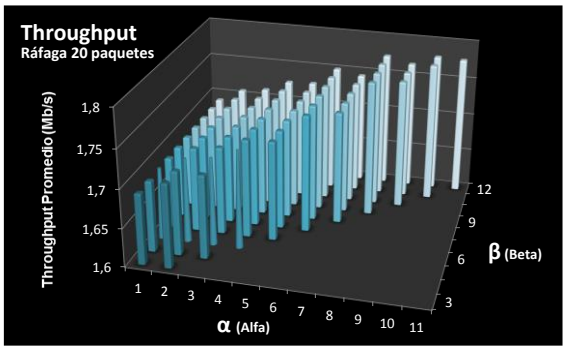

Figura 191: Throughput Promedio en función de α y β
20 paquetes perdidos

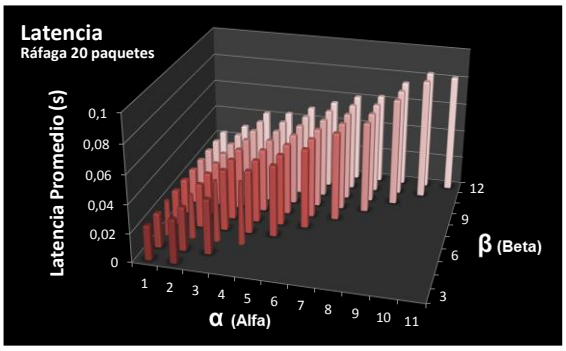

Figura 192: Latencia Promedio en función de α y β
20 paquetes perdidos

La secuencia anterior de figuras, permite corroborar claramente que para un valor de α dado, éste subordina influencia de β dentro del rango seleccionado para todos los ensayos realizados en este escenario y en estas condiciones de tráfico.

Como se ve en la Figura 178 y la Figura 179, si se evalúa el rendimiento de la transmisión a través del throughput promedio, este se mantiene relativamente similar para los valores de α analizados hasta la longitud de 6. Para valores mayores a 6 errores en ráfagas, tiene una tendencia decreciente.

Por último, la serie de pruebas de throughput y latencia, para variaciones de α (2≤α≤11) y β (3≤β≤12), con la restricción α<β (Figura 181 a Figura 192), confirman claramente que los resultados tienen una dependencia directa de la variación de α y se mantienen independientes de la variación de β, para el rango ensayado en este escenario. El motivo no es sólo la simpleza del modelo utilizado, sumado a que la variante es proactiva y no llega por si misma a la red al estado de congestión. Además, la presencia de un único flujo evita las contiendas por los recursos de la red.

4.6.4. TCP Vegas: variación de los parámetros α y β, con errores en ráfaga

En el grupo de simulaciones descriptas en *4.6.2 Errores en ráfaga de distinta longitud en un escenario simple*, quedó en evidencia que TCP Vegas presenta mayor susceptibilidad que otras variantes a los errores en ráfaga. Sin embargo, en *4.6.3 TCP Vegas: Errores en ráfaga de distinta longitud,* se observó que TCP Vegas puede mejorar su respuesta mediante la variación de sus parámetros α y β en este escenario. A partir de esto, este conjunto de simulaciones tiene como objetivo analizar la posibilidad de encontrar una configuración de parámetros que permitan mejorar el rendimiento de Vegas [147] y llevarlo a un valor similar al de Reno, en el modelo de la Figura 36.

Al observar la serie de figuras (Figura 129 a Figura 134), se puede inferir cómo es la respuesta de TCP Vegas en este escenario, considerando los valores por defecto de los parámetros (α=1, β=3). Determinar una combinación adecuada de valores para estos parámetros resulta de interés para obtener una mejora en el rendimiento. En particular, en estos resultados se observó que el tiempo de recuperación ante los errores en ráfaga, aumenta notablemente a partir de 15 paquetes de longitud en comparación con el resto de las variantes ensayadas.

Sin embargo, se pudo observar en la secuencia de figuras (Figura 181 a Figura 192), que se puede mejorar la respuesta de Vegas cuando se varían sistemáticamente los valores de α y de β en este escenario. Por ello es interesante tomar como referencia esta condición de transmisión y realizar ahora pruebas con una variación de los parámetros de mayor amplitud.

Con este objetivo, se realizaron una serie de simulaciones en el mismo escenario (Figura 36), con un solo flujo TCP Vegas ampliando el rango de variación de los parámetros (1≤α≤30 y 3≤β≤32), para las distintas longitudes de errores en ráfaga.

En forma análoga a la tanda de simulaciones anterior, se comenzó con simulaciones independientes, con una ráfaga de 15 errores consecutivos, variando los valores de α y β, con la condición que siempre α<β. Asimismo, se realizó, en las mismas condiciones, una serie de simulaciones, utilizando la variante TCP Reno para su contraste.

A partir de lo anteriormente expuesto, se propone encontrar un par de valores de α y de β, que en este modelo mejoren el rendimiento de TCP Vegas y que sea comparable con el rendimiento de TCP Reno en estas condiciones.

Resultados

A partir de observado anteriormente (4.6.3 TCP Vegas: Errores en ráfaga de distinta longitud) que el valor de α condiciona al valor de β y que este tiene escasa influencia en el escenario, se obtuvieron los valores de tiempo total de trasmisión, throughput promedio y latencia promedio en función de α, para una longitud de errores en ráfaga de 15 paquetes. En las siguientes figuras (Figura 193 a Figura 195) se observa el comportamiento de estas métricas.

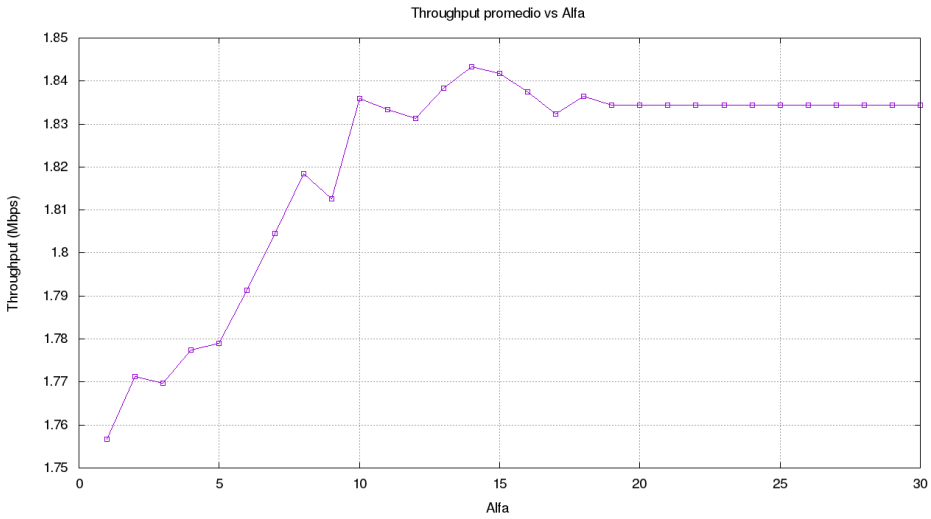

Figura 193: Throughput Promedio vs. α (alfa), 15 paquetes perdidos

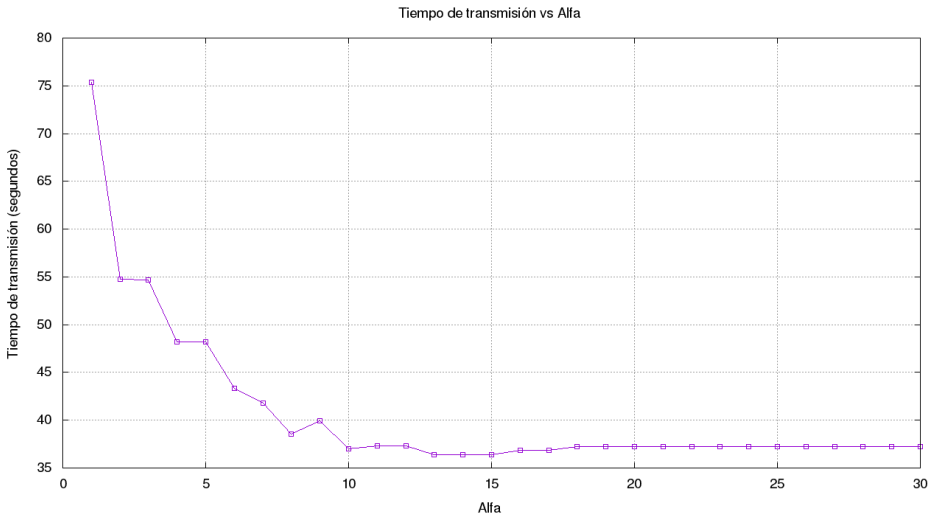

Figura 194: Tiempo Total de Transmisión vs. α (alfa), 15 paquetes perdidos

Como se observa en las figuras anteriores, es posible encontrar mediante la variación de los parámetros en un rango determinado, valores que mejoren en rendimiento a través del Tiempo Total de Transmisión (Figura 193) y de Throughput Promedio (Figura 194) para este escenario. Estas dos curvas presentan coherencia y muestran que podría obtenerse la mejor respuesta, en estas condiciones, para un valor aproximado de α=15. Por lo tanto, para el rango de α analizado y, en este escenario en particular, ambas curvas demuestran tener los mejores valores en el entorno de α=15.

En la siguiente figura (Figura 195) se observa la latencia promedio en función del valor de α, para una ráfaga de errores de 15 paquetes de longitud.

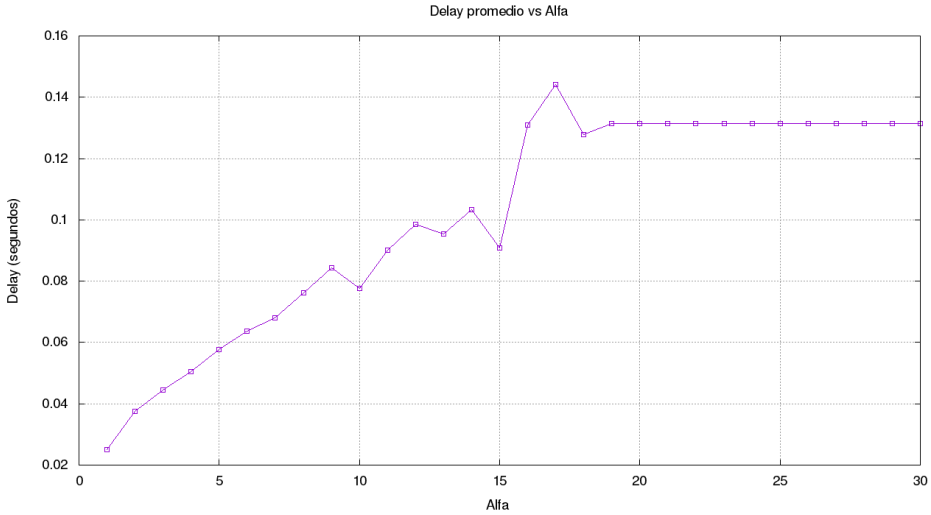

Figura 195: Latencia Promedio vs. α (alfa), 15 paquetes perdidos

En la figura anterior se observa que, análogamente, la latencia promedio presenta un mínimo local en los alrededores de α=15.

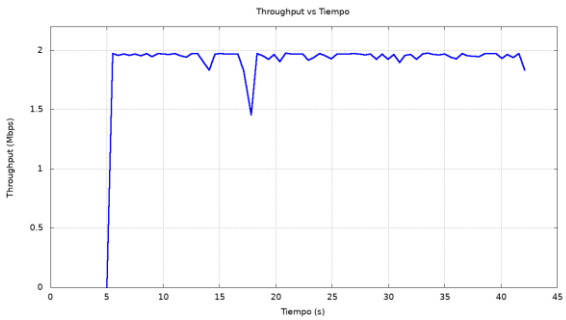

Figura 196: Throughput vs. Tiempo
TCP Reno – 15 paquetes perdidos

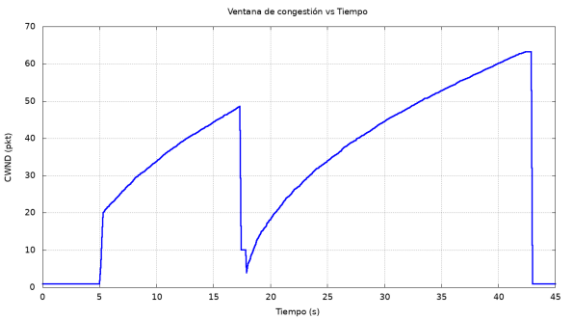

Figura 197: CWND vs. Tiempo
TCP Reno – 15 paquetes perdidos

En las figuras anteriores se observa el comportamiento del throughput (Figura 196) y de la ventana de congestión (Figura 197) de TCP Reno para una ráfaga de error de 15 paquetes en el escenario propuesto.

En las siguientes figuras (Figura 198 a Figura 206) se puede observar el throughput en función del tiempo de TCP Vegas, para distintos valores de α y β tales que, α>12 y β=α+2, con pérdida de 15 paquetes en forma consecutiva. Esta serie de gráficos es la extensión del rango de valores para los parámetros de TCP Vegas, de la que se analizó en *4.6.3 TCP Vegas: variación de los parámetros α y β, con errores en ráfaga* (Figura 172 a Figura 177).

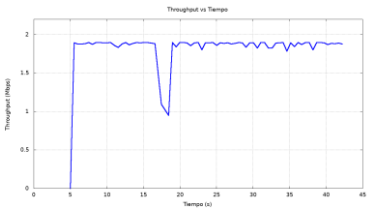

Figura 198: Throughput vs. Tiempo
TCP Vegas(12,14)–15 paq. perdidos

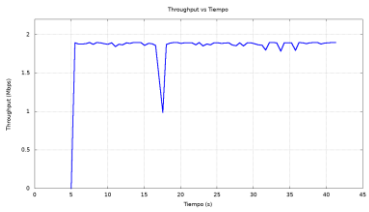

Figura 199: Throughput vs. Tiempo
TCP Vegas(14,16)–15 paq. perdidos

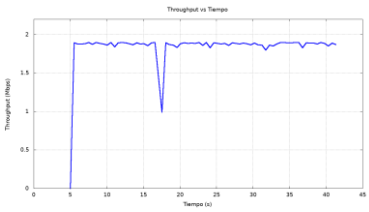

Figura 200: Throughput vs. Tiempo
TCP Vegas(15,17)–15 paq. perdidos

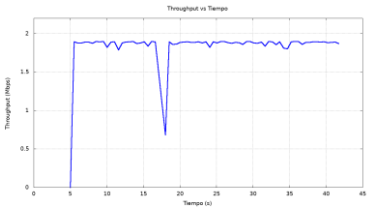

Figura 201: Throughput vs. Tiempo
TCP Vegas(16,18)–15 paq. perdidos

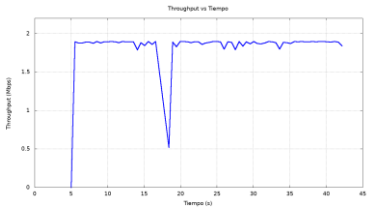

Figura 202: Throughput vs. Tiempo
TCP Vegas(18,20)–15 paq. perdidos

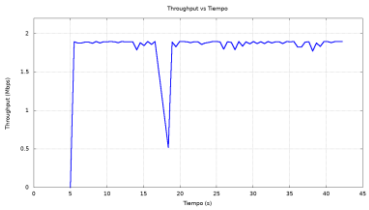

Figura 203: Throughput vs. Tiempo
TCP Vegas(21,23)–15 paq. perdidos

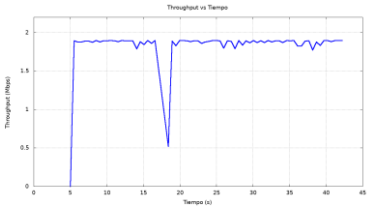

Figura 204: Throughput vs. Tiempo TCP Vegas(24,26)–15 paq. perdidos

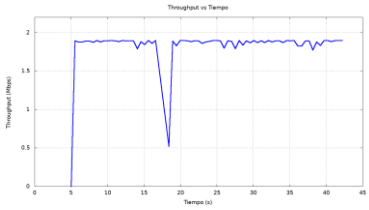

Figura 205: Throughput vs. Tiempo TCP Vegas(27,29)–15 paq. perdidos

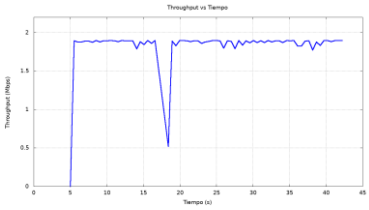

Figura 206: Throughput vs. Tiempo TCP Vegas(30,32)–15 paq. perdidos

A partir de la serie de figuras anterior, queda en evidencia los expresando anteriormente, en los alrededores de α=15 se observa que la ráfaga de 15 paquetes perdidos afecta en menor medida el rendimiento de TCP Vegas.

A continuación, se puede observar en las siguientes figuras (Figura 207 a Figura 212), el tamaño de la ventana de congestión en función del tiempo de la simulación. La serie comienza con el par de valores por defecto de los parámetros (α=1, β=3) y culmina con el último par de valores de la simulación (α=30, β=32).

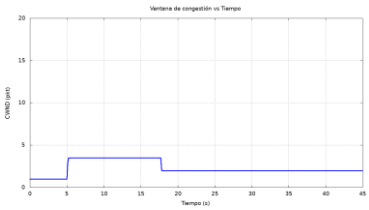

Figura 207: CWND vs. Tiempo TCP Vegas(1,3)–15 paq. perdidos

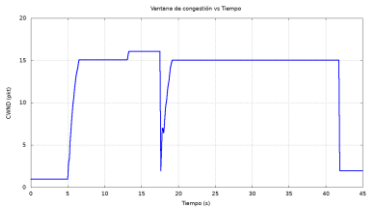

Figura 208: CWND vs. Tiempo TCP Vegas(13,15)–15 paq. perdidos

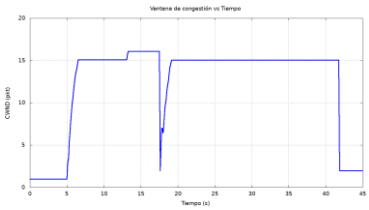

Figura 209: CWND vs. Tiempo TCP Vegas(15,17)–15 paq. perdidos

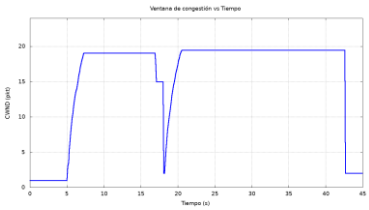

Figura 210: CWND vs. Tiempo TCP Vegas(17,19)–15 paq. perdidos

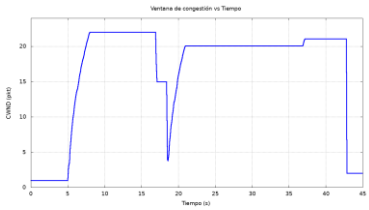

Figura 211: CWND vs. Tiempo TCP Vegas(19,21)–15 paq. perdidos

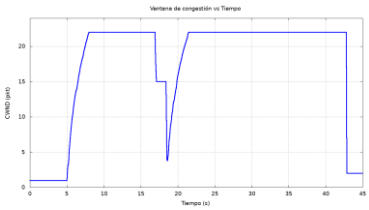

Figura 212: CWND vs. Tiempo TCP Vegas(30,32)–15 paq. perdidos

De la serie anterior, la Figura 209 corresponde al valor del par de parámetros, que, en principio, mejoran el rendimiento de TCP Vegas en este escenario, según lo analizado en la Figura 178. Se observa que, en comparación con las otras de la serie, se reduce significativamente el tiempo necesario para volver a transmitir a la misma tasa inmediata anterior a la ráfaga de paquetes perdidos, alcanzando un mayor tamaño de la ventana de congestión.

A partir de estos resultados, es posible observar que resulta aparente la existencia de una combinación de α y β que presenta la mejor respuesta a la transferencia de los datos en estas condiciones.

En las siguientes figuras (Figura 213 a Figura 218) se observa la respuesta de TCP Vegas para las distintas longitudes de error en ráfaga (0, 5, 10, 15, 20 y 25) con los parámetros α=15 y β=17.

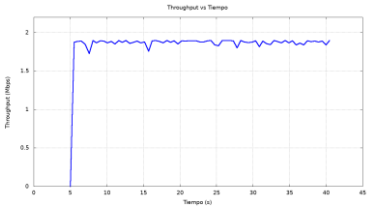

Figura 213: Throughput vs. Tiempo TCP Vegas(15,17)–0 paq. perdidos

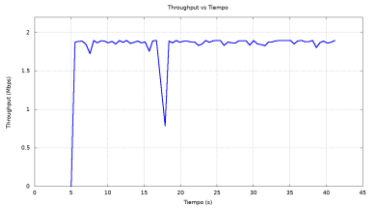

Figura 214: Throughput vs. Tiempo TCP Vegas(15,17)–5 paq. perdidos

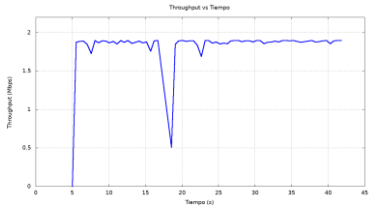

Figura 215: Throughput vs. Tiempo TCP Vegas(15,17)–10 paq. perdidos

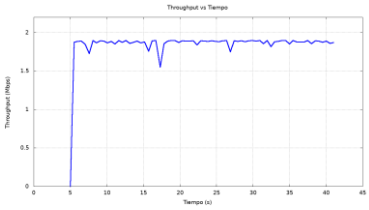

Figura 216: Throughput vs. Tiempo TCP Vegas(15,17)–15 paq. perdidos

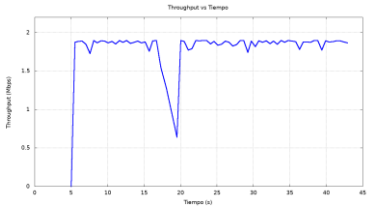

Figura 217: Throughput vs. Tiempo TCP Vegas(15,17)–20 paq. perdidos

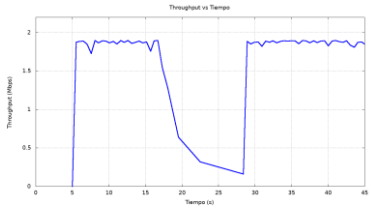

Figura 218: Throughput vs. Tiempo TCP Vegas(15,17)–25 paq. perdidos

Comparando los valores de throughput de las figuras anteriores, correspondientes a TCP Vegas con α=15 y β=17, con la secuencia de la misma métrica con α=1 y β=3 en *4.6.2 Errores en ráfaga de distinta longitud en un escenario simple* (Figura 129 a Figura 134), se puede observar que la mejora de la respuesta de esta métrica es apreciable en casi todas las longitudes de ráfaga, excepto cuando la longitud es de 5 paquetes consecutivos perdidos (Figura 130 vs. Figura 214).

Para poder realizar la comparación de las respuestas, se muestra a continuación el tiempo total de transmisión, el throughput promedio y la latencia promedio en función de la longitud de la ráfaga medida en cantidad de paquetes consecutivos perdidos. En estas graficas se superpone las respuestas de Vegas, con α=1 y β=3, Vegas, con α=15 y β=17, Reno, Westwood y CUBIC.

La siguiente figura (Figura 219) representa el tiempo total de transmisión en función de la longitud de la ráfaga de errores para las distintas variantes.

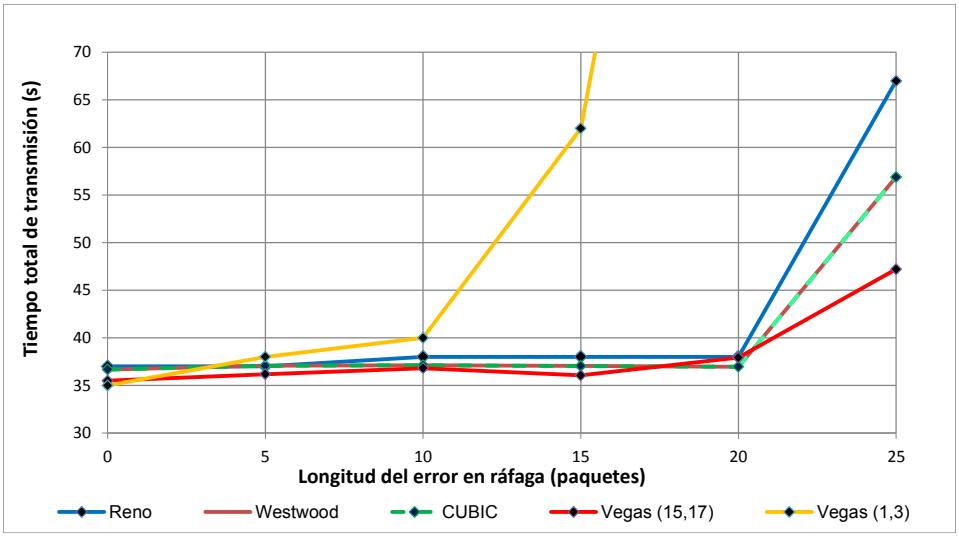

Figura 219: Tiempo Total de Transmisión vs. Longitud de la Ráfaga - TCP Reno, Westwood, CUBIC, Vegas (α=1, β=3) y Vegas (α=15, β=17)

En esta figura anterior se observa que Vegas con los parámetros por defecto comienza a estirar significativamente el tiempo necesario para concluir el envío de los datos, a partir de los 10 paquetes perdidos. Las otras variantes e inclusive Vegas con α=15 y β=17, se mantienen en aproximadamente el mismo rango de tiempo hasta los 20 paquetes perdidos. En el caso puntual de Vegas con α=15 y β=17, se puede observar que mantiene los tiempos en el mismo orden que Reno, CUBIC y Westwood. En particular, para las ráfagas de 25 paquetes de longitud, logra completar la transmisión de los datos anticipadamente al resto.

En la siguiente figura (Figura 220) se observa el valor del throughput promedio en función de la longitud de la ráfaga de paquetes perdidos. En el mismo grafico se superponen las respuestas de las variantes Reno, CUBIC, Westwood y Vegas con α=1, β=3 y con α=15, β=17. Lo que se observa en la figura es que Reno, CUBIC y Westwood logran un throughput promedio levemente más alto que Vegas con las dos configuraciones. Este valor se mantiene aproximadamente constante hasta llegar a la longitud de 20 paquetes en donde sufre una caída apreciable. En el caso de Vegas, las dos configuraciones tienen valores similares cuando no hay paquetes perdidos y, cuando se pierden 5 paquetes, es la configuración α=1 y β=3 la que logra una muy leve ventaja.

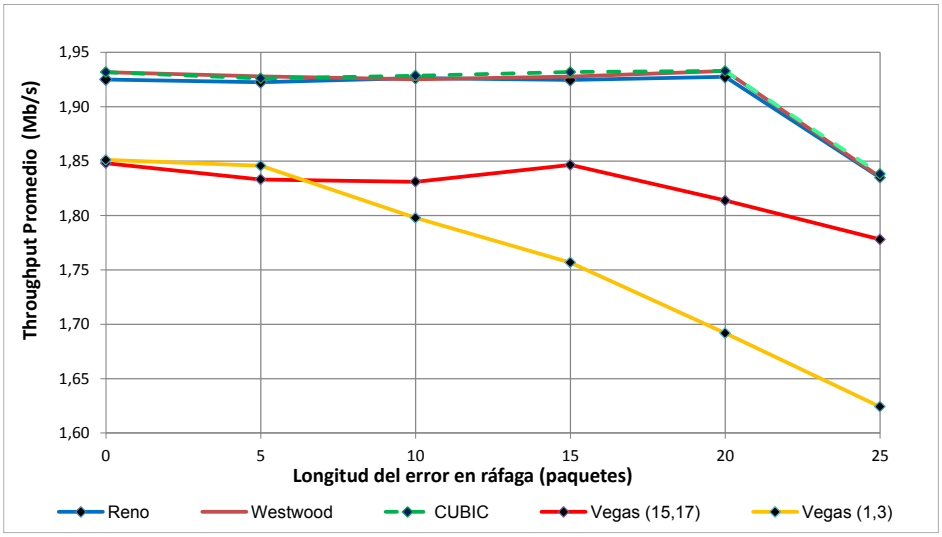

Figura 220: Throughput Promedio vs. Longitud de la Ráfaga - TCP Reno, Westwood, CUBIC, Vegas (α=1, β=3) y Vegas (α=15, β=17)

Sin embargo, a partir de ese punto, a medida que aumentan la cantidad de paquetes perdidos, comienza a caer el valor promedio del throughput. En el caso de Vegas con los parámetros α=15 y β=17 se mantiene dentro del rango del valor hasta la longitud de 15 paquetes, donde comienza un descenso leve pero constante.

En la siguiente figura (Figura 221) se observa la latencia promedio en función de la longitud de la ráfaga de paquetes perdidos para las variantes Reno, CUBIC, Westwood y Vegas con α=1, β=3 y con α=15, β=17.

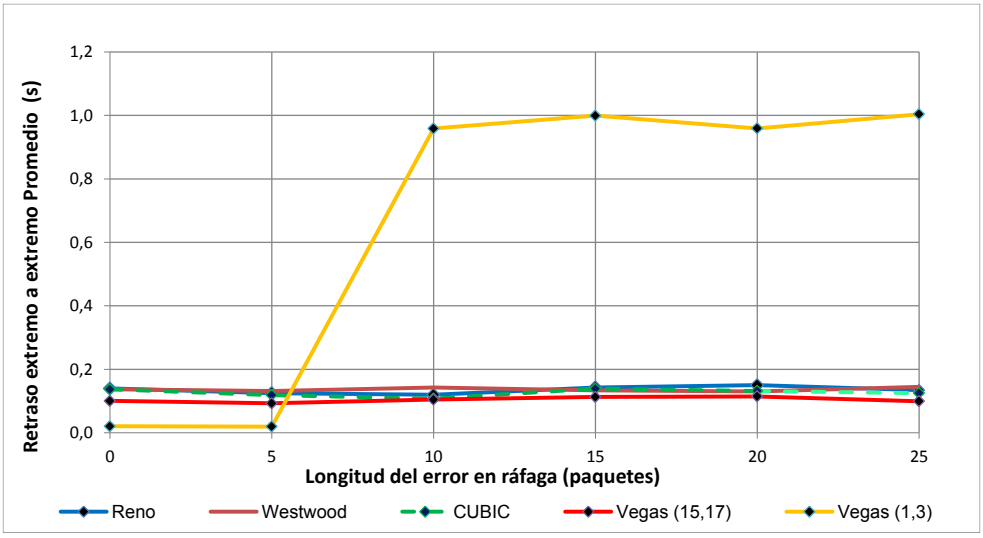

Figura 221: Latencia Promedio vs. Longitud de la Ráfaga - TCP Reno, Westwood, CUBIC, Vegas (α=1, β=3) y Vegas (α=15, β=17)

En la figura anterior se observa que la latencia promedio es del mismo orden de magnitud para Reno, CUBIC, Westwood y Vegas con α=15 y β=17. Sin embargo, Vegas con α=1 y β=3 tiene valores levemente menores hasta los 5 paquetes, para aumentar sensiblemente a partir de los 10 paquetes perdidos, en concordancia con la Figura 220, donde se observó la disminución de throughput promedio a partir de esos valores de ráfaga.

En las pruebas realizadas se observa que el comportamiento de TCP Vegas, con los parámetros α y β especificados por defecto, presenta un desempeño pobre si comparamos su tiempo total de transmisión, throughput promedio y latencia respecto a CUBIC, Westwood y Reno, cuando sometemos estos cuatro protocolos a un ensayo que posee errores en ráfaga de distinto tamaño, en el modelo planteado. Sin embargo, cuando se modifican los valores de los parámetros se observan cambios significativos. En particular, y luego de haber realizado varios ensayos, se observa que para la combinación α=15 y β=17, TCP Vegas logra equiparar el desempeño e inclusive, en algunos casos, mejorando los desempeños de CUBIC, Westwood y Reno en este escenario.

4.6.5. Contienda de mecanismos de control de congestión en un modelo heterogéneo (2 flujos)

Internet es un entorno heterogéneo y compartido por lo que el uso efectivo de la red no sólo depende de que un flujo TCP pueda aprovechar sus recursos, sino también de qué tan bien interactúa con otros flujos a través de la misma red. La eficacia no es el único parámetro importante de los algoritmos de control de congestión, también deben hacer valer el uso equitativo de los recursos compartidos, en especial el ancho de banda [148].

Es por ello que la contienda de dos flujos de datos con distintas variantes TCP, puede aportar una visión acerca de cómo se utiliza el ancho de banda bajo las distintas estrategias y ver cómo interactúan entre sí en una competencia por la utilización de este recurso. Explorando este camino, se plantea aquí un modelo simple, con dos flujos que compiten entre sí en la misma trayectoria de red heterogenia, sin otros flujos presentes. A pesar de su simpleza, incorpora la complejidad que puede presentar el accionar de los distintos mecanismos de congestión que deben lidiar con las características de una conexión

inalámbrica. A partir de esto, se realiza una comparación de la contienda entre distintos tipos de control de congestión desarrollados en algunas de las variantes del protocolo TCP.

En la Figura 222 se observa el modelo implementado en el simulador de eventos discretos. Como se puede ver, se trata de un modelo que representa un escenario con enlaces mixtos, con dos nodos fijos con sus enlaces cableados, una estación base y dos nodos con sus enlaces inalámbricos, lo que puede representar una red de acceso WLAN.

Los dos enlaces cableados que conectan los nodos fijos (0) y (1) con la estación base (2) son full dúplex, de 10 Mb/s y un retardo de 2 ms. El enlace inalámbrico que vincula la estación base (2) y los nodos inalámbricos (3) y (4) se establece en 1 Mb/s, con MAC 802.11. Los nodos inalámbricos no poseen movimiento.

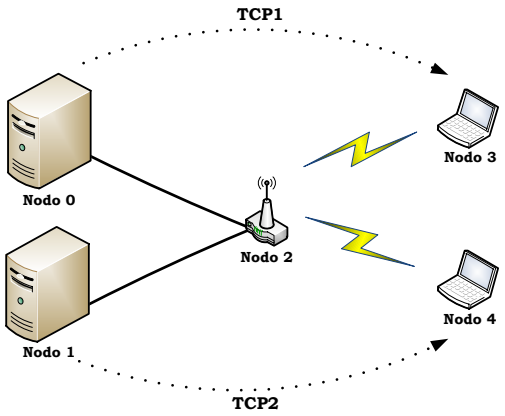

Figura 222: Modelo de dos nodos fijos y dos nodos inalámbricos

Sobre esta configuración, se realizó una confrontación de dos protocolos por vez, para evitar que existan otro tipo de influencias en el resultado.

En cada una de las simulaciones, el flujo TCP se estableció desde el nodo (0) (emisor) al nodo (3) (receptor) para la primera variante del protocolo, mientras que para la segunda variante se configuró el nodo (1) como emisor y el nodo (4) como receptor.

Se realizaron 450 simulaciones independientes, en las cuales se confrontaron distintas variantes del protocolo TCP, incluyendo cada una de ellas contra sí mismas. Las variantes utilizadas son: Reno, New Reno, SACK, FACK, Vegas,

Veno, LP (Low Priority), Westwood, HS (High Speed), H-TCP, Hybla, BIC, CUBIC, C-TCP (Compound) e Illinois.

Para cada par de protocolos se realizaron dos simulaciones distintas, intercambiando el orden de inicio de cada flujo, separados por 10 segundos. El primer flujo comienza a trasmitir a los 5 segundos de iniciada la simulación y termina a los 105 segundos, mientras que el segundo flujo comienza a los 15 segundos y termina a los 120 segundos.

De cada una de estas simulaciones se obtuvo el throughput promedio y el Packet Delivery Ratio (PDR). El throughput promedio se obtuvo sumando todos los paquetes salientes del nodo emisor, en el intervalo que va desde los 40 a los 100 segundos. El PDR para cada uno de los flujos, se obtuvo calculando el cociente entre los paquetes recibidos correctamente en el receptor y el total de los enviados en el mismo intervalo de tiempo. Además, se obtuvieron los valores instantáneos del throughput y del tamaño de la ventana de congestión para cada uno de esos flujos.

El objetivo de tomar las muestras solo en el intervalo de tiempo 40 a 100 segundos, es analizar la utilización de los recursos de la red, por parte de los dos flujos que compiten, en un estado estacionario relativo, y de esta manera, evitar las oscilaciones típicas de un estado transitorio, para analizar el comportamiento en un trayecto común de una comunicación establecida.

Para la comparación de las distintas variantes de TCP en competencia se utilizó una métrica resultante del producto del throughput promedio y el PDR correspondiente para cada una de las simulaciones y para cada uno de los flujos expresado por la siguiente ecuación (Ecuación 64).

$$\mathrm{TPP} = \mathrm{Thoughput\ promedio} * \mathrm{PDR} \qquad \textbf{Ecuación 64}$$

A partir de la heterogeneidad de Internet, en el que conviven distintas variantes de TCP y que un flujo puede atravesar distintos tipos de enlaces desde el emisor hasta el receptor, resulta interesante la evaluación del comportamiento de las variantes propuestas en este escenario.

Resultados

Sobre el modelo de la Figura 222 se realizaron los ensayos que posibilitaron conformar dos matrices, que representan la confrontación de 15 protocolos. Los resultados pueden observarse en la Tabla 5 y la Tabla 6, donde las filas y columnas representan a las variantes del protocolo TCP analizadas. En ambas tablas, las filas representan al protocolo que inicia primero la trasmisión y, por lo tanto, las columnas al que lo hace en segundo lugar.

En la primera matriz (Tabla 5), cada casilla expresa el valor de TPP de la variante del protocolo TCP de la fila, en su contienda con la variante TCP de la columna.

Mientras que en la segunda matriz (Tabla 6), el valor de celda expresa el valor de TPP del protocolo correspondiente a la columna, es decir al que inicia su transmisión en segunda instancia.

	Bic	Compound	Cubic	Fack	Highspeed	HTCP	Hybla	Illinois	LowPriority	NewReno	Reno	Sack	Tahoe	Vegas	Veno	Westwood	YeAH
Bic	0,311	0,317	0,315	0,314	0,317	0,319	0,274	0,314	0,317	0,318	0,314	0,314	0,318	0,538	0,316	0,316	0,280
Compound	0,316	0,316	0,317	0,315	0,316	0,316	0,256	0,315	0,316	0,313	0,315	0,315	0,313	0,520	0,315	0,314	0,287
Cubic	0,317	0,314	0,315	0,310	0,314	0,315	0,270	0,318	0,314	0,315	0,312	0,312	0,315	0,564	0,315	0,314	0,293
Fack	0,320	0,317	0,316	0,322	0,317	0,312	0,255	0,314	0,317	0,315	0,322	0,322	0,315	0,562	0,315	0,318	0,303
Highspeed	0,316	0,316	0,317	0,315	0,316	0,316	0,256	0,315	0,316	0,313	0,315	0,315	0,313	0,520	0,315	0,314	0,287
HTCP	0,311	0,317	0,315	0,314	0,317	0,319	0,273	0,314	0,317	0,318	0,314	0,314	0,318	0,538	0,316	0,316	0,279
Hybla	0,339	0,347	0,345	0,367	0,347	0,345	0,296	0,382	0,347	0,389	0,353	0,356	0,389	0,571	0,445	0,347	0,344
Illinois	0,320	0,316	0,316	0,317	0,316	0,319	0,264	0,315	0,316	0,315	0,317	0,317	0,315	0,539	0,319	0,316	0,266
LowPriority	0,316	0,316	0,317	0,315	0,316	0,316	0,256	0,315	0,316	0,313	0,315	0,315	0,313	0,520	0,315	0,314	0,287
NewReno	0,312	0,315	0,318	0,315	0,315	0,318	0,230	0,316	0,315	0,318	0,318	0,318	0,318	0,444	0,321	0,317	0,305
Reno	0,320	0,317	0,316	0,322	0,317	0,312	0,269	0,314	0,317	0,315	0,322	0,322	0,315	0,562	0,315	0,318	0,264
Sack	0,320	0,317	0,316	0,322	0,317	0,312	0,265	0,314	0,317	0,315	0,322	0,322	0,315	0,562	0,315	0,318	0,307
Tahoe	0,312	0,315	0,318	0,315	0,315	0,318	0,230	0,316	0,315	0,318	0,318	0,318	0,318	0,444	0,321	0,317	0,305
Vegas	0,061	0,063	0,063	0,063	0,063	0,062	0,034	0,068	0,063	0,062	0,063	0,063	0,062	0,313	0,062	0,066	0,049
Veno	0,317	0,313	0,320	0,317	0,313	0,306	0,213	0,315	0,313	0,318	0,317	0,317	0,318	0,590	0,318	0,317	0,258
Westwood	0,315	0,314	0,317	0,319	0,314	0,322	0,290	0,315	0,314	0,313	0,319	0,319	0,313	0,564	0,313	0,322	0,263
YeAH	0,335	0,318	0,311	0,320	0,318	0,317	0,311	0,317	0,318	0,317	0,315	0,313	0,317	0,479	0,317	0,328	0,329

Tabla 5: TPP del flujo que inicia en primer término

Si bien el valor de esta métrica (TPP) presenta algún valor de comparación en la contienda por los recursos de la red, no expresa en forma cabal el comportamiento de los protocolos, ni el de los algoritmos de control de congestión. Por tal motivo, se presentan a continuación gráficos en donde

pueden apreciarse las evoluciones del throughput de los flujos y sus respectivas variaciones de ventana de congestión.

	Bic	Compound	Cubic	Fack	Highspeed	HTCP	Hybla	Illinois	LowPriority	NewReno	Reno	Sack	Tahoe	Vegas	Veno	Westwood	YeAH
Bic	0,311	0,318	0,316	0,317	0,318	0,317	0,355	0,312	0,318	0,318	0,317	0,317	0,318	0,106	0,318	0,314	0,349
Compound	0,316	0,317	0,316	0,316	0,317	0,321	0,372	0,316	0,317	0,315	0,316	0,316	0,315	0,126	0,314	0,315	0,340
Cubic	0,319	0,314	0,313	0,314	0,314	0,319	0,352	0,315	0,314	0,316	0,314	0,313	0,316	0,086	0,317	0,313	0,337
Fack	0,321	0,319	0,317	0,325	0,319	0,315	0,378	0,315	0,319	0,315	0,325	0,325	0,315	0,086	0,314	0,319	0,331
Highspeed	0,316	0,317	0,316	0,316	0,317	0,321	0,372	0,316	0,317	0,315	0,316	0,316	0,315	0,126	0,314	0,315	0,340
HTCP	0,311	0,318	0,316	0,317	0,318	0,317	0,354	0,312	0,318	0,318	0,317	0,317	0,318	0,106	0,318	0,314	0,346
Hybla	0,285	0,279	0,281	0,262	0,279	0,290	0,334	0,248	0,279	0,249	0,271	0,280	0,249	0,078	0,184	0,281	0,285
Illinois	0,317	0,320	0,312	0,317	0,320	0,317	0,364	0,313	0,320	0,315	0,317	0,317	0,315	0,104	0,319	0,317	0,361
LowPriority	0,316	0,317	0,316	0,316	0,317	0,321	0,372	0,316	0,317	0,315	0,316	0,316	0,315	0,126	0,314	0,315	0,340
NewReno	0,314	0,315	0,316	0,317	0,315	0,316	0,405	0,315	0,315	0,319	0,319	0,319	0,319	0,191	0,317	0,313	0,334
Reno	0,321	0,319	0,317	0,325	0,319	0,315	0,355	0,315	0,319	0,315	0,325	0,325	0,315	0,086	0,314	0,319	0,369
Sack	0,321	0,319	0,317	0,325	0,319	0,315	0,369	0,315	0,319	0,315	0,325	0,325	0,315	0,086	0,314	0,319	0,326
Tahoe	0,314	0,315	0,316	0,317	0,315	0,316	0,405	0,315	0,315	0,319	0,319	0,319	0,319	0,191	0,317	0,313	0,334
Vegas	0,591	0,594	0,588	0,590	0,594	0,586	0,625	0,583	0,594	0,590	0,590	0,590	0,590	0,310	0,592	0,589	0,608
Veno	0,316	0,312	0,317	0,316	0,312	0,323	0,418	0,318	0,312	0,319	0,316	0,316	0,319	0,061	0,315	0,319	0,371
Westwood	0,313	0,314	0,315	0,315	0,314	0,319	0,346	0,315	0,314	0,315	0,315	0,315	0,315	0,086	0,315	0,314	0,360
YeAH	0,300	0,316	0,313	0,321	0,316	0,317	0,315	0,317	0,316	0,316	0,316	0,314	0,316	0,158	0,317	0,299	0,305

Tabla 6: TPP del flujo que inicia en segundo término

Analizando las representaciones de los valores instantáneos, se puede apreciar, en forma cualitativa, la evolución en el tiempo de los flujos de datos. Por otro lado, a través de las variaciones de la ventana de congestión, podemos apreciar cómo los algoritmos de control de congestión intentan desplegar su estrategia a fin de adaptarse al ancho de banda disponible.

En las siguientes figuras (Figura 223 a Figura 226) se puede observar el valor del throughput y de la ventana de congestión para la confrontación de las variantes *CUBIC* e *Hybla*, cuando comienza a transmitir en primer lugar CUBIC (Figura 223 y Figura 225) y cuando lo hace Hybla (Figura 224 y Figura 226).

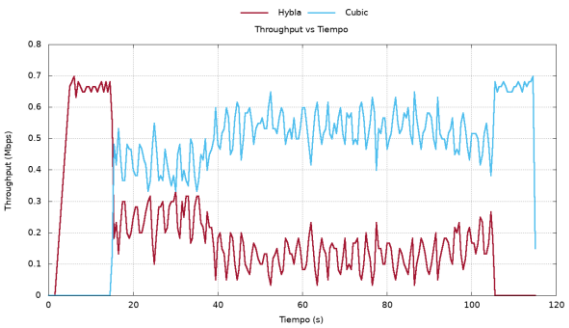

Figura 223: Throughput vs. Tiempo – CUBIC vs. Hybla

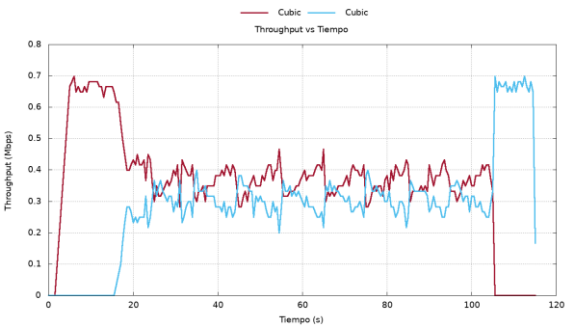

Figura 224: Throughput vs. Tiempo – Hybla vs. CUBIC

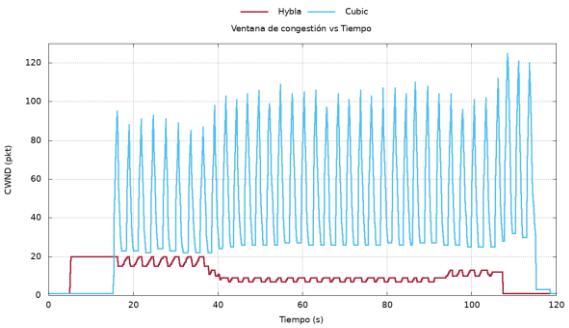

Figura 225: CWND vs. Tiempo – CUBIC vs. Hybla

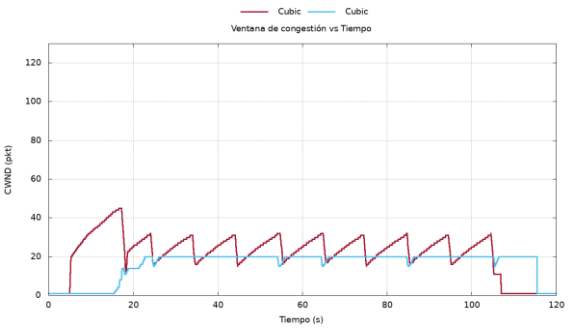

Figura 226: CWND vs. Tiempo – Hybla vs. CUBIC

En las figuras anteriores, se puede observar que tanto el comportamiento del throughput como la evolución del tamaño de la ventana de congestión difieren significativamente si es CUBIC o Hybla la que comienza a transmitir en primer lugar. Esto produce necesariamente distintas formas de distribución de los recursos de red. De las tablas anteriores se obtiene que cuando es CUBIC el que comienza a transmitir primero, su valor de TPP es de 0,270 (Tabla 5) y el de Hybla 0,352 (Tabla 6). Análogamente, cuando es Hybla el que trasmite primero su TPP es de 0,281 (Tabla 5) mientras que el de CUBIC es 0,345 (Tabla 6). En las siguientes figuras (Figura 227 a Figura 232), se observa un comportamiento similar en la contienda contra otras de las variantes ensayadas.

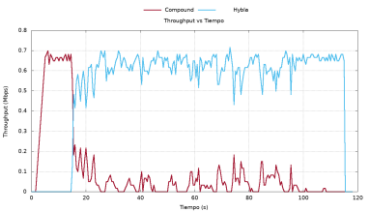

Figura 227: Throughput vs. Tiempo – Compound vs. Hybla

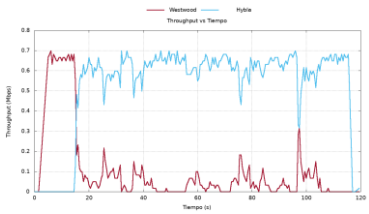

Figura 228: Throughput vs. Tiempo – Westwood vs. Hybla

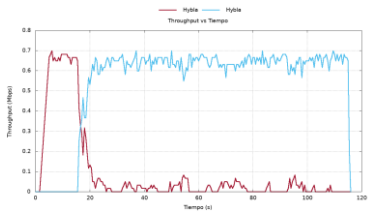

Figura 229: Throughput vs. Tiempo – Hybla vs. Hybla

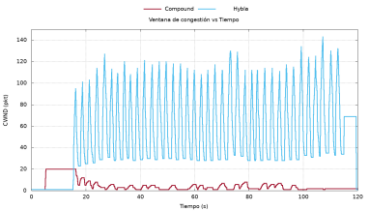

Figura 230: CWND vs. Tiempo – Compound vs. Hybla

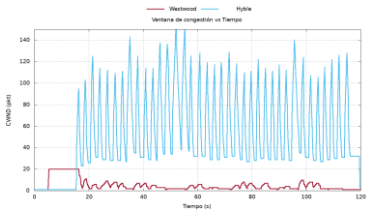

Figura 231: CWND vs. Tiempo – Westwood vs. Hybla

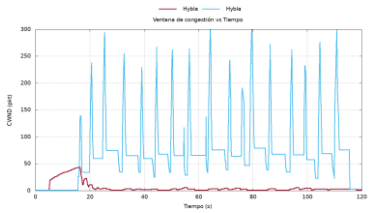

Figura 232: CWND vs. Tiempo – Hybla vs. Hybla

Como se puede observar en las figuras anteriores, el protocolo Hybla tiende a acaparar el ancho de banda disponible intentando transmitir la mayor cantidad de datos posibles en detrimento de las posibilidades del otro protocolo. Este

comportamiento, se pueden observar las figuras la evolución del tamaño de la ventana de congestión en función del tiempo, lo que permiten comprender como actúan los algoritmos de control de congestión.

En la Figura 225 y la Figura 226, se puede observar el modo en que los algoritmos de control de congestión de ambas variantes ajustan el tamaño de la cwnd a lo largo del tiempo. Se observa que la causa de esa preponderancia que tiene el protocolo Hybla sobre CUBIC, está reflejada por la mayor agresividad del primero, debido a los constantes intentos de aumentar su tasa de envío, aumentando el tamaño de cwnd. Esto solo se ve limitado al llegar a un estado de congestión. Este comportamiento agresivo de Hybla provoca estados de congestión periódica que influyen sobre el control de congestión de CUBIC, que también reacciona ante ellos y tiende a disminuir su cwnd a fin de no causar el colapso en el canal.

Las simulaciones de Westwood vs. Hybla (Figura 228) muestran la agresividad de Hybla para capturar el ancho de banda. Este comportamiento se verifica en la Figura 231, donde se observa la evolución su cwnd.

En la Figura 229 se observa el throughput y en la Figura 232 el tamaño de la ventana de congestión en función del tiempo, cuando compiten por los recursos de la red dos flujos con idéntico control de congestión Hybla. A pesar de utilizar los mismos algoritmos, se puede observar en las figuras que una de los flujos relega al otro en el uso del ancho de banda.

Las siguientes figuras (Figura 233 a Figura 236) representan el throughput y el tamaño de la ventana de congestión en función del tiempo de la variante de TCP Vegas en contienda con TCP Reno.

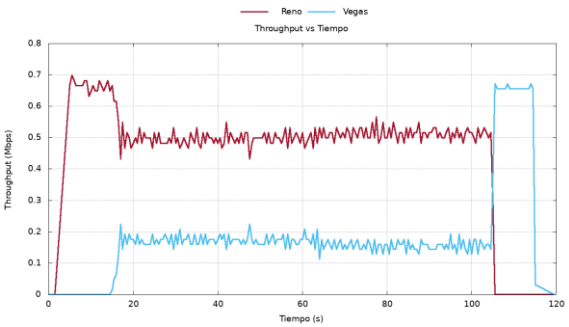

Figura 233: Throughput vs. Tiempo – Reno vs. Vegas

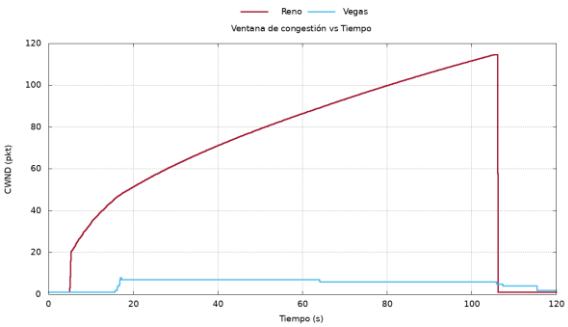

Figura 234: CWND vs. Tiempo – Reno vs. Vegas

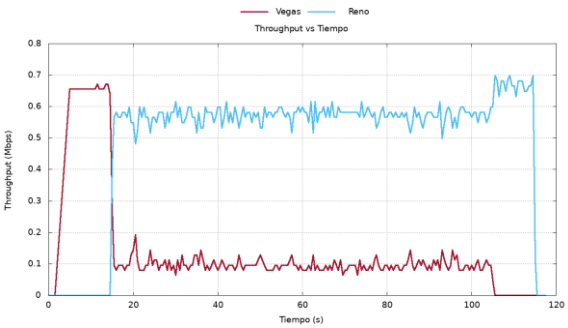

Figura 235: Throughput vs. Tiempo – Vegas vs. Reno

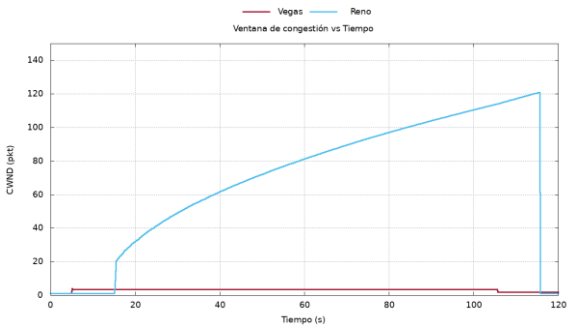

Figura 236: CWND vs. Tiempo – Vegas vs. Reno

En las figuras anteriores se puede observar que no importa en qué momento comienza la transmisión el flujo que utiliza la variante de TCP Vegas, siempre obtiene una porción menor de los recursos disponibles de la red. Cuando Reno es el que comienza primero, el valor de TPP es 0,562 y el de Vegas es 0,086. Análogamente, cuando es Vegas el que comienza la transmisión en primer término su TPP es 0,063 y el de Reno 0,590. En las figuras de throughput en función del tiempo (Figura 233 y Figura 235) se observa con claridad que es TCP Reno el que logra utilizar la mayor parte del ancho de banda disponible.

Debido a la naturaleza proactiva de los algoritmos de control de congestión de TCP Vegas, este comportamiento se observa en todas las contiendas ensayadas. Este comportamiento queda en evidencia en las siguientes figuras (Figura 237 a Figura 242) donde se observa la evolución del throughput en función del tiempo de la simulación, en la contienda de TCP Vegas contra Compound, CUBIC y Westwood, intercambiando el orden de comienzo.

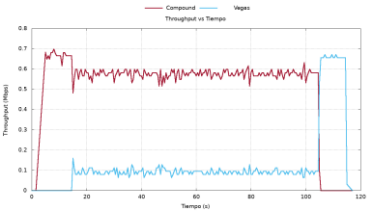

Figura 237: Throughput vs. Tiempo – Compound vs. Vegas

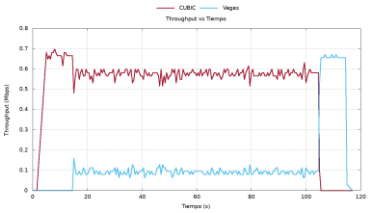

Figura 238: Throughput vs. Tiempo – CUBIC vs. Vegas

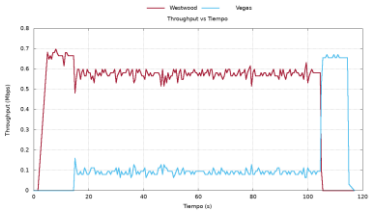

Figura 239: Throughput vs. Tiempo – Westwood vs. Vegas

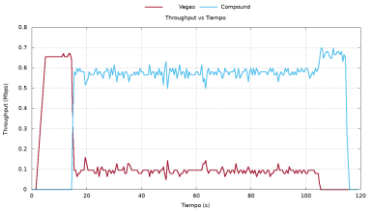

Figura 240: Throughput vs. Tiempo – Vegas vs. Compound

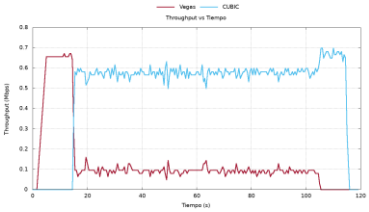

Figura 241: Throughput vs. Tiempo – Vegas vs. CUBIC

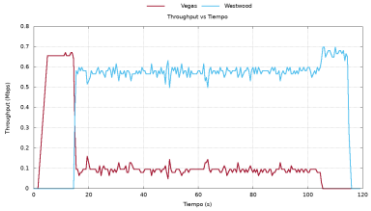

Figura 242: Throughput vs. Tiempo – Vegas vs. Westwood

En las figuras anteriores, TCP Vegas, aun siendo el que inicia en primera instancia, resulta particularmente sensible respecto al resto de las variantes. Se observa en las gráficas de throughput de Compound vs. Vegas (Figura 237) o Vegas vs. CUBIC (Figura 241), donde comenzando a transmitir primero o segundo, TCP Vegas resigna rápidamente si participación en el uso del ancho de banda del canal. Por lo tanto, en todos los casos analizados Vegas siempre recibe una porción menor de ancho de banda. Esto se confirma observando los valores de TPP en la Tabla 5 y la Tabla 6.

A partir de lo observado en la Figura 229 donde 2 flujos, con el mismo control de congestión (Hybla) que comparten el camino en una red heterogénea, no repartían equitativamente el ancho de banda, resulta relevante verificar este comportamiento en otras de las variantes ensayadas. A continuación, se observa, en las siguientes figuras (Figura 243 a Figura 247), los gráficos del valor del throughput en función del tiempo de la contienda de 2 flujos con el mismo control de congestión. Como se observa en las figuras, en todos los casos presentados, Vegas, Compound, CUBIC, Reno y Westwood, se produce un reparto equitativo del ancho de banda cuando compiten contra sí mismos.

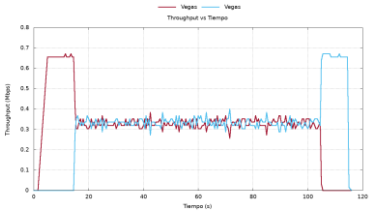

Figura 243: Throughput vs. Tiempo – Vegas vs. Vegas

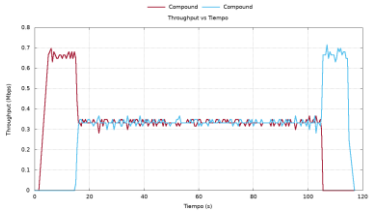

Figura 244: Throughput vs. Tiempo – Compound vs. Compound

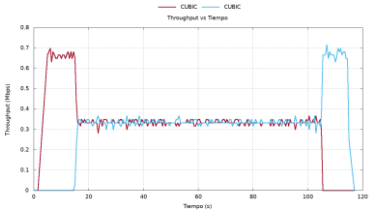

Figura 245: Throughput vs. Tiempo – CUBIC vs. CUBIC

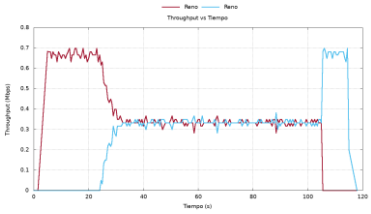

Figura 246: Throughput vs. Tiempo – Reno vs. Reno

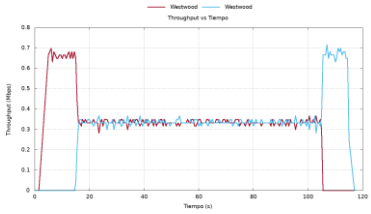

Figura 247: Throughput vs. Tiempo – Westwood vs. Westwood

Las siguientes figuras (Figura 248 a Figura 255) representan el throughput y el tamaño de la ventana de congestión en función del tiempo de algunas de las

contiendas que se ensayaron y que dan como resultado una distribución aproximadamente equitativa del ancho de banda disponible.

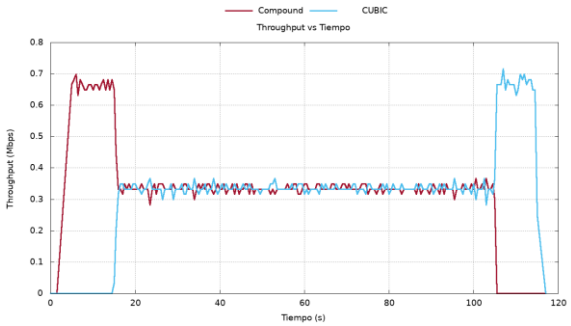

Figura 248: Throughput vs. Tiempo
Compound vs. CUBIC

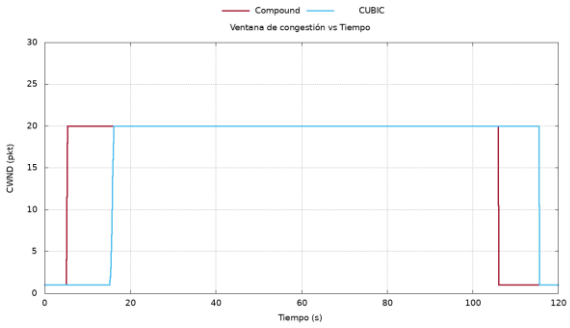

Figura 249: CWND vs. Tiempo
Compound vs. CUBIC

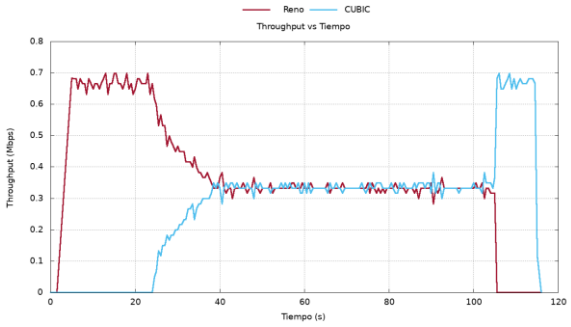

Figura 250: Throughput vs. Tiempo
Reno vs. CUBIC

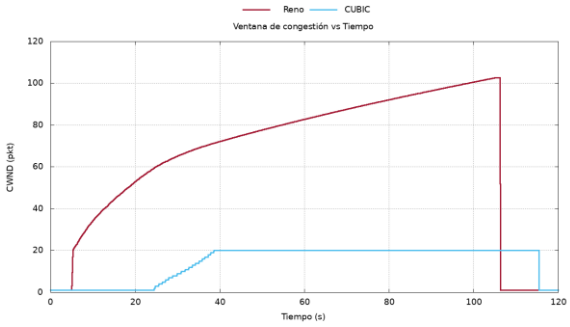

Figura 251: CWND vs. Tiempo
Reno vs. CUBIC

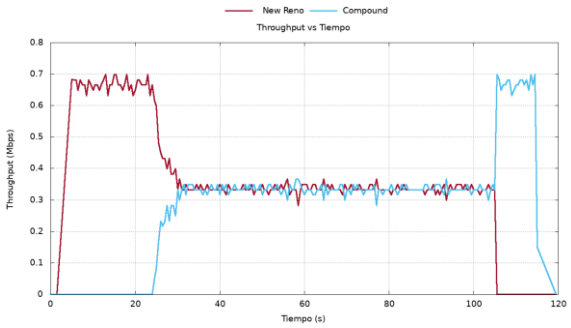

Figura 252: Throughput vs. Tiempo
NewReno vs. Compound

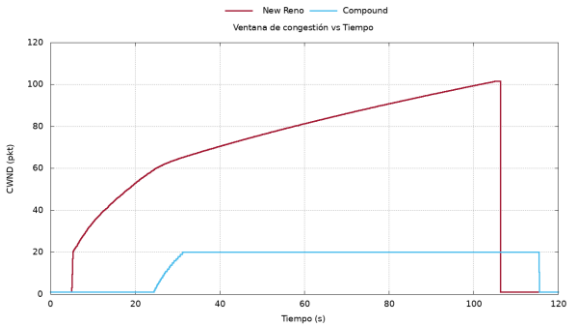

Figura 253: CWND vs. Tiempo
NewReno vs. Compound

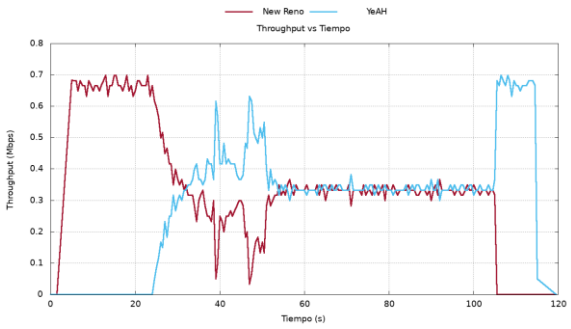

Figura 254: Throughput vs. Tiempo
NewReno vs. Yeah

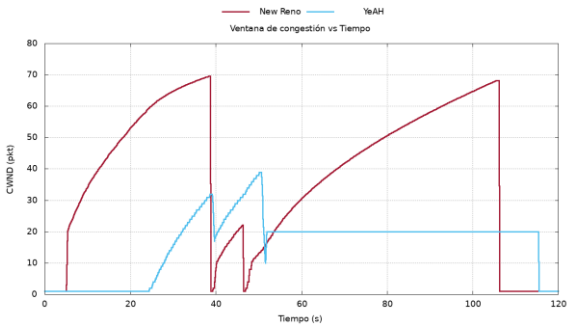

Figura 255: CWND vs. Tiempo
NewReno vs. Yeah

En las figuras anteriores, se observa que cuando el primer flujo comienza a transmitir, utiliza la totalidad del ancho de banda disponible. Al iniciar el segundo flujo se producen fluctuaciones donde ninguno parece ejercer un predominio marcado sobre el otro y, paulatinamente, se llega a una distribución equitativa del ancho de banda.

Este comportamiento se da en la gran mayoría de las confrontaciones que se observaron. Si bien el comportamiento es similar, no son exactamente iguales. La diferencia fundamental se da en la forma en que llegan al estado de equilibrio. Se puede observar la forma en la que alcanza el punto de estabilidad en la Figura 250, en contraste con lo que se observa en la Figura 248.

Se puede observar que existen variantes del protocolo TCP que pueden coexistir mientras que otras no lo pueden hacer. Para poner en evidencia esta situación, es interesante observar los resultados que arrojan las medidas de TPP obtenidas (Tabla 5 y Tabla 6) Estos valores muestran que existen variantes del protocolo TCP que captan la mayor parte del ancho de banda para transmitir los datos, mientras que otras muestran ceden su participación en el uso de los recursos.

Analizando las distintas pruebas de confrontación, vemos que el algoritmo de control de congestión que posee Vegas no le permite sostener un uso de ancho de banda justo cuando compite con los restantes protocolos seleccionados para realizar las pruebas. Así Vegas resigna ancho de banda y pasa a un segundo plano. En el otro extremo, protocolos como Hybla, Illinois y aquellos destinados a redes de alta velocidad, tienden a dominar a sus contrapartes en los ensayos e intentan capturar el mayor ancho de banda posible. De esta manera, se tiene una primera aproximación de la agresividad que pueden manifestar ciertos protocolos y la manera en que el algoritmo de congestión intenta obtener el mayor ancho de banda disponible para transmitir los datos.

4.6.6. Equidad: Contienda de mecanismos de control de congestión

El estudio de la interacción de dos o más flujos que compiten por los recursos de una red compartida, ha ayudado a comprender cómo los mecanismos de

control de congestión de las distintas variantes de protocolo TCP interaccionan entre sí. En estos casos, resulta fundamental entender cómo esas interacciones se producen y determinar, bajo determinados parámetros, si la convivencia de estos flujos es posible. Por otro lado, resulta interesante analizar qué preponderancia pueden tener unos sobre otros, en referencia al uso del ancho de banda. A partir de esto, se analizan esos aspectos y se intenta establecer cómo es que esa convivencia se logra, determinando de qué forma dos flujos de datos del protocolo TCP, llegan a un estado de equilibrio sobre el canal, teniendo en cuenta que en la actualidad las redes presentan escenarios que implican tener trayectos heterogéneos [149].

Es por todo esto que el análisis de la contienda entre dos flujos de datos, cuando ambos poseen los mismos algoritmos de control de congestión o bien distintos algoritmos, puede aportar una visión acerca de cómo es utilizado el ancho de banda cuando dos estrategias confrontan. El estudio de la confrontación de protocolos bajo estas condiciones da origen a lo que se denomina como equidad y resulta interesante explorar en detalle las variantes y efectos derivados de este fenómeno. La equidad es una medida de cómo un flujo TCP, afecta al resto de los flujos y, cómo es afectado por el resto de los flujos, en términos utilización del ancho de banda.

Explorando este camino, se plantea aquí un modelo simple, con dos flujos que compiten entre sí y comparten un nodo que genera un efecto cuello de botella, en un escenario heterogéneo. Este modelo puede representar una red de acceso inalámbrica.

Para realizar los ensayos se utilizó el mismo modelo de la Figura 222. En cada una de las simulaciones se establecieron los flujos TCP, la configuración del modelo, los tiempos y parámetros de las simulaciones de la misma forma que en *4.6.5 Contienda de mecanismos de control de congestión en un modelo heterogéneo (2 flujos)*. Sin embargo, en esas simulaciones se utilizó para el estudio de la contienda de distintas variantes de TCP la métrica TPP definida por la Ecuación 64, cuyos resultados se volcaron en la Tabla 5 y la Tabla 6. Esta métrica permitió realizar una evaluación de cómo se reparten dos flujos los recursos de la red sobre un modelo heterogéneo y proporcionó un valor de cómo cada variante interviniente utiliza un porcentaje del ancho de banda. Sin

embargo, también quedó en evidencia, que esta métrica por sí sola no expresa en forma cabal el comportamiento de los protocolos, es decir, no aporta más datos de cómo es la forma en que se llega a un estado de equilibro.

Por lo anteriormente expuesto, es conveniente recurrir a otras métricas que puedan brindar mayor información al respecto y, de ese modo, permitan evaluar este comportamiento de una manera más exhaustiva.

La medida de la equidad entre un conjunto de flujos que se disputan el ancho de banda, se puede definir por el *Índice de Equidad (IE)* expresado por la siguiente ecuación (Ecuación 65).

$$f(x1, x2, x3 \ldots . xn) = \frac{\left[\sum_{j=1}^{n} xj\right]^2}{n \sum_{j=1}^{n} (xj)^2} \qquad \textbf{Ecuación 65}$$

Donde, f es el Índice de equidad (IE), 0 ≤ f ≤ 1, n es el número de flujos simultáneos y x_j es el throughput del flujo j.

Por otro lado, si se consideran dos flujos, donde un flujo inicia primero, y un tiempo después inicia el segundo flujo, se define el *Tiempo de Convergencia (TC)* como tiempo que le toma al segundo flujo alcanzar el valor equivalente al 80% del valor del throughput instantáneo del primer flujo en ese momento. El tiempo de convergencia se calcula utilizando la siguiente ecuación (Ecuación 66).

$$Tc = T_{80} - T_{f2} \qquad \textbf{Ecuación 66}$$

Donde, Tc es el tiempo de convergencia (TC), T_{80} es el tiempo que le toma al segundo flujo para llegar al 80% del throughput del primer flujo y T_{f2} es el tiempo donde comienza a transmitir el segundo flujo.

A partir de estas métricas, se realizaron una serie de nuevos ensayos y se obtuvieron los resultados que se presentan a continuación.

Resultados

Con el objetivo de cuantificar los resultados de confrontación de las estrategias de los algoritmos de control de congestión de las 17 variantes que se ensayaron, se construyeron otras dos matrices que se observan en la Tabla 7 y en la Tabla 8 a partir de las métricas definidas por la Ecuación 65 y la Ecuación

66 respectivamente. Estas tablas representan el Índice de Equidad y el Tiempo de Convergencia.

	Bic	Compound	Cubic	Fack	Highspeed	HTCP	Hybla	Illinois	LowPriority	NewReno	Reno	Sack	Tahoe	Vegas	Veno	Westwood	YeAH
Bic	0,9994	0,9994	0,9993	0,9999	0,9994	0,9996	0,9983	0,9990	0,9994	0,9999	0,9999	0,9999	0,9999	0,7773	0,9987	0,9994	0,9959
Compound	0,9992	0,9992	0,9992	0,9998	0,9992	0,9997	0,9960	0,9990	0,9992	0,9999	0,9998	0,9998	0,9999	0,8054	0,9981	0,9993	0,9986
Cubic	0,9993	0,9993	0,9989	0,9999	0,9993	0,9996	0,9988	0,9990	0,9993	0,9999	0,9999	0,9998	0,9999	0,7490	0,9986	0,9992	0,9986
Fack	0,9995	0,9996	0,9994	0,9999	0,9996	0,9998	0,9903	0,9994	0,9996	0,9999	0,9999	0,9999	0,9999	0,7468	0,9984	0,9996	0,9996
Highspeed	0,9992	0,9992	0,9992	0,9998	0,9992	0,9997	0,9960	0,9990	0,9992	0,9999	0,9998	0,9998	0,9999	0,8054	0,9981	0,9993	0,9986
HTCP	0,9994	0,9994	0,9993	0,9999	0,9994	0,9996	0,9990	0,9990	0,9994	0,9999	0,9999	0,9999	0,9999	0,7773	0,9987	0,9994	0,9972
Hybla	0,9909	0,9858	0,9883	0,9783	0,9858	0,9942	0,9980	0,9595	0,9858	0,9704	0,9756	0,9889	0,9704	0,7247	0,8989	0,9855	0,9860
Illinois	0,9994	0,9996	0,9991	0,9999	0,9996	0,9997	0,9967	0,9993	0,9996	0,9999	0,9999	0,9999	0,9999	0,7747	0,9989	0,9996	0,9933
LowPriority	0,9992	0,9992	0,9992	0,9998	0,9992	0,9997	0,9960	0,9990	0,9992	0,9999	0,9998	0,9998	0,9999	0,8054	0,9981	0,9993	0,9986
NewReno	0,9994	0,9991	0,9992	0,9882	0,9991	0,9994	0,9848	0,9988	0,9991	0,9872	0,9872	0,9872	0,9872	0,9035	0,9985	0,9991	0,9999
Reno	0,9995	0,9996	0,9994	0,9999	0,9996	0,9998	0,9985	0,9994	0,9996	0,9999	0,9999	0,9999	0,9999	0,7468	0,9984	0,9996	0,9923
Sack	0,9995	0,9996	0,9994	0,9999	0,9996	0,9998	0,9974	0,9994	0,9996	0,9999	0,9999	0,9999	0,9999	0,7468	0,9984	0,9996	0,9998
Tahoe	0,9994	0,9991	0,9992	0,9882	0,9991	0,9994	0,9848	0,9988	0,9991	0,9872	0,9872	0,9872	0,9872	0,9035	0,9985	0,9991	0,9999
Vegas	0,7051	0,7040	0,7141	0,7031	0,7040	0,7071	0,6964	0,7136	0,7040	0,7045	0,7031	0,7031	0,7045	0,9999	0,7068	0,7099	0,6799
Veno	0,9994	0,9994	0,9994	0,9999	0,9994	0,9999	0,9793	0,9995	0,9994	1,0000	0,9999	0,9999	1,0000	0,7044	0,9982	0,9996	0,9878
Westwood	0,9989	0,9992	0,9992	0,9999	0,9992	0,9994	0,9995	0,9990	0,9992	0,9999	0,9999	0,9999	0,9999	0,7433	0,9986	0,9990	0,9943
YeAH	0,9923	0,9962	0,9965	0,9993	0,9962	0,9979	0,9961	0,9958	0,9962	0,9977	0,9972	0,9991	0,9977	0,8421	0,9943	0,9922	0,9984

Tabla 7: Índice de Equidad (IE)

En la Tabla 7 se pueden apreciar los valores del Índice de Equidad que ponen en evidencia la forma en que los dos flujos negocian el uso del ancho de banda disponible. De acuerdo a la definición, este valor varía entre 0 y 1. De tal manera que, cuando esta métrica toma el valor de 1, los dos protocolos han utilizado en ancho de banda de forma equitativa, mientras que cuanto más se acerca este valor a 0 más inequitativo es la distribución de ancho de banda.

	Bic	Compound	Cubic	Fack	Highspeed	HTCP	Hybla	Illinois	LowPriority	NewReno	Reno	Sack	Tahoe	Vegas	Veno	Westwood	YeAH
Bic	5,90	5,40	5,30	1,40	5,40	3,30	8,90	5,40	5,40	1,80	1,40	1,40	1,80	N/C	7,80	5,80	5,90
Compound	6,40	5,90	5,45	1,90	5,90	3,90	8,90	5,90	5,90	2,40	1,90	1,90	2,40	N/C	10,45	6,45	6,90
Cubic	6,40	3,45	5,40	1,85	3,45	3,90	8,40	3,45	3,45	1,90	1,85	1,85	1,90	N/C	9,40	6,40	6,45
Fack	5,65	5,15	4,85	1,85	5,15	3,15	7,35	5,15	5,15	1,85	1,85	1,85	1,85	N/C	11,15	4,65	5,65
Highspeed	6,40	5,90	5,45	1,90	5,90	3,90	8,90	5,90	5,90	2,40	1,90	1,90	2,40	N/C	10,45	6,45	6,90
HTCP	5,90	5,40	5,30	1,40	5,40	3,30	8,90	5,40	5,40	1,80	1,40	1,40	1,80	N/C	7,80	5,80	5,90
Hybla	9,85	13,10	6,60	4,25	13,10	5,85	14,65	27,00	13,10	3,60	13,25	4,25	3,60	N/C	70,01	13,15	14,15
Illinois	5,90	5,55	4,90	0,90	5,55	3,05	8,40	5,55	5,55	0,90	0,90	0,90	0,90	N/C	8,40	6,40	5,55
LowPriority	6,40	5,90	5,45	1,90	5,90	3,90	8,90	5,90	5,90	2,40	1,90	1,90	2,40	N/C	10,45	6,45	6,90
NewReno	6,35	6,90	5,35	15,55	6,90	3,40	9,10	6,90	6,90	16,05	16,05	16,05	16,05	N/C	7,90	5,35	6,35
Reno	5,65	5,15	4,85	4,85	5,15	3,15	7,35	5,15	5,15	1,85	1,85	1,85	1,85	N/C	11,15	4,65	5,65
Sack	5,65	5,15	4,85	1,85	5,15	3,15	7,35	5,15	5,15	1,85	1,85	1,85	1,85	N/C	11,15	4,65	5,65
Tahoe	6,35	6,90	5,35	15,55	6,90	3,40	9,10	6,90	6,90	16,05	16,05	16,05	16,05	N/C	7,90	5,35	6,35
Vegas	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C	0,85	N/C	N/C	N/C
Veno	5,40	4,40	4,95	1,15	4,40	2,95	4,90	4,40	4,40	1,15	1,15	1,15	1,15	N/C	10,95	5,45	4,95
Westwood	6,35	6,35	5,35	1,70	6,35	3,35	3,90	6,35	6,35	1,70	1,70	1,70	1,70	N/C	7,85	5,35	6,35
YeAH	5,80	10,90	11,50	2,90	10,90	5,40	12,90	10,90	10,90	3,40	2,90	2,90	3,40	N/C	14,40	4,90	9,80

Tabla 8: Tiempo de Convergencia (TC)

En la Tabla 8 se pueden apreciar los resultados del Tiempo de Convergencia. Cuando el segundo flujo comienza a transmitir, el primer flujo está utilizando la capacidad de la red sin ningún tipo de contienda, por lo que todos los recursos

de la red se encuentran a su disposición y solo depende de su eficiencia la utilización del ancho de banda. Cuando se introduce el segundo flujo, este comienza a demandar ancho de banda para la trasmisión de sus propios datos. En este punto, donde se produce un estado transitorio y dependiendo del grado de equidad, el primer flujo comienza a ceder ancho de banda mientras que el segundo comienza a ganarlo. De esta manera, se puede definir la métrica como el tiempo que transcurre desde que inicia la trasmisión del segundo flujo, hasta el punto en el que este flujo crece al 80% del valor de throughput del primer flujo, el cual viene cediéndole ancho de banda. En la siguiente figura (Figura 256) se observa lo descripto anteriormente.

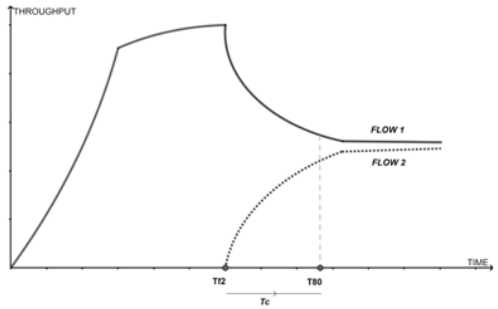

Figura 256: Medida del Tiempo de Convergencia (TC)

En la misma tabla (Tabla 8), se pueden observar casilleros tipificados como N/C. Esto se debe a que no existe un tiempo para el cual la diferencia entre los valores instantáneos de throughput de ambos flujos sea menor al 20% y se mantenga dentro de ese rango en un estado de equilibrio y justicia.

En la siguiente figura (Figura 257) se observan los valores del Índice de convergencia (IE) para las variantes ensayadas en estas simulaciones ordenadas de menor a mayor. En la figura se observa que TCP Vegas es la que obtiene los menores valores de Índice de Equidad contra las demás variantes.

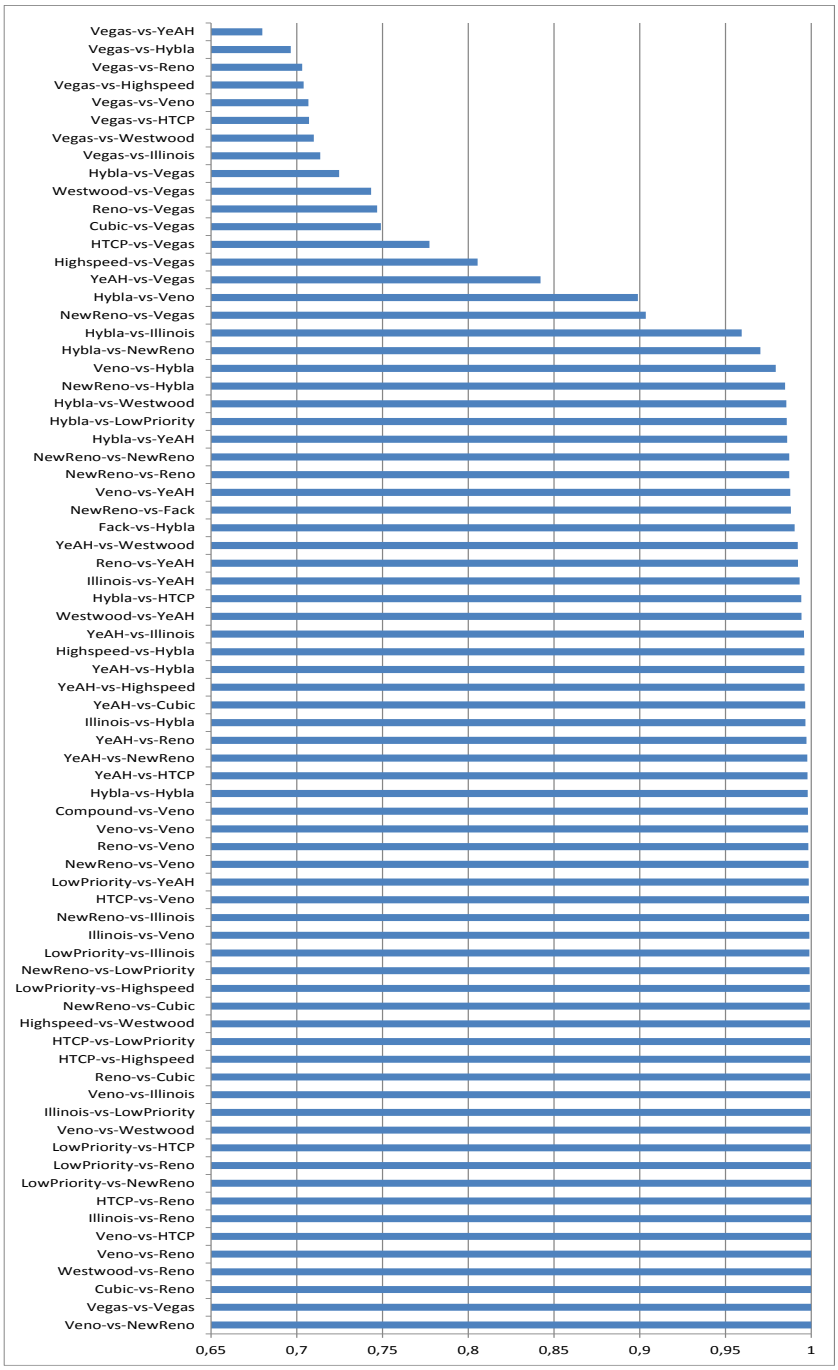

Figura 257: Índice de equidad ordenado

A continuación, se observa en las figuras que representan el valor del throughput (Figura 258, Figura 260 y Figura 262) y el tamaño de la ventana de congestión (Figura 259, Figura 261 y Figura 263) en función del tiempo, para las simulaciones que arrojaron los menores valores de IE.

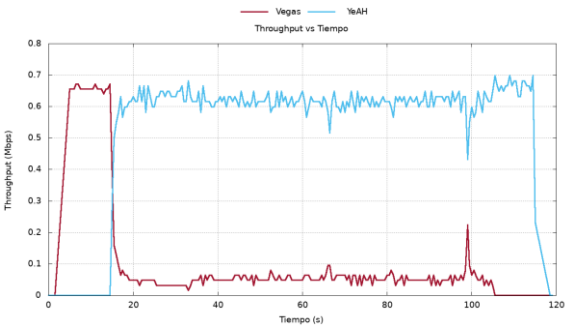

Figura 258: Throughput vs. Tiempo – Vegas vs. Yeah IE=0,6799

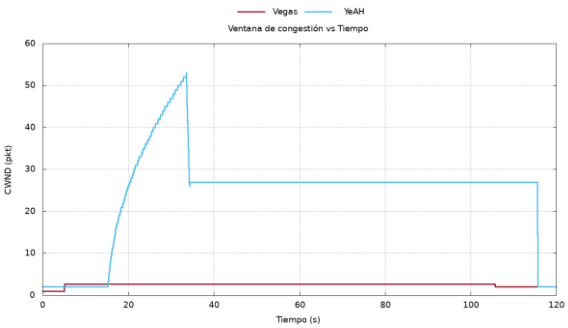

Figura 259: CWND vs. Tiempo – Vegas vs. YeAH

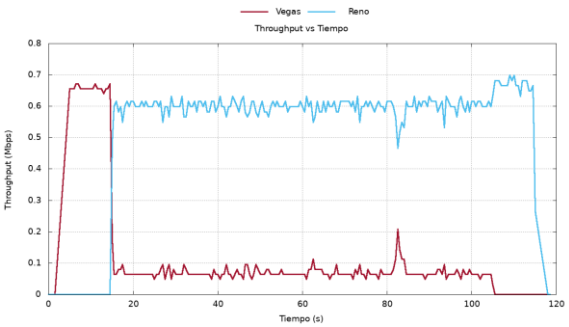

Figura 260: Throughput vs. Tiempo – Vegas vs. Reno IE=0,7031

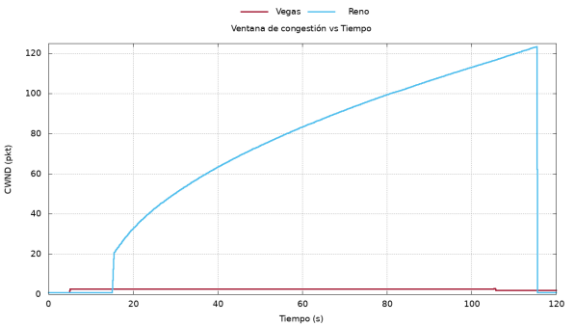

Figura 261: CWND vs. Tiempo – Vegas vs. Reno

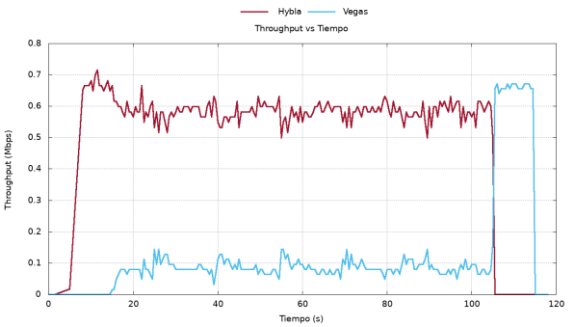

Figura 262: Throughput vs. Tiempo – Hybla vs. Vegas IE=0,7247

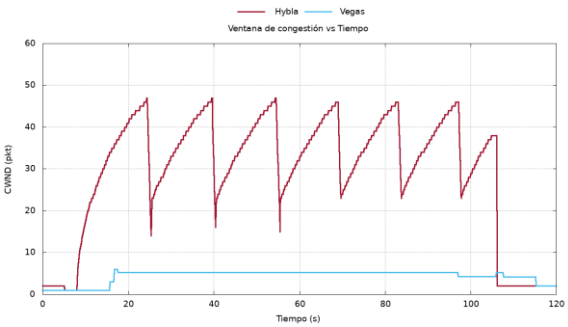

Figura 263: CWND vs. Tiempo – Hybla vs. Vegas

En las figuras anteriores de throughput en función del tiempo (Figura 258, Figura 260 y Figura 262) queda en evidencia que no existe un reparto justo del ancho de banda. Las figuras que muestran la evolución de la ventana de congestión (Figura 259, Figura 261 y Figura 263) permiten observar las estrategias de los diferentes algoritmos de control de congestión.

Sin embargo, cuando Vegas compite con otro flujo Vegas en este escenario, se obtiene uno de los índices de equidad más altos. En las siguientes figuras se observa el throughput (Figura 264) y el tamaño de la ventana de congestión (Figura 265) en función del tiempo para dos flujos TCP Vegas en contienda por los recursos de la red.

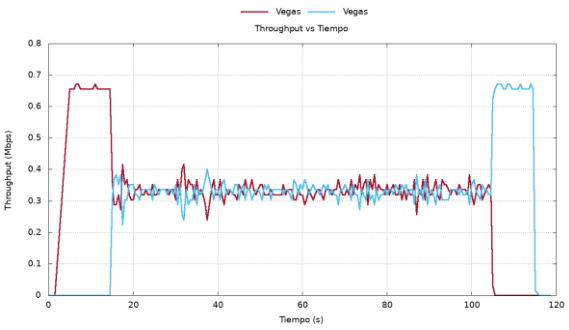

Figura 264: Throughput vs. Tiempo – Vegas vs. Vegas IE=0,9999 TC=0,85s

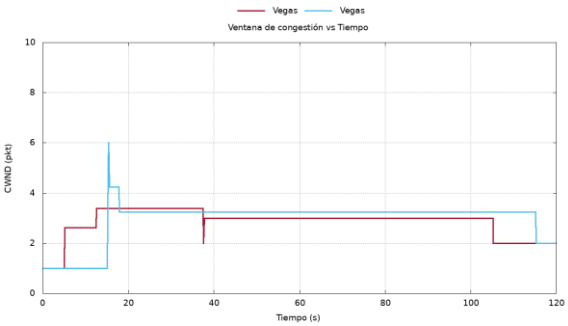

Figura 265: CWND vs. Tiempo – Vegas vs. Vegas

En la figura anterior de throughput (Figura 264) se observa que el reparto del ancho de banda es prácticamente en mitades, logrando un valor de Índice de Equidad muy cercano a 1 (IE=0,9999). Se puede observar también que el tiempo requerido para llegar a esa distribución es pequeño (TC=0,85 s). En la figura de la ventana de congestión (Figura 265) se observa como el flujo que comienza en primer término comienza a enviar datos a una tasa más alta y, cuando comienza a trasmitir el segundo flujo debe ajustar el tamaño de la ventana.

Otra de las contiendas que dio como resultado un valor del Índice de Equidad alto, en este escenario de prueba, es Veno vs. New Reno. A continuación, se muestran las gráficas del throughput (Figura 266) y ventana de congestión (Figura 267) para estos dos flujos. Se observa que la distribución del ancho de banda es en partes prácticamente iguales, aun cuando el tiempo que demora en llegar a ese equilibrio es levemente mayor al caso de dos flujos Vegas (TC=1,15 s).

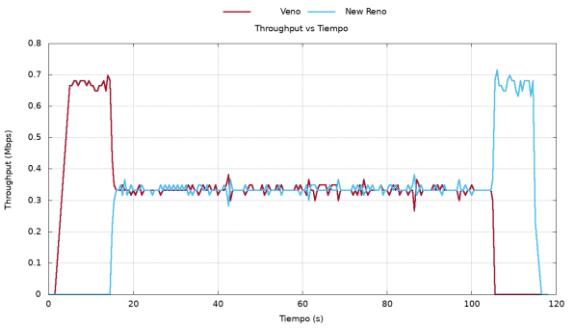

Figura 266: Throughput vs. Tiempo–Veno vs. NewReno IE =1 TC=1,15s

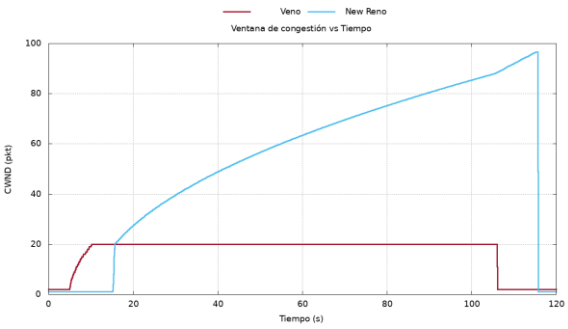

Figura 267: CWND vs. Tiempo – Veno vs. NewReno

Dentro de las contiendas de variantes de TCP que obtienen los valores medios de IE en la Figura 257, se puede observar Veno vs. Hybla y YeAH vs. Westwood entre otros. A continuación (Figura 268 a Figura 271) se muestran los gráficos del throughput y cwnd en función del tiempo para estos dos casos.

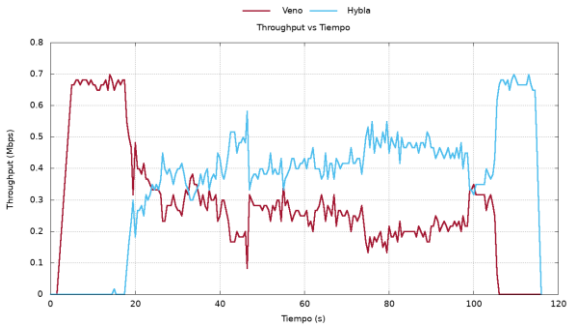

Figura 268: Throughput vs. Tiempo–Veno vs. Hybla IE=0,9793 TC=4,9

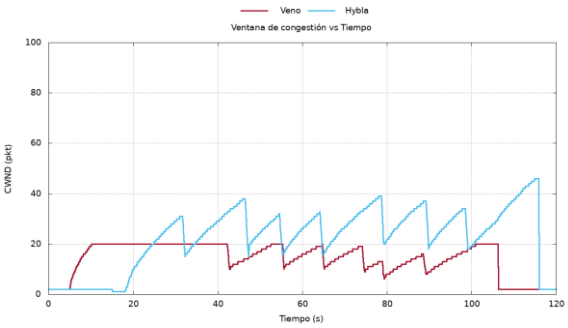

Figura 269: CWND vs. Tiempo – Veno vs Hybla

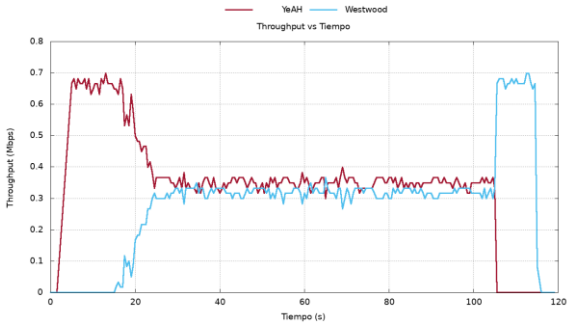

Figura 270: Throughput vs. Tiempo–Yeah vs. Westwood IE =0,9922 TC=4,9s

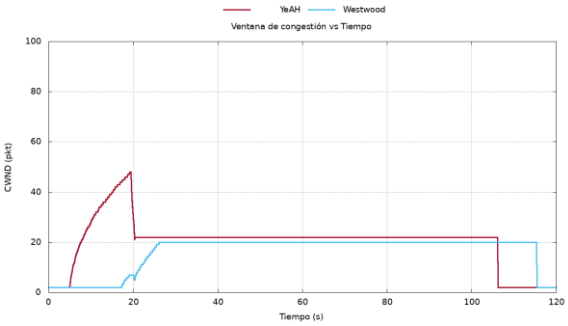

Figura 271: CWND vs. Tiempo – Yeah vs. Westwood

Las figuras anteriores de throughput (Figura 268 y Figura 270), comparadas con la de 2 flujos TCP Vegas (Figura 264) dejan en evidencia que el reparto de ancho de banda disponible es levemente diferente para cada una de los flujos. Este hecho queda cuantificado comparando los IE de ambas simulaciones, donde Vegas vs. Vegas obtiene IE=0.9999, mientras que Veno vs. Hybla obtiene IE=0,9793.

Como se observó, el Índice de Equidad (IE) (Ecuación 65) pondera qué tan ecuánime es la distribución de ese ancho de banda. Por otra parte, el Tiempo de Convergencia (TC), definido por la Ecuación 66, indica con cuánta celeridad llegan los dos protocolos al estado de equilibrio en la distribución del ancho de banda.

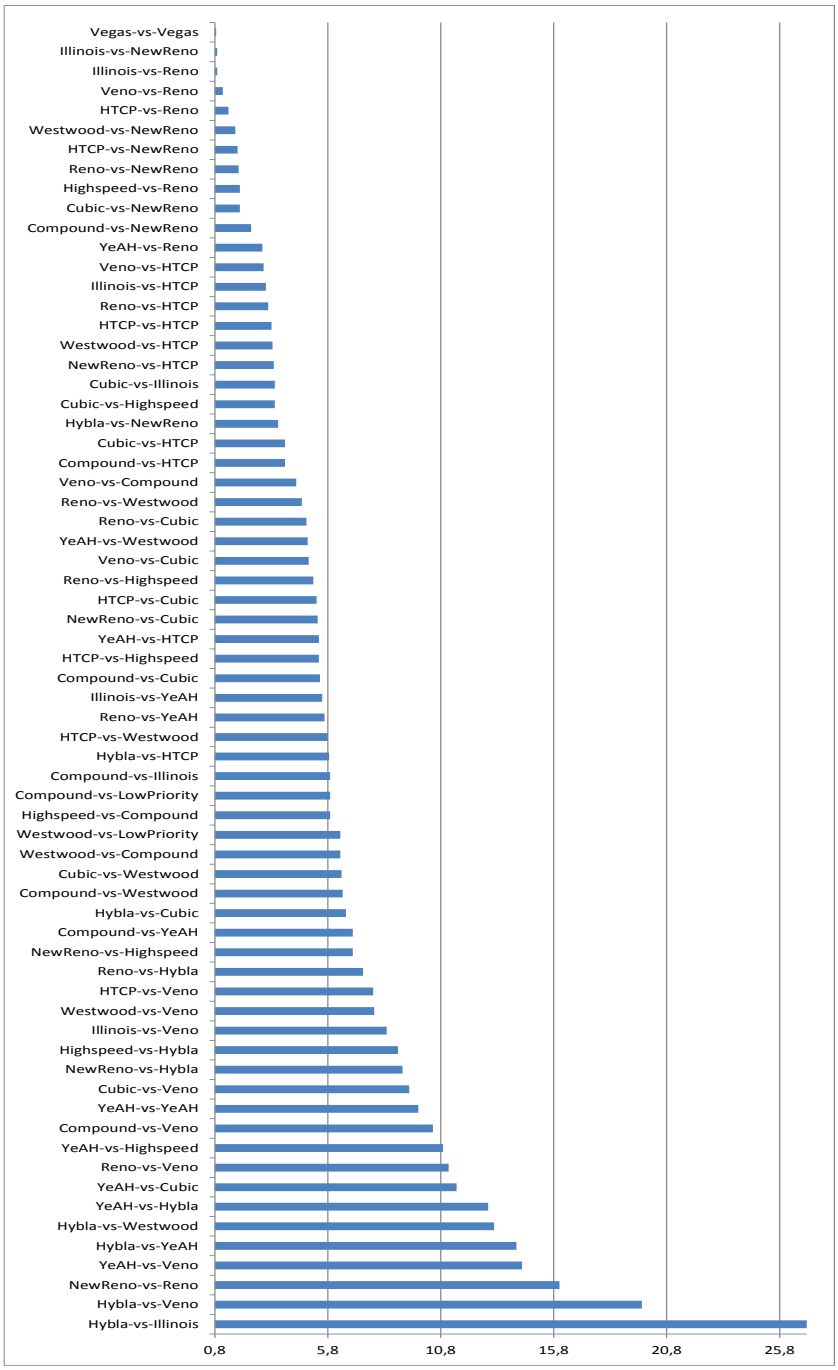

Figura 272: Tiempo de convergencia ordenado de menor a mayor

A partir de lo definido, en la Figura 272 se observan los valores de los Tiempo de Convergencia (TC) para las variantes ensayadas en estas simulaciones ordenadas de menor a mayor. En esta figura se observa que la simulación de 2 flujos en las que intervienen dos TCP Vegas tiene menor tiempo de convergencia, lo que se puede confirmar en la Figura 264.

A continuación, en las siguientes figuras (Figura 273 a Figura 276) se observa el throughput y cwnd en función del tiempo para dos ensayos, Veno vs. Reno y

HTCP vs. Reno que se encuentran entre los ensayos que poseen los menores valores de tiempo de convergencia.

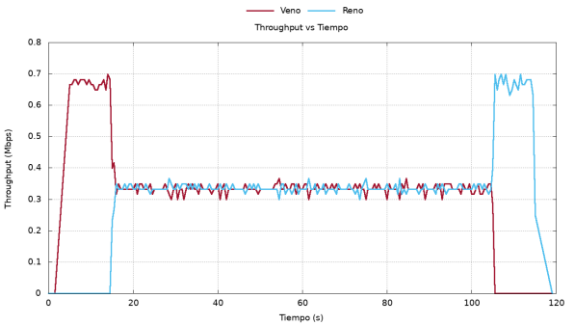

Figura 273: Throughput vs. Tiempo – Veno vs. Reno
IE =0,9999 TC=1,15s

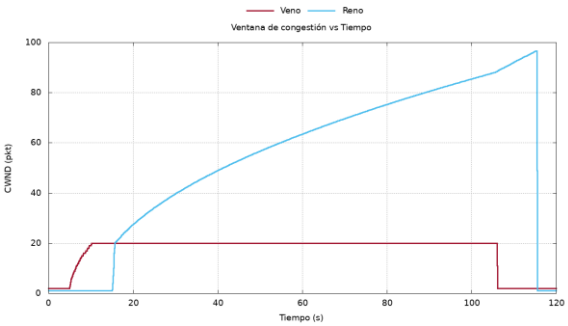

Figura 274: CWND vs. Tiempo – Veno vs. Reno

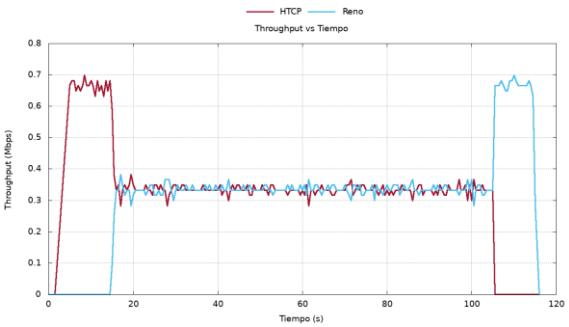

Figura 275: Throughput vs. Tiempo – HTCP vs. Reno
IE =0,9999 TC=1,4s

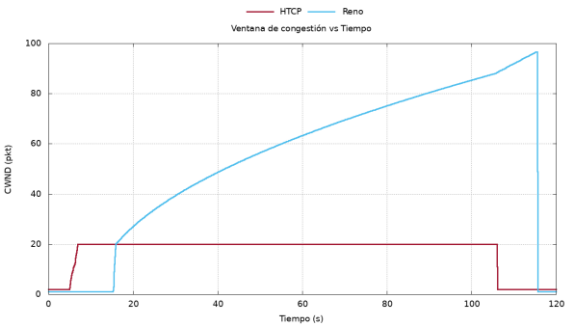

Figura 276: CWND vs. Tiempo – HTCP vs. Reno

En la Figura 273, se observa que el valor de TC es pequeño comparado a otros casos. Se observa que al momento de iniciar la transferencia Reno, se llega rápidamente a una ocupación equitativa del ancho de banda que se mantiene hasta el final, en donde Veno termina su trasmisión de datos. Es así, que, en ese momento, Reno ya si contienda por los recursos, toma todo el ancho de banda que su control de congestión le permite y transmite el volumen restante de datos. Además, el Índice de Equidad es prácticamente 1 (0.9999, de acuerdo a los decimales considerados), por ello la distribución del ancho de banda que se observa es fuertemente equitativa. El comportamiento es análogo al observado en la Figura 275.

A continuación, se observan los gráficos de las contiendas de flujos que obtuvieron los valores medios del Tiempo de Convergencia en la Figura 272. De esta manera, las siguientes figuras (Figura 277 a Figura 280) representan el throughput y la ventana de congestión en función del tiempo para Westwood vs. Compound y CUBIC vs. Veno

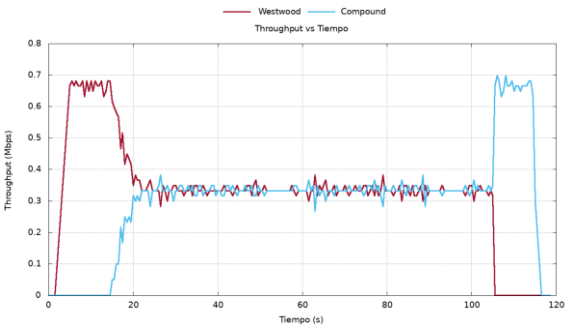

Figura 277: Throughput vs. Tiempo – Westwood vs. Compound

IE=0,9992 TC=6,35s

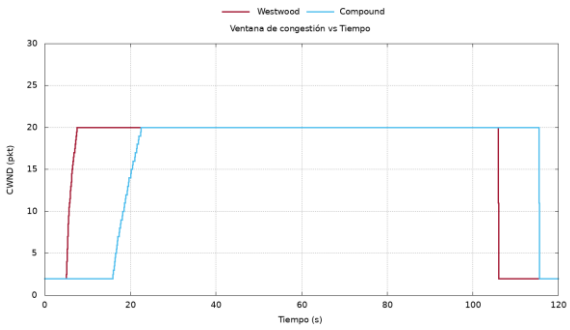

Figura 278: CWND vs. Tiempo – Westwood vs. Compound

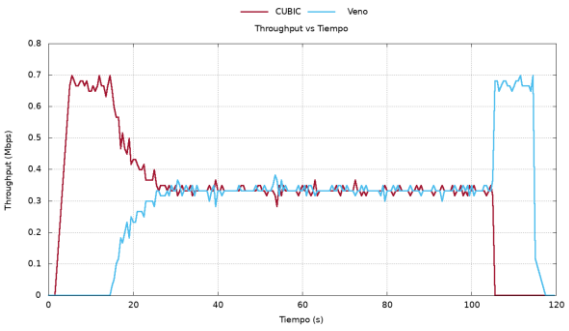

Figura 279: Throughput vs. Tiempo – CUBIC vs. Veno

IE =0.9986 TC=9,4s

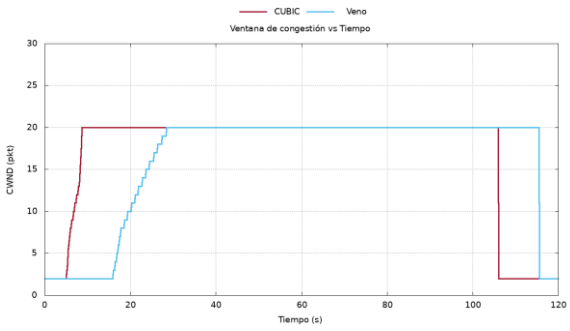

Figura 280: CWND vs. Tiempo – CUBIC vs. Veno

A continuación, se observan los gráficos de las métricas de las variantes de TCP que obtuvieron los valores más altos del tiempo de convergencia en la Figura 272. Las siguientes figuras (Figura 281 a Figura 284) representan el throughput y la ventana de congestión en función del tiempo para YeAH vs. Veno e Hybla vs. Illinois.

En la Figura 283, se observa el caso de la confrontación de los protocolos Hybla y Illinois y queda en evidencia que el IE indica que la distribución del ancho de banda es poco ecuánime. Hybla predomina a lo largo de gran parte del ensayo.

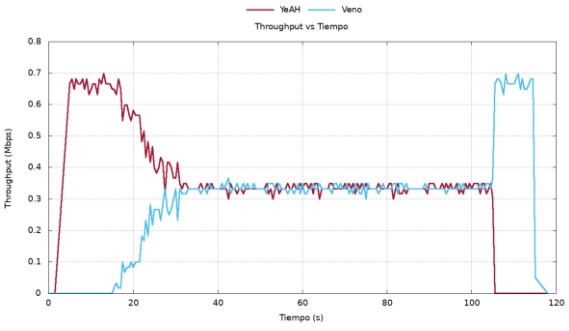

Figura 281: Throughput vs. Tiempo – Yeah vs. Veno

IE =0,9943 TC=14,4s

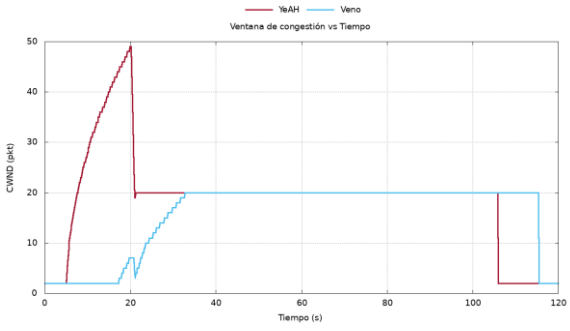

Figura 282: CWND vs. Tiempo – Yeah vs. Veno

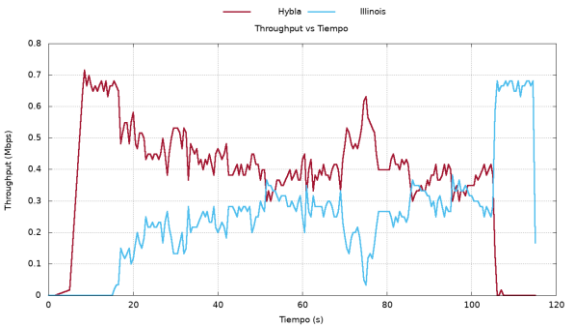

Figura 283: Throughput vs. Tiempo – Hybla vs. Illinois
IE=0,9595 TC=27s

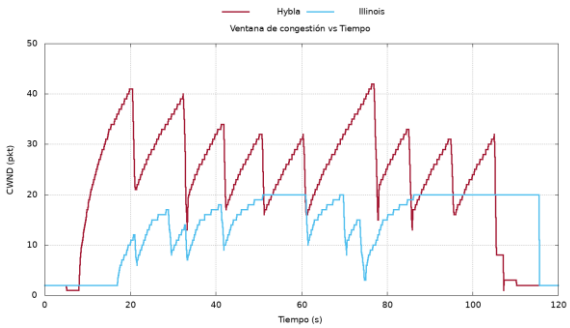

Figura 284: CWND vs. Tiempo – Hybla vs. Illinois

Otro aspecto importante a evaluar es la equidad que presentan las distintas variantes contra sí mismas, llamada *equidad intraprotocolo*. En el caso de TCP Vegas se analizó en la Figura 264, en donde obtenía un Tiempo de Convergencia muy bajo y un Índice de Equidad alto.

En las siguientes figuras (Figura 285 a Figura 290) se observa la contienda en el modelo analizado, de dos flujos TCP que implementan el mismo control de congestión. Para poder analizar el comportamiento, se presentan las gráficas del algunas de las variantes ensayadas.

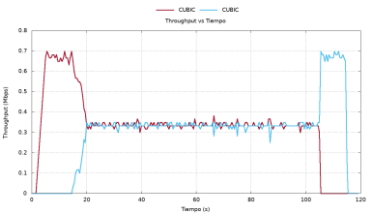

Figura 285: Throughput vs. Tiempo – CUBIC vs. CUBIC
IE=0,9989 TC=5.4

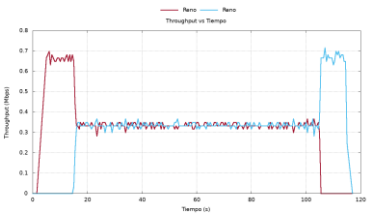

Figura 286: Throughput vs. Tiempo – Reno vs. Reno
IE=0,9999 TC=1.85

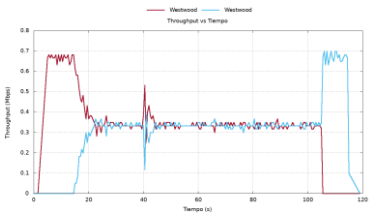

Figura 287: Throughput vs. Tiempo – Westwood vs. Westwood
IE=0,9990 TC=5.35

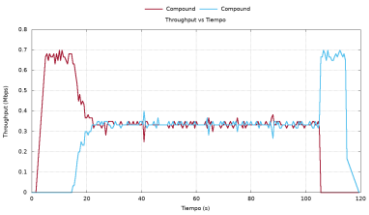

Figura 288: Throughput vs. Tiempo – Compound vs. Compound
IE=0,9992 TC=5,9

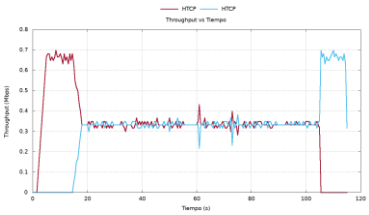

Figura 289: Throughput vs. Tiempo – HSTCP vs. HSTCP
IE=0,9992 TC=5,9

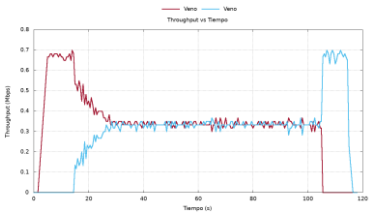

Figura 290: Throughput vs. Tiempo – Veno vs.Veno
IE=0,9982 TC=10,95

De la Tabla 7 se obtiene que el valor del Índice de Equidad en todos los casos observados en las figuras anteriores, fueron superiores a 0,998. De la Tabla 8 se obtiene que solo Reno vs. Reno (Figura 286) tiene un tiempo de

convergencia menor a 1 segundo. En el otro extremo (Figura 290), Veno vs. Veno tiene un TC mayor a los 10 segundos. En el resto de las pruebas el tiempo de convergencia estuvo en los alrededores de los 5,5 segundos.

A continuación, se presentan dos series de gráficos Throughput vs. Tiempo que permiten observar en forma cualitativa la dispersión de los resultados de las métricas analizadas, IE y TC.

En primer lugar, las siguientes figuras (Figura 291 a Figura 296) representan una serie de resultados de este grupo de simulaciones ordenados de acuerdo a los valores del Índice de Equidad, ordenaros en forma descendente.

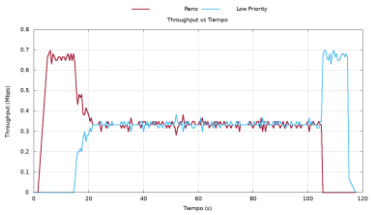

Figura 291: Throughput vs. Tiempo – Reno vs. LowPriority
IE =0,9996

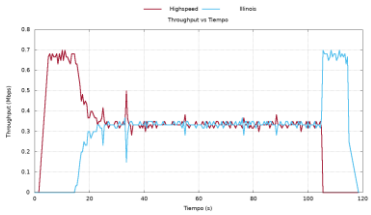

Figura 292: Throughput vs. Tiempo – Highspeed vs. Illinois
IE =0,9990

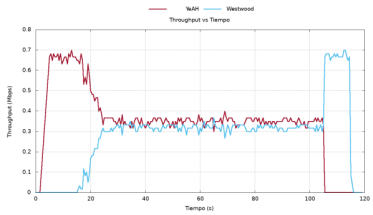

Figura 293: Throughput vs. Tiempo – YeAH vs. Westwood
IE =0,9922

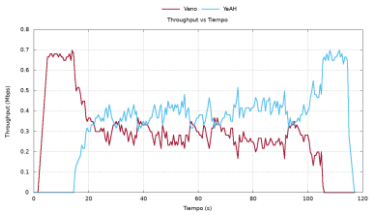

Figura 294: Throughput vs. Tiempo – Veno vs. YeAH
IE =0,9878

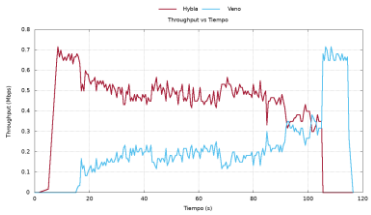

Figura 295: Throughput vs. Tiempo – Hybla vs. Veno
IE =0,8989

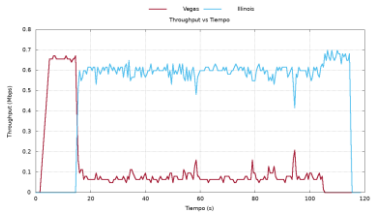

Figura 296: Throughput vs. Tiempo – Vegas vs. Illinois
IE =0,7136

En segundo lugar, en la siguiente serie de gráficos de throughput en función del tiempo (Figura 297 a Figura 302), se observa los resultados de la métrica Tiempo de Convergencia, cuyos valores de se ordenaron en forma creciente.

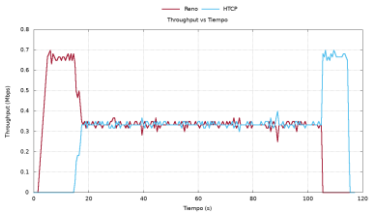

Figura 297: Throughput vs. Tiempo – Reno vs. HTCP
IE =0,9998 TC=3,15s

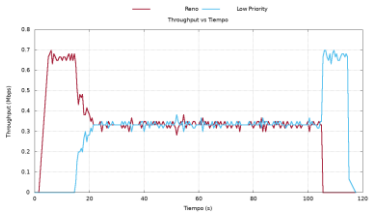

Figura 298: Throughput vs. Tiempo – Reno vs. LowPriority
IE =0,9996 TC=5,15

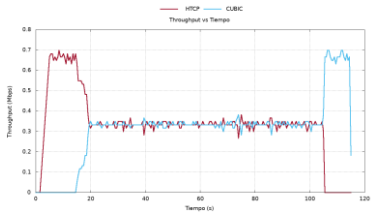

Figura 299: Throughput vs. Tiempo – HTCP vs. CUBIC
IE =0,9993 TC=5,3s

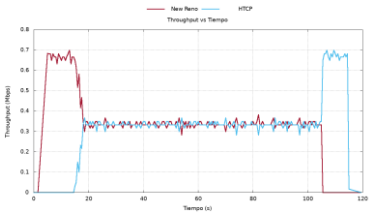

Figura 300: Throughput vs. Tiempo – NewReno vs. HSTCP IE =0,9991 TC=6,9s

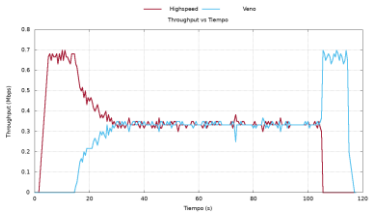

Figura 301: Throughput vs. Tiempo – Highspeed vs. Veno IE =0,9981 TC=10,45s

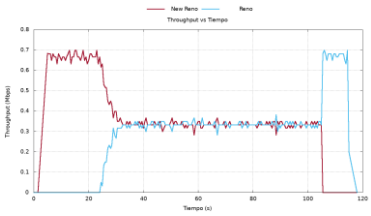

Figura 302: Throughput vs. Tiempo – NewReno vs. Reno IE =0,9872 TC=16,05s

Analizando las distintas pruebas de confrontación, se observa que el algoritmo de control de congestión de Vegas presenta un caso particular, no le permite sostener un uso de ancho de banda cuando compite con las restantes variantes de las utilizadas en estas simulaciones. En todos los casos, Vegas resigna ancho de banda y pasa a un segundo plano. En la Tabla 7, se observa que obtiene valores de IE bajos, lo que implica porciones de ancho de banda reducido con relación a cualquiera de las otras que comparten el mismo medio. Sin embargo, se observó que cuando comparte el medio con otro flujo con el mismo control de congestión, alcanza un alto valor de IE en un tiempo de convergencia reducido.

Por lo tanto, en este caso particular, se puede decir que Vegas no presenta equidad InterProtocolo, pero si presenta equidad IntraProtocolo.

En el otro extremo, variantes como Hybla, Illinois y aquellos con controles de congestión diseñados para redes de alta velocidad, tienden a ser muy agresivos y a dominar a sus contrapartes en los ensayos e intentan capturar el mayor ancho de banda posible.

En el medio de estos dos extremos, se encuentran la mayoría de las variantes de TCP ensayadas en estas simulaciones, donde los algoritmos de control de congestión distribuyen de una manera equitativa el ancho de banda disponible. Este comportamiento se pudo observar en la gran mayoría de las confrontaciones. Si se comparan la Figura 277 y la Figura 279, la diferencia entre ellas se evidencia en el tiempo necesario para llegar a ese reparto equitativo. Es tiempo desde que se inicia la confrontación hasta que los dos protocolos llegan a un estado de equilibrio en la negociación del ancho de banda.

A partir de lo analizado, se puede apreciar que las dos métricas utilizadas proporcionan suficiente información como para realizar un análisis del comportamiento de la confrontación entre mecanismos de control de congestión. Además, resulta evidente que este formato de representación es extensible al análisis de más de dos flujos que deben competir por los recursos, siendo esta una línea de futuras investigaciones.

4.6.7. Contienda de hasta 8 flujos con errores en ráfaga

En virtud de lo analizado en el grupo de *4.6.5 Contienda de mecanismos de control de congestión en un modelo heterogéneo (2 flujos)* y en *4.6.6 Equidad: Contienda de mecanismos de control de congestión*, se propone un nuevo ensayo modificando y complejizando el modelo para las simulaciones, agregando sucesivamente un nodo cableado y un nodo inalámbrico que permita adicionar un flujo de la misma variante. Los nuevos nodos y enlaces poseen las mismas características que los nodos y enlaces anteriores, y el nuevo flujo TCP es un FTP que no presenta errores y trasmite la cantidad de paquetes necesarios para estar trasmitiendo hasta el final de la simulación [150].

Este procedimiento se repitió sucesivamente en distintas simulaciones para cada uno de las variantes de TCP ensayadas y para cada una de las longitudes de errores en ráfagas estipulados, agregando de a dos nodos y un flujo TCP hasta llegar a las 8 trasferencias FTP simultaneas, en donde solo la primera presenta los errores.

La siguiente figura (Figura 303) representa el modelo utilizado para esta tanda de ensayos. Los nodos 0 y 2 están vinculados por un enlace cableado que se configuro como dúplex, con un ancho de banda 2 Mb/s, retardo de propagación 2 ms. y política de servicio de las colas DropTail. Análogamente los enlaces entre 1 hasta j y 2 tienen la misma configuración. El enlace entre los nodos 2 y 3 es inalámbrico y se configuro como modo de propagación TwoRayGround, la capa física WirelessPhy, MAC 802.11, la antena OmniAntenna y el nodo inalámbrico sin movilidad. La misma configuración tienen los enlaces entre la estación base 2 y los nodos desde 3 hasta k.

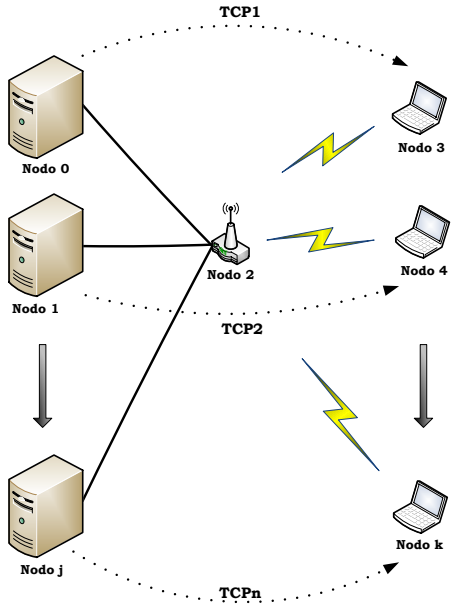

Figura 303: modelo para la simulación de la contienda de n flujos con errores en ráfagas

Se realizaron simulaciones independientes para cada una de las implementaciones de las distintas variantes de TCP. Los agentes TCP que se utilizaron fueron Reno, CUBIC, Vegas, y Westwood, tal como están implementados en NS-2 (ver. 2.35), sin modificación alguna. En el caso de TCP Vegas, para α y β se utilizan los valores por defecto, es decir, α=1 y β=3.

Para cada una de ellas, se realizaron simulaciones con 1, 2, 4 y 8 flujos simultáneos, todos con el mismo control de congestión, donde se le introdujo solo al primer flujo errores en ráfaga de 0, 5, 10, 15 y 20 paquetes de longitud.

Resultados

Con este ensayo, basado en una batería de simulaciones, se pretende determinar el comportamiento del throughput de los cuatro protocolos propuestos ante la presencia de errores en ráfaga de distintas longitudes y la presencia de otros flujos TCP de la misma variante.

A continuación, se presentan los gráficos obtenidos de las métricas throughput y número de secuencia en función del tiempo, superponiendo los resultados para el caso de 1, 2, 4 y 8 tráficos simultáneos de una misma variante del protocolo TCP.

Los gráficos se han agrupado teniendo en cuenta la variante de TCP implementada, para poder estudiar la forma en que afecta el rendimiento ante una ráfaga de errores de longitud creciente que solo afecta al primer flujo.

En las siguientes figuras (Figura 304 a Figura 313) se observa el throughput y la evolución del número de secuencia en función del tiempo de *TCP Reno*. En cada figura se observan superpuestos los valores de la métrica del primer flujo para los casos de 1, 2, 4 y 8 flujos TCP simultáneos.

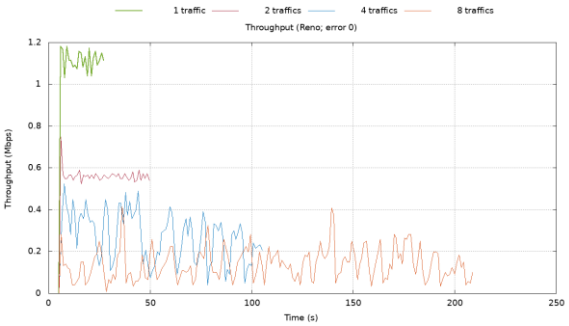

Figura 304: Throughput vs. Tiempo – TCP Reno
0 paquetes perdidos

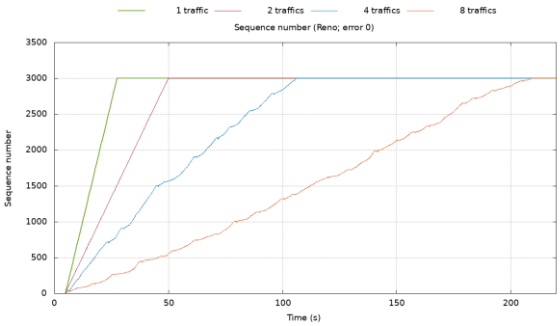

Figura 305: N° de Secuencia vs. Tiempo – TCP Reno
0 paquetes perdidos

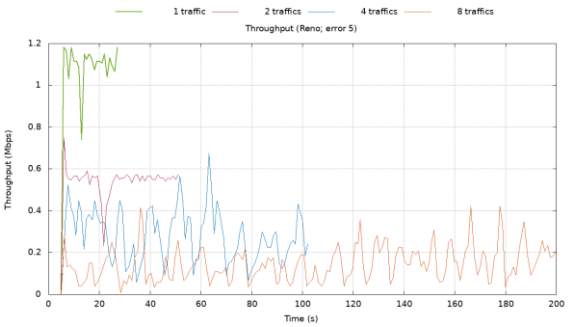

Figura 306: Throughput vs. Tiempo - TCP Reno
5 paquetes perdidos

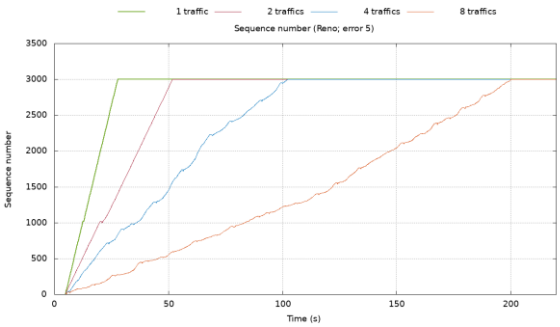

Figura 307: N° de Secuencia vs. Tiempo – TCP Reno
5 paquetes perdidos

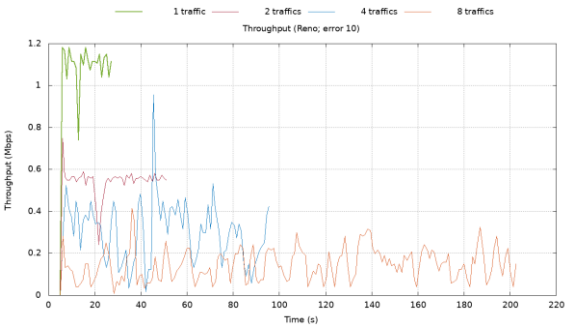

Figura 308: Throughput vs. Tiempo - TCP Reno
10 paquetes perdidos

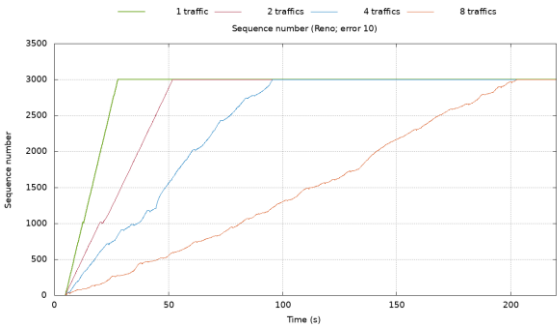

Figura 309: N° de Secuencia vs. Tiempo – TCP Reno
10 paquetes perdidos

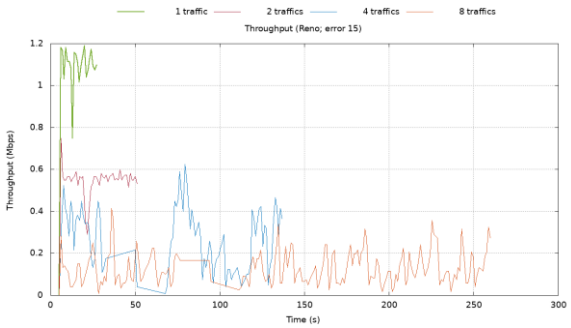

Figura 310: Throughput vs. Tiempo - TCP Reno 15 paquetes perdidos

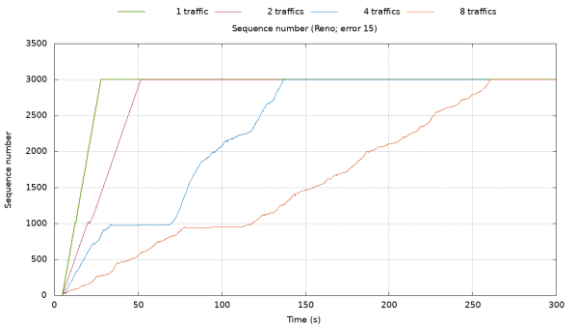

Figura 311: N° de Secuencia vs. Tiempo – TCP Reno 15 paquetes perdidos

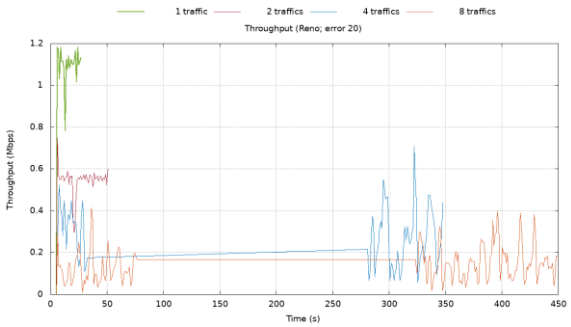

Figura 312: Throughput vs. Tiempo - TCP Reno 20 paquetes perdidos

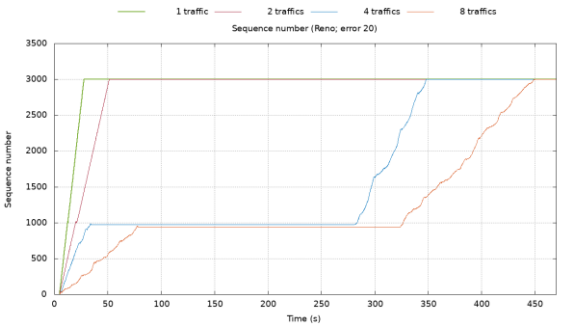

Figura 313: N° de Secuencia vs. Tiempo – TCP Reno 20 paquetes perdidos

En la Figura 304 se observa como el throughput evoluciona en función del tiempo de las simulaciones para el caso sin paquetes perdidos. La Figura 305 muestra cómo avanza el número de secuencia en esta condición. Se puede observar que los valores instantáneos de throughput son menores a medida que aumenta la cantidad de flujos lo que se condice con la pendiente de la curva de avance de los números de secuencia. Está pendiente se hace cada vez menos pronunciada a medida que aumentan la cantidad de flujos simultáneos.

En el caso de la ráfaga de 10 paquetes el efecto de reducción del throughput es del mismo orden de magnitud relativa para las distintas cantidades de flujos en contienda (Figura 308). Se verifica en las curvas de número de secuencia (Figura 309).

En la Figura 310 y la Figura 311 se observan los valores de las métricas en función del tiempo para una ráfaga de errores de 15 paquetes de longitud y se observa que se producen caídas en su valor instantáneo mientras se pierden paquetes. Si bien este hecho se percibe en menor medida desde longitudes de ráfagas más cortas, cuando se pierden 15 paquetes consecutivos, en este

escenario, se puede observar que, en los ensayos con 4 y 8 flujos, TCP demora más tiempo en volver a la tasa anterior. Esto queda en evidencia en la forma en que avanza el número de secuencia en la Figura 311.

En el caso de 20 paquetes pedidos (Figura 312 y Figura 313) se hace mucho más evidente la diferencia en el tiempo que requiere Reno para volver a la tasa de envío anterior para 4 y 8 flujos simultáneos. En estos casos, el resto de los flujos utilizaron el ancho de banda que dejo disponible el que sufrió los errores en ráfaga y debe volver a competir para recuperarlos. Esta situación, se hace más evidente a medida que más cantidad de flujos compiten por el recurso.

En la siguiente serie de gráficos (Figura 313 a Figura 323), se observa la evolución del throughput y del número de secuencia en función del tiempo de la simulación para la variante *TCP CUBIC*. En forma análoga al caso anterior, se superponen las métricas del primer flujo para 1, 2, 4 y 8 flujos simultáneos.

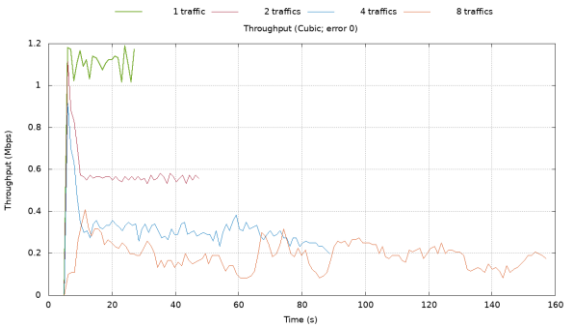

Figura 314: Throughput vs. Tiempo - TCP CUBIC
0 paquetes perdidos

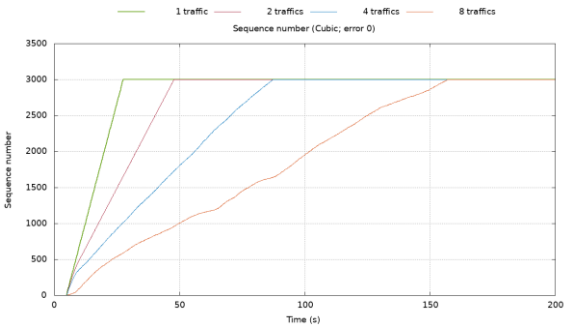

Figura 315: N° de Secuencia vs. Tiempo – TCP CUBIC
0 paquetes perdidos

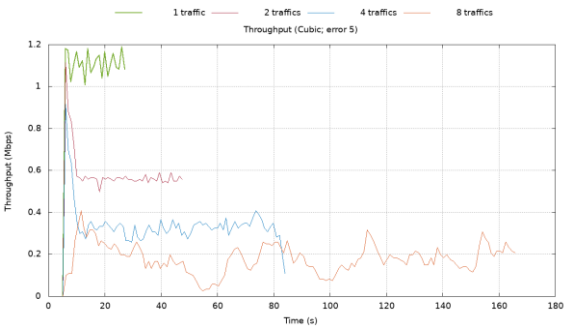

Figura 316: Throughput vs. Tiempo - TCP CUBIC
5 paquetes perdidos

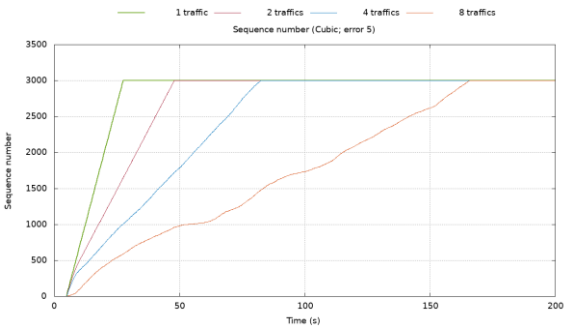

Figura 317: N° de Secuencia vs. Tiempo – TCP CUBIC
5 paquetes perdidos

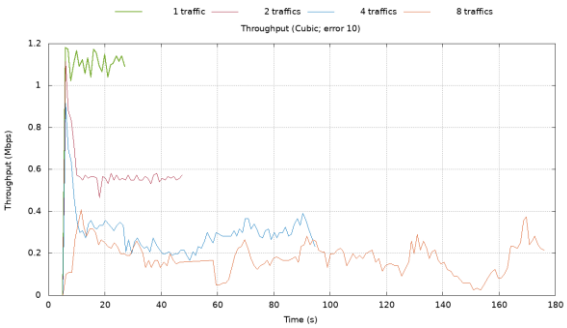

Figura 318: Throughput vs. Tiempo - TCP CUBIC
10 paquetes perdidos

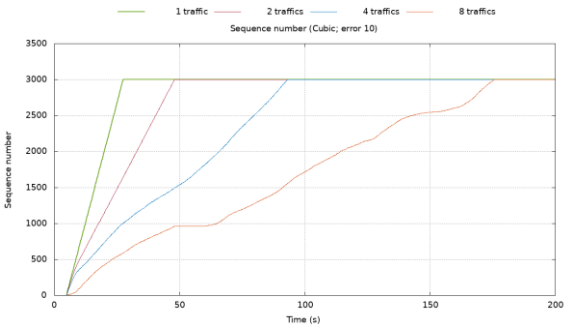

Figura 319: N° de Secuencia vs. Tiempo – TCP CUBIC
10 paquetes perdidos

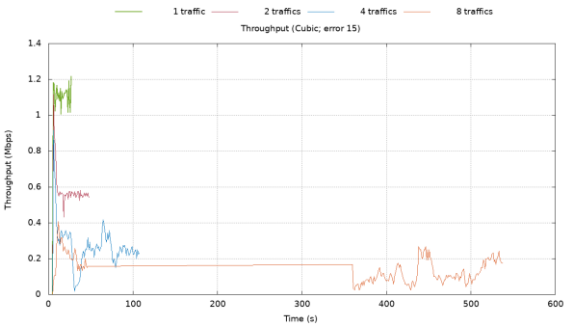

Figura 320: Throughput vs. Tiempo - TCP CUBIC
15 paquetes perdidos

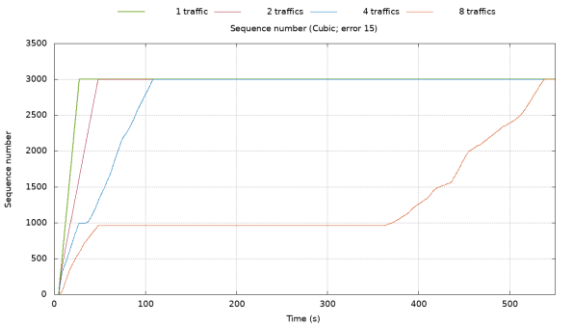

Figura 321: N° de Secuencia vs. Tiempo – TCP CUBIC
15 paquetes perdidos

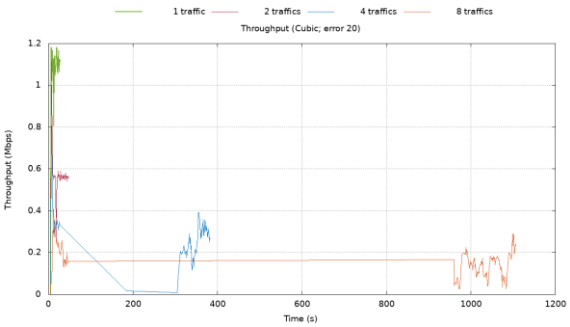

Figura 322: Throughput vs. Tiempo - TCP CUBIC
20 paquetes perdidos

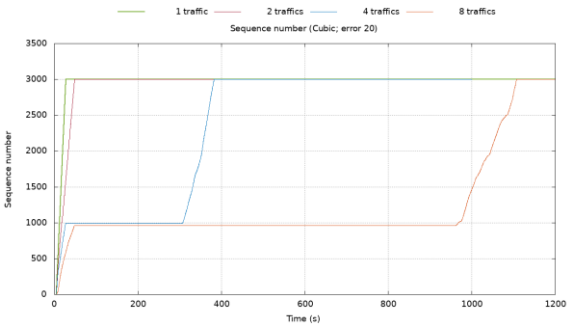

Figura 323: N° de Secuencia vs. Tiempo – TCP CUBIC
20 paquetes perdidos

De las figuras anteriores se desprende que TCP CUBIC es un poco más susceptible a los errores en ráfaga. En las figuras correspondiente a los ensayos con 10 paquetes perdidos (Figura 318 y Figura 319), CUBIC es levemente más sensible que Reno, excepto en la de la contienda de 8 flujos.

En el caso de errores de 15 paquetes de longitud (Figura 320 y Figura 321) ya se aprecia una sensibilidad mayor de CUBIC comparado con Reno. En el ensayo de 4 flujos, se observa una caída en el throughput con una recuperación rápida que se puede verificar en el breve corte que tiene la gráfica de Número de Secuencia (Figura 321). Sin embargo, en la simulación

con 8 flujos simultáneos, el tiempo que requiere para recuperar la tasa de envío, es sensiblemente más alto.

En la ráfaga de 20 paquetes (Figura 322 y Figura 323) se observa que las simulaciones de 4 y 8 flujos demoran en reiniciar la trasmisión y recuperar la tasa de envío. En el caso de los 8 flujos el tiempo de recuperación es alto.

En forma análoga, se presenta a continuación la serie de figuras (Figura 324 a Figura 333) donde se observan las métricas del primer flujo *TCP Vegas* para 1, 2, 4 y 8 flujos simultáneos. Para estas simulaciones, se utilizaron los parámetros por defecto de TCP Vegas.

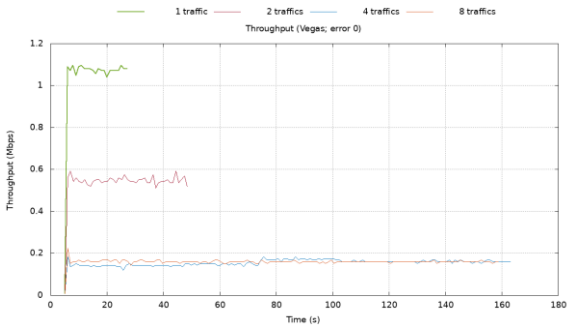

Figura 324: Throughput vs. Tiempo - TCP Vegas
0 paquetes perdidos

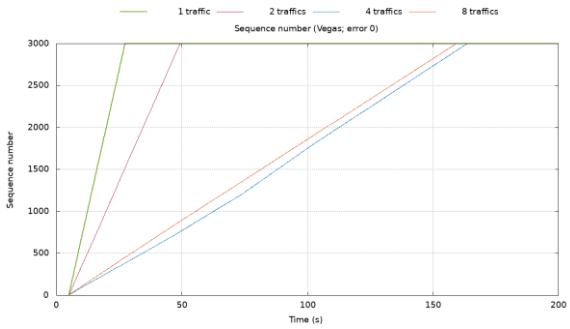

Figura 325: N° de Secuencia vs. Tiempo – TCP Vegas
0 paquetes perdidos

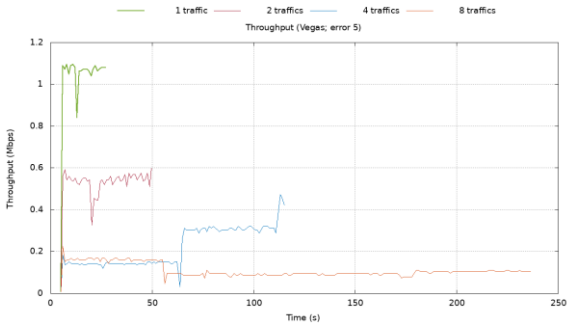

Figura 326: Throughput vs. Tiempo - TCP Vegas
5 paquetes perdidos

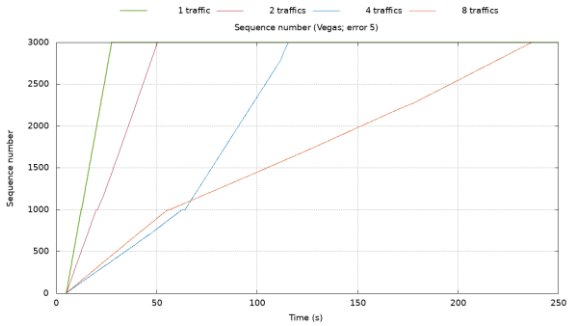

Figura 327: N° de Secuencia vs. Tiempo – TCP Vegas
5 paquetes perdidos

Figura 328: Throughput vs. Tiempo - TCP Vegas
10 paquetes perdidos

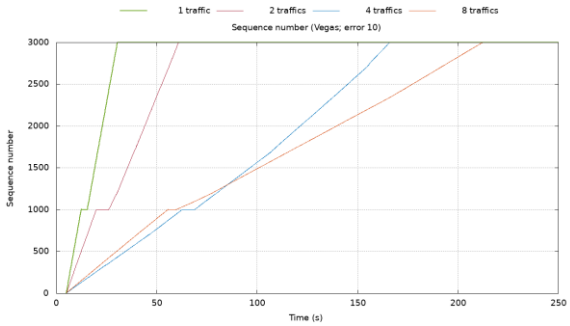

Figura 329: N° de Secuencia vs. Tiempo – TCP Vegas
10 paquetes perdidos

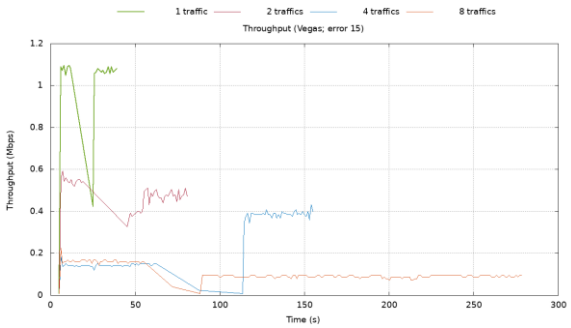

Figura 330: Throughput vs. Tiempo - TCP Vegas 15 paquetes perdidos

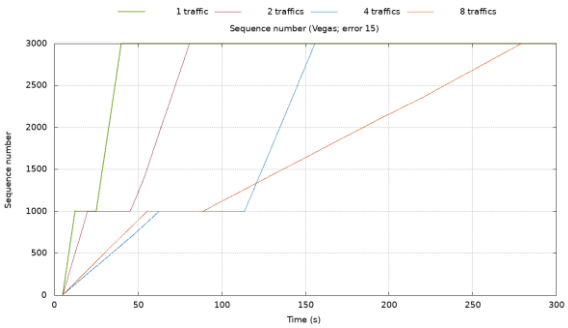

Figura 331: N° de Secuencia vs. Tiempo – TCP Vegas 15 paquetes perdidos

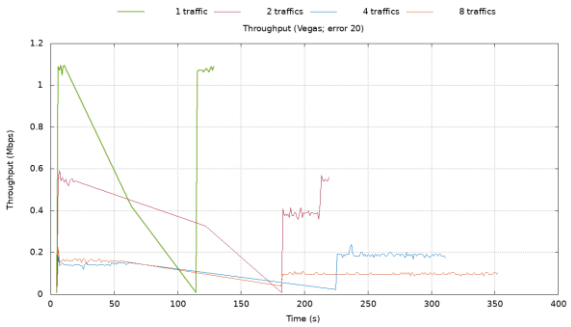

Figura 332: Throughput vs. Tiempo - TCP Vegas 20 paquetes perdidos

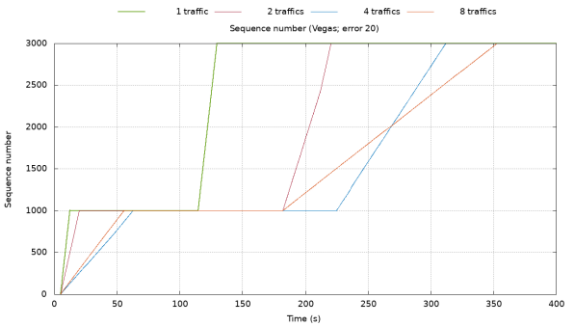

Figura 333: N° de Secuencia vs. Tiempo – TCP Vegas 20 paquetes perdidos

De las figuras anteriores se observa que desde el primer ensayo con 5 paquetes perdidos TCP Vegas es muy susceptible a este tipo de errores. Se verifica en la Figura 326 con las leves caídas en el valor del throughput y en la Figura 327 con el quiebre de las gráficas de número se secuencia. En particular, en el ensayo de 4 flujos, después de ocurrida la ráfaga de errores, el flujo Vegas retoma la transmisión con una tasa de envío mayor a la que traía antes de los errores. Caso contrario es el de 8 flujos simultáneos, donde se recupera con una tasa de envío menor. Esto explica el cruce de las gráficas que se observa la Figura 327.

En las simulaciones con 10 paquetes perdidos (Figura 328 y Figura 329) se observa que en todos los casos el throughput desciende casi hasta 0 durante el tiempo de duración de los errores en ráfaga. En la mayoría de los casos la recuperación es relativamente rápida, excepto en los casos de 2 y 8 flujos simultáneos, donde se realiza a una tasa de envío menor. Se puede comprobar con los cruces y las pendientes de las gráficas de la Figura 329.

En el caso de 15 paquetes perdidos (Figura 330 y Figura 331) se observa que en todos los casos el throughput tiene una caída significativa y un tiempo de

recuperación más alto. Se puede observar también que el ensayo de 4 flujos es el que más demora en retomar el envío de datos y lo hace a una tasa levemente mayor de la que lo venía haciendo antes de los errores. Sin embargo, en el caso de 2 y 8 flujos, la recuperación se logra a una tasa de envío apenas menor. Todo esto queda en evidencia en la Figura 331, donde las gráficas del número de secuencia se cruzan y cambian su pendiente.

Para 20 paquetes (Figura 332 y Figura 333), todos los ensayos sufren caídas significativas y una lenta recuperación. Solo en el ensayo de 1 flujo se mantiene tasa de envío después de su recuperación.

En las siguientes figuras (Figura 334 a Figura 343) se observan el throughput y la evolución del número de secuencia TCP en función del tiempo para la serie de ensayos con la variante *TCP Westwood*, incrementando los flujos simultáneos desde 1 hasta 8. En todos los casos, las métricas corresponden al primer flujo.

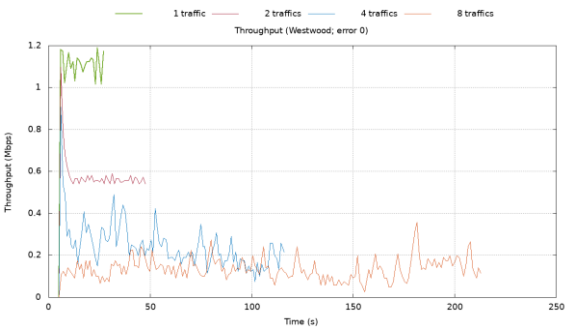

Figura 334: Throughput vs. Tiempo - TCP Westwood
0 paquetes perdidos

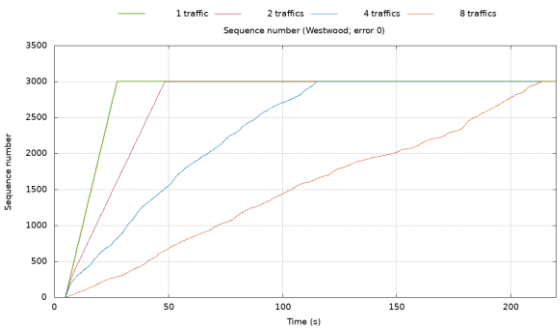

Figura 335: N° de Sec. vs. Tiempo – TCP Westwood
0 paquetes perdidos

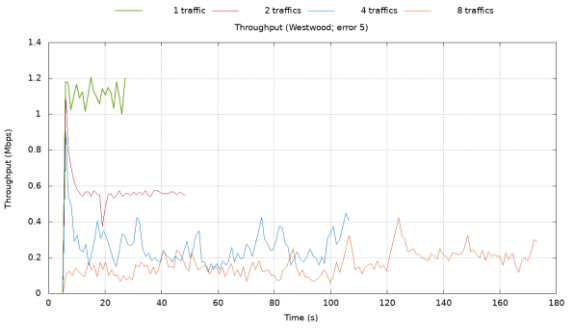

Figura 336: Throughput vs. Tiempo - TCP Westwood
5 paquetes perdidos

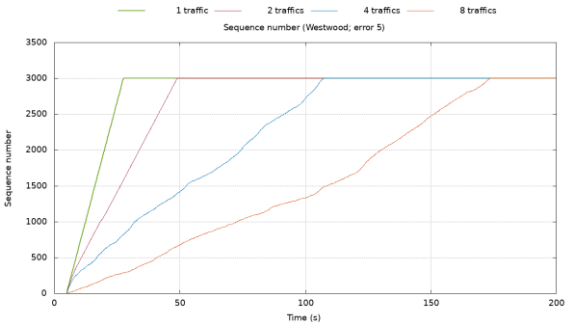

Figura 337: N° de Sec. vs. Tiempo – TCP Westwood
5 paquetes perdidos

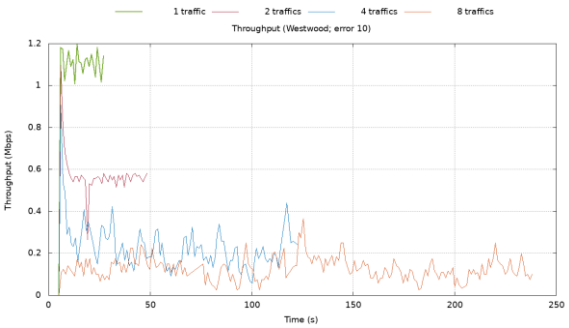

Figura 338: Throughput vs. Tiempo - TCP Westwood 10 paquetes perdidos

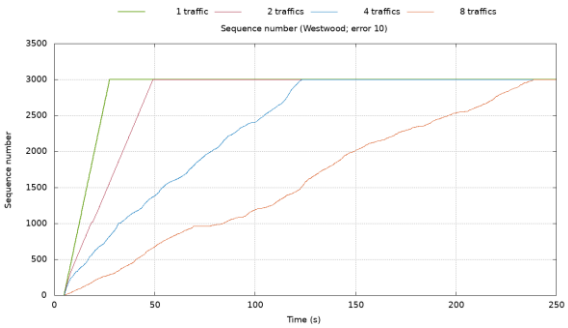

Figura 339: N° de Sec. vs. Tiempo – TCP Westwood 10 paquetes perdidos

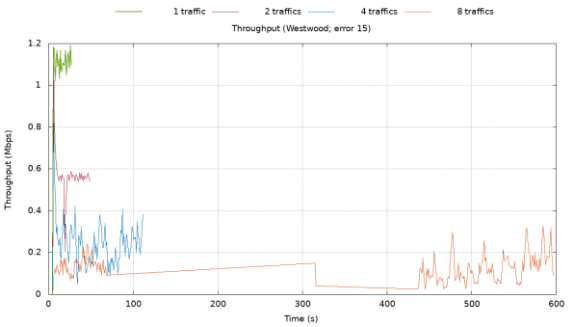

Figura 340: Throughput vs. Tiempo - TCP Westwood 15 paquetes perdidos

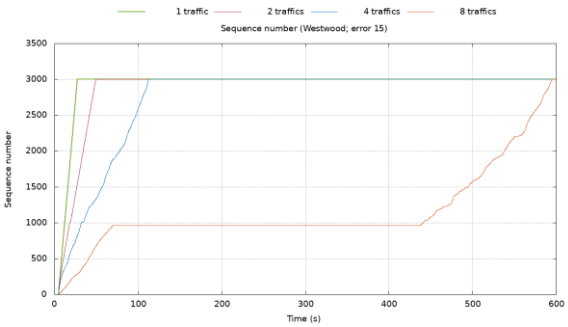

Figura 341: N° de Sec. vs. Tiempo – TCP Westwood 15 paquetes perdidos

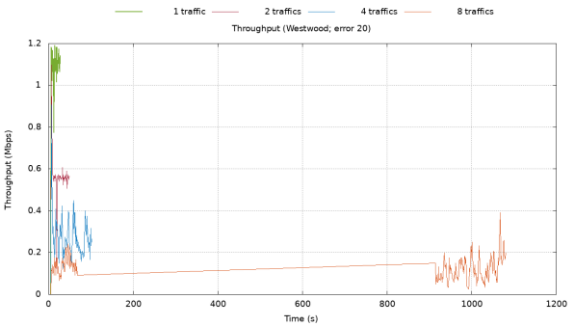

Figura 342: Throughput vs. Tiempo - TCP Westwood 20 paquetes perdidos

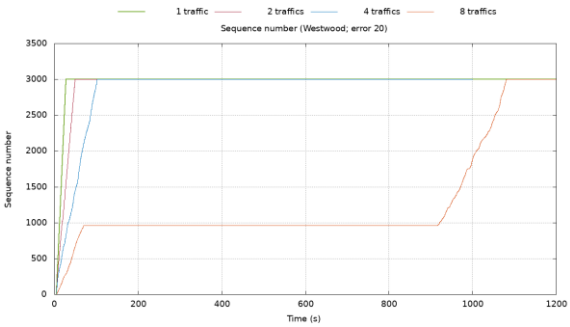

Figura 343: N° de Sec. vs. Tiempo – TCP Westwood 20 paquetes perdidos

La contienda sin errores (Figura 334 y Figura 335) muestra que para el caso de 1 y 2 flujos la tasa de envío es prácticamente constante, mientras que para 4 flujos y, en mayor medida, para 8 flujos se ve que el control de congestión ajusta suavemente la tasa de envío. Eso se puede observar en las variaciones de las pendientes de las curvas de la Figura 335.

En la tanda de ensayos para una longitud de ráfaga de 5 paquetes (Figura 336 y Figura 337), se observa un comportamiento similar, a excepción del caso de 2 flujos, donde se observa que el throughput sufre una muy leve caída.

En el caso de 10 paquetes perdidos (Figura 338 y Figura 339) se observa en la gráfica de número de secuencia que en el ensayo de 8 flujos es el más

susceptible, lo que queda de manifiesto por la forma de curva y por el tiempo necesario para completar la transmisión. El efecto sobre los ensayos con 1, 2 y 4 flujos simultáneos es pequeño.

Cuando se pierden 15 paquetes consecutivos (Figura 340 y Figura 341) el ensayo de 8 flujos simultáneos muestra que requiere de mayor tiempo para poder recuperar la tasa de envío. También se observa un comportamiento similar en los ensayos con la longitud de la ráfaga de 20 paquetes (Figura 342 y Figura 343).

Como se observa en los resultados, a medida que se incrementa el tamaño de la ráfaga y la cantidad de flujos simultáneos, el tiempo de recuperación de la tasa de transmisión de datos se incrementa. Este efecto se verifica tanto en los gráficos de throughput, como así también en los de número de secuencia. En particular, se puede observar sobre las pruebas realizadas, que este efecto se vuelve más notable a partir del tamaño de ráfaga de 15 paquetes de longitud y, es aún más acentuado cuando las pruebas incrementan la cantidad de flujos en un trayecto común, lo que implica una mayor competencia por el ancho de banda.

De los protocolos analizados, es nuevamente TCP Vegas quién muestra características particulares. En la secuencia de pruebas, se mantuvieron los parámetros por defecto y los resultados se observan desde la Figura 324 a la Figura 333. Estos resultados son coherentes con lo observado en *4.6.2 Errores en ráfaga de distinta longitud en un escenario simple*, donde TCP Vegas resultaba la variante más susceptible a este tipo de errores. A partir de lo observado en *4.6.3 TCP Vegas: Errores en ráfaga de distinta longitud*, resulta interesante realizar, sobre este mismo escenario, algunos ensayos con distintos valores de α y β para estudiar su comportamiento.

Las siguientes figuras (Figura 344 y Figura 345) representan el throughput y el número de secuencia en función del tiempo de TCP Vegas con α=1 y β=5 para una ráfaga de 15 paquetes perdidos. Análogamente solo se grafican las métricas del primero flujo.

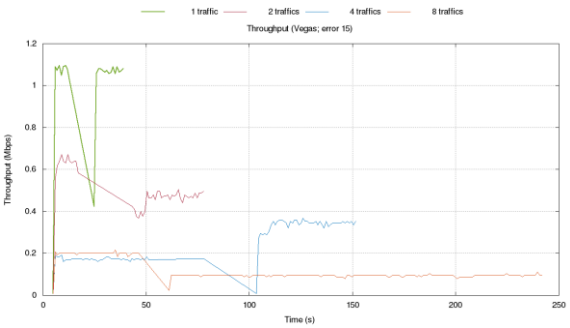

Figura 344 : Throughput vs. Tiempo – TCP Vegas (α=1, β=5) – 15 paquetes perdidos

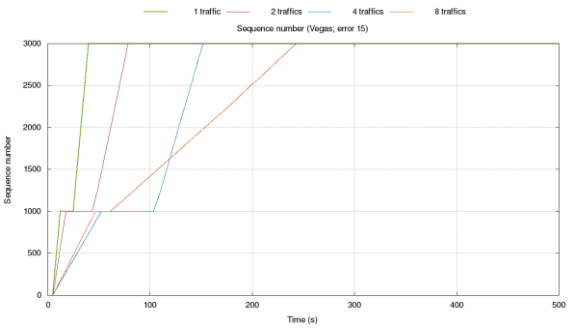

Figura 345 : Nº Secuencia vs. Tiempo – TCP Vegas (α=1, β=5) – 15 paquetes perdidos

Analizando las figuras anteriores y comparándolas con la Figura 330 y con la Figura 331, respectivamente, se observa que el aumento del valor de β no produce efecto sobre el ensayo con 1 flujo, tal cual se verifico en las simulaciones anteriores. En el caso de los ensayos con 2 y 4 flujos, el tiempo que requieren para recuperarse es levemente mayor, lo que produce un pequeño crecimiento en el tiempo total de transmisión. Sin embargo, en el caso de 8 flujos simultáneos, el resultado es lo opuesto, el tiempo de recuperación es más corto lo que produce que el tiempo total de transmisión sea significativamente más corto. En las simulaciones con α=1, β=7 y α=1, β=9 los resultados son muy similares a los resultados con α=1, β=5 (Figura 344 y Figura 345).

En las siguientes figuras (Figura 346 y Figura 347) se observa el throughput y la evolución del número de secuencia para 15 paquetes perdidos de TCP Vegas con α=4, β=6. Analizando las figuras, se observa en los ensayos de 1 y 2 flujos una caída más pronunciada en el momento de la ráfaga de errores, comparado con la configuración α=1, β=3 (Figura 318 y Figura 319). Sin embargo, el tiempo de recuperación es significativamente menor para todos los flujos y la tasa de envío se recupera a valores similares. Esto da como resultado un menor tiempo total de transmisión para 1, 2, 4 y 8 flujos simultáneos en comparación con la configuración por defecto de Vegas.

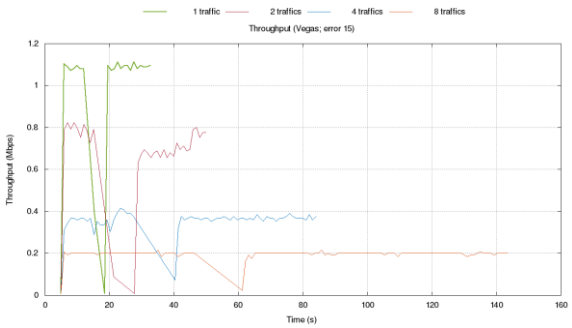

Figura 346 : Throughput vs. Tiempo – TCP Vegas (α=4, β=6) – 15 paquetes perdidos

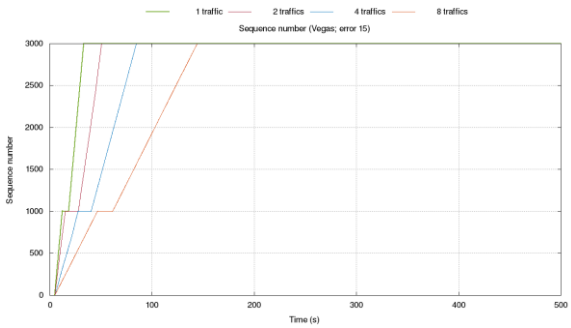

Figura 347 : Nº Secuencia vs. Tiempo – TCP Vegas (α=4, β=6) – 15 paquetes perdidos

Las simulaciones con α=4, β=8 y α=4, β=10 dieron resultados muy similares de TCP Vegas con α=4, β=6 (Figura 346 y Figura 347).

Las siguientes figuras (Figura 348 y Figura 349) corresponden a las simulaciones de 15 paquetes perdidos en forma consecutiva con la configuración de TCP Vegas con α=7 y β=9.

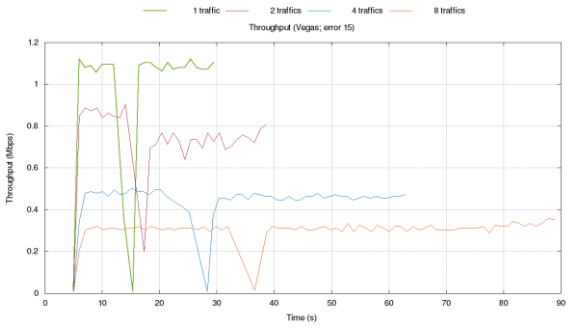

Figura 348 : Throughput vs. Tiempo – TCP Vegas (α=7, β=9) – 15 paquetes perdidos

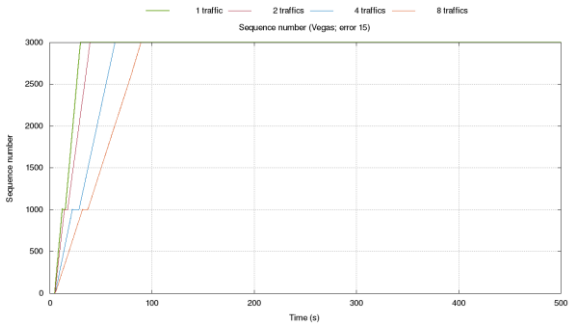

Figura 349 : Nº Secuencia vs. Tiempo – TCP Vegas (α=7, β=9) – 15 paquetes perdidos

A partir de los gráficos anteriores se infiere que la combinación de α y β resultan en una mejora significativa para este grupo de ensayos. Esto queda en evidencia observado que los tiempos de recuperación son menores, las tasas que recuperarán posterior a la ráfaga son del mismo orden lo que implica una reducción sensible en el tiempo necesario para completar la transmisión de todos los datos (tiempo total de transmisión).

Dada esta configuración de parámetros de α y β que permiten mejorar el rendimiento de TCP Vegas en este escenario para un error en ráfaga de 15 paquetes perdidos, resulta interesante observar si esta mejora es general para todas las longitudes de ráfaga ensayadas. A continuación, se muestran las gráficas (Figura 350 a Figura 357) de throughput y número de secuencia en

función del tiempo para la configuración de TCP Vegas con los parámetros α=7 y β=9.

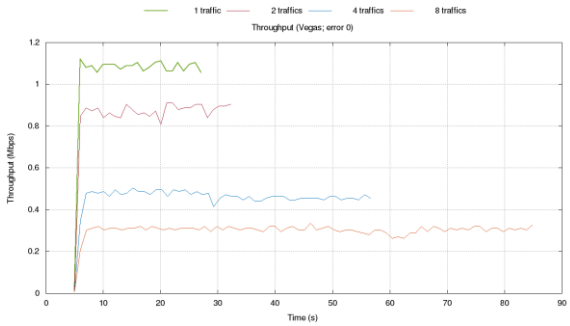

Figura 350 : Throughput vs. Tiempo – TCP Vegas (α=7, β=9) – 0 paquetes perdidos

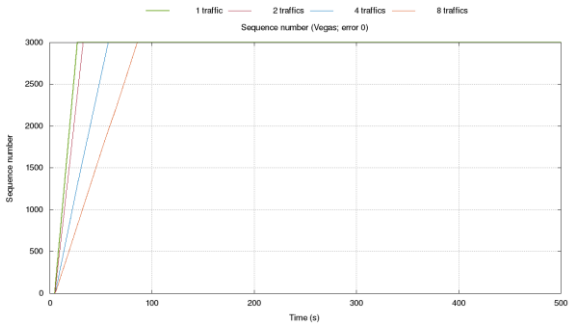

Figura 351 : Nº Secuencia vs. Tiempo – TCP Vegas (α=7, β=9) – 0 paquetes perdidos

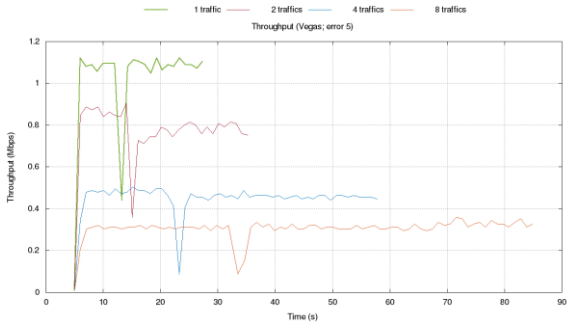

Figura 352 : Throughput vs. Tiempo – TCP Vegas (α=7, β=9) – 5 paquetes perdidos

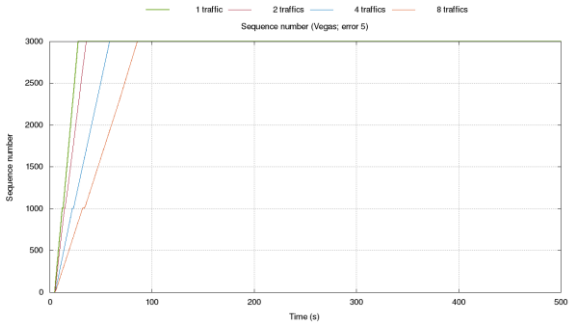

Figura 353 : Nº Secuencia vs. Tiempo – TCP Vegas (α=7, β=9) – 5 paquetes perdidos

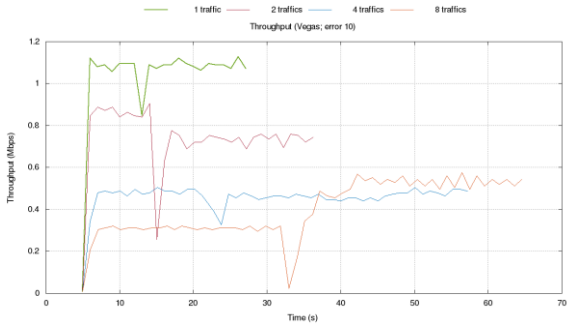

Figura 354 : Throughput vs. Tiempo – TCP Vegas (α=7, β=9) – 10 paquetes perdidos

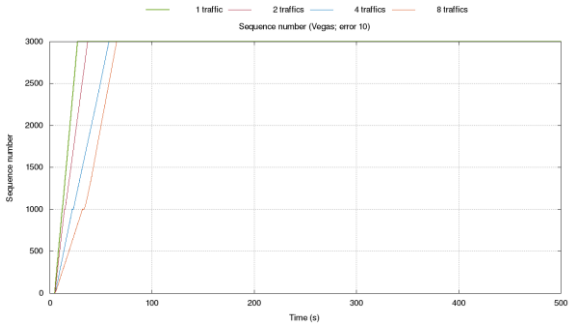

Figura 355 : Nº Secuencia vs. Tiempo – TCP Vegas (α=7, β=9) – 10 paquetes perdidos

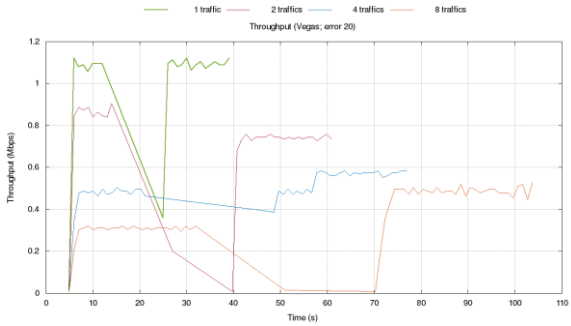

Figura 356 : Throughput vs. Tiempo – TCP Vegas (α=7, β=9) – 20 paquetes perdidos

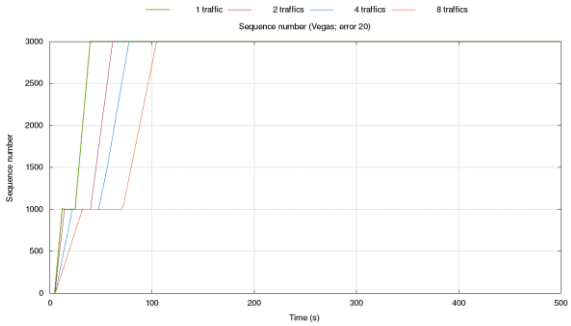

Figura 357 : Nº Secuencia vs. Tiempo – TCP Vegas (α=7, β=9) – 20 paquetes perdidos

Comparando las figuras correspondientes a los ensayos de TCP Vegas con α=1 y β=3 (Figura 324 a Figura 333) con las figuras de TCP Vegas con α=7 y

β=9 (Figura 348 a Figura 357) se observa que el tiempo total de transmisión se reducen drásticamente con los nuevos valores de los parámetros en los distintos ensayos, excepto para la simulación de 1 flujo en el que es poco apreciable. En las figuras de número de secuencia en función del tiempo, se observa que la pendiente de las gráficas es más empinada, lo que se explica dado que el throughput alcanza mayor valor para los nuevos ensayos. En el caso de 8 flujos simultáneos, con la configuración por defecto, el tiempo total de transmisión se encuentra en los alrededores de 160 s, mientras que con los nuevos valores es menor a 90 s.

En el caso de los ensayos de 5 paquetes perdidos, se observa que la caída de throughput (Figura 326 vs. Figura 352) es más importante, pero, la recuperación de la tasa de envío es más rápida. Se observa la mejora a través de la pendiente de las gráficas de número de secuencia (Figura 327 vs. Figura 353) aunque en menor medida para 1 flujo. Se observa que en esta figura la curva de 8 flujos no cruza a la de 4 flujos. Esto se traduce en un tiempo total de transmisión sustancialmente menor sobre todo en los ensayos con 8 flujos. En este caso, con los parámetros por defecto el tiempo total de trasmisión era menor a 250 s y con α=7 yβ=9 menos de 90 s. En relación con esto, se puede observar que el throughput instantáneo que alcanzan es mayor.

Para el caso de comparación de los ensayos de ráfaga de 10 paquetes de longitud (Figura 328 vs. Figura 354) se observa una caída de menor magnitud del throughput y una recuperación más rápida de la tasa de envío para todos los ensayos. El throughput que alcanzan es mayor que con la configuración por defecto, pero en el caso de 1 flujo es casi exigua su diferencia. Esto da como resultado que el tiempo total de transmisión sea sustancialmente menor. Se puede inferir de las figuras de los números de secuencia en función del tiempo (Figura 329 vs. Figura 355), en donde se observan mayores pendientes. Un detalle es que el caso de 8 flujos, se recupera a una tasa de envío mayor de la que traía antes de los errores.

Para el caso de 20 paquetes perdidos (Figura 332 vs. Figura 356) la respuesta es similar al caso anterior, donde se reducen en forma notable los tiempos de transmisión necesarios para los ensayos sin excepción. La comparación de las

figuras de número de secuencia vs. tiempo (Figura 333 vs. Figura 357) de ambas configuraciones poner en evidencia esta sustancial diferencia.

De lo anteriormente analizado, se puede inferir que, de la misma forma de lo observado en simulaciones anteriores, TCP Vegas puede mejorar su respuesta en este escenario.

A continuación, se presentan una serie de gráficos (Figura 358 a Figura 361) donde se grafican el tiempo total de transmisión en función de la longitud de la ráfaga medida en cantidad de paquetes consecutivos perdidos. En ellos se superpone las respuestas de Vegas, con α=1 y β=3, Vegas, con α=7 y β=9, Reno, Westwood y CUBIC. Cada uno representa la contienda de 1, 2, 4 y 8 flujos simultáneos respectivamente.

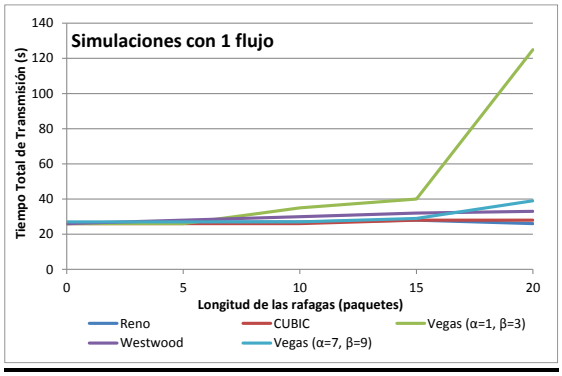

Figura 358: Tiempo Total vs. Long.de la ráfaga 1 Flujo

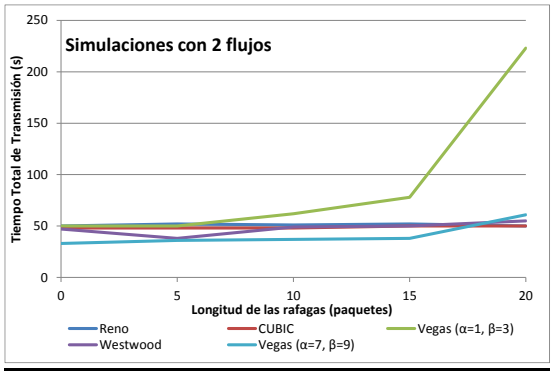

Figura 359: Tiempo Total vs. Long.de la ráfaga 2 Flujos

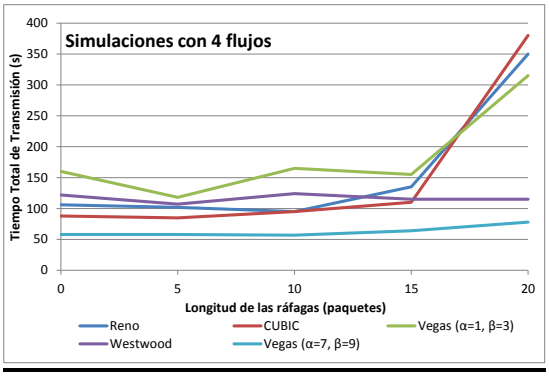

Figura 360: Tiempo Total vs. Long.de la ráfaga 4 Flujos

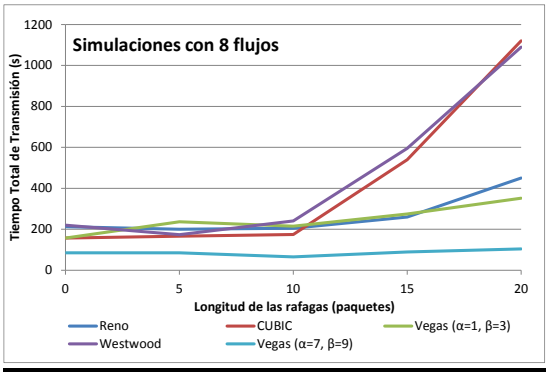

Figura 361: Tiempo Total vs. Long.de la ráfaga 8 Flujos

Las figuras anteriores permiten comparar la métrica de cada una de las variantes de TCP ensayadas a medida que aumenta la cantidad de flujos que compiten por los recursos de la red.

Los valores de la métrica de la Figura 358 son coherentes con lo observado en los ensayos *Errores en ráfaga de distinta longitud en un escenario simple*, donde se describió lo susceptible que resulta ser TCP Vegas desde las longitudes de ráfagas más cortas. En este gráfico resulta evidente que las variantes de TCP mantienen un valor similar del tiempo total de transmisión, es Vegas, con sus parámetros por defecto, quien comienza a extender este tiempo a partir de los 5 paquetes perdidos y, en forma mucho más apreciable, a partir de los 15 paquetes perdidos. Sin embargo, Vegas con los valores de α=7 y β=9, se mantiene dentro del rango de valores del resto, aunque el valor del tiempo necesario comienza a crecer a partir de los 15 paquetes perdidos y es levemente superior al resto para 20 paquetes.

Se observa un comportamiento análogo en la Figura 359, donde otra vez TCP Vegas con sus valores por defecto resulta ser la variante más susceptible a estos errores en este modelo. Por otro lado, cuando se ajustan los parámetros de Vegas a los valores α=7 y β=9, resulta ser la variante que menos tiempo requiere para completar su transmisión hasta los 15 paquetes perdidos, a partir de donde comienza el crecimiento de ese tiempo.

En la figura Figura 360 se observa que, a partir de los 15 paquetes perdidos, cuando compiten 4 flujos de la misma variante por los recursos de la red, Reno, CUBIC y Vegas con los parámetros por defecto, requieren un tiempo significativamente mayor para completar su transmisión de datos. Inclusive para el caso de 20 paquetes perdidos el tiempo que requiere Vegas es levemente menor que los que requieren Reno y CUBIC. Para todas las longitudes de ráfagas ensayadas, Vegas con α=7 y β=9, es la variante que presenta el menor valor de la métrica.

Cuando la contienda es de 8 flujos (Figura 361), los valores de la métrica comienzan a crecer a partir de los 10 paquetes perdidos. Se observa que CUBIC y Westwood son los más sensibles en estas condiciones. Otra vez Vegas con α=7 y β=9 es la que mejor respuesta.

Como se observa en los resultados, a medida que se incrementa la longitud de las ráfagas de errores y la cantidad de flujos simultáneos, el tiempo requerido para completar la transmisión de los datos se incrementa. De las variantes analizadas, es TCP Vegas la que presenta características particulares.

Conclusiones

Debido al crecimiento que han tenido las redes inalámbricas y por consiguiente el uso masivo de estas, es que se ha optado por realizar el presente trabajo, sobre modelos que representen estas topologías. La selección de un modelo heterogéneo, en cuanto a las tecnologías de los enlaces, obedece a que es cada vez es más frecuente que un flujo TCP recorra múltiples tipos de redes en su camino al destino, por lo que resulta relevante verificar el funcionamiento de distintas variantes en estos escenarios. En las evaluaciones del estado del arte, se observó que la mayor parte de las investigaciones han sido realizadas en modelos homogéneos, donde se incluyen exclusivamente enlaces inalámbricos. Sin embargo, las redes actuales son heterogéneas pues, en general, las conexiones terminan en un servidor conectado a una red mediante un enlace cableado.

A través del estudio del estado del arte sobre el tema, también se observó que los mecanismos de control de congestión son particularmente sensibles a los errores en ráfaga y las desconexiones. A partir de esto, resultó interesante conocer la respuesta a estos eventos y el hecho de aumentar la cantidad de nodos inalámbricos no produce demasiado impacto para su primer análisis, dado que el efecto inmediato produce una reacción inmediata. A pesar de ello, en los ensayos se consideró el agregado de nodos para estudiar la utilización y reparto del ancho de banda disponible entre múltiples flujos que comparten la red.

Las pruebas realizadas en el presente trabajo permitieron clasificar y cuantificar los efectos negativos para el rendimiento de TCP que producen las desconexiones frecuentes y los errores en ráfaga característicos de los enlaces inalámbricos. Asimismo, permitió analizar diferentes situaciones, como la competencia de distintos flujos por los recursos de la red, y apreciar las distintas estrategias de los mecanismos de control de congestión implementados en las variantes de TCP ensayadas.

A partir de las pruebas realizadas en el capítulo 4, se pueden destacar de los resultados obtenidos, las siguientes particularidades previo a abordar las conclusiones.

Al producirse desconexiones del nodo móvil, se pudo observar que el throughput promedio se reduce en similares proporciones para todas las variantes a medida que el tiempo de desconexión crece. A pesar de que TCP Vegas presenta un valor promedio levemente inferior, la reducción observada es de similar magnitud a las del resto de las variantes ensayadas. Al analizar el tiempo total de transmisión queda en evidencia que el más afectado es TCP Reno. Para las desconexiones de menor duración, el tiempo requerido para completar la transmisión de los datos es del mismo orden de magnitud. Sin embargo, a partir de los ensayos con un segundo de desconexión, TCP Reno comienza a aumentar el tiempo total de transmisión por encima del valor del resto de las variantes.

Ante la introducción de errores en ráfaga en las simulaciones, se puede analizar cómo son afectadas las diferentes variantes del protocolo TCP, estudiando los gráficos del throughput. En las figuras, resulta evidente la dificultad que presenta tempranamente el protocolo Vegas en recuperarse de un evento de este tipo, cuando se lo compara con las otras variantes ensayadas. Esta situación queda en evidencia a través de la métrica de la evolución del número de secuencia del segmento TCP, donde se observa cómo TCP Vegas demora en recuperar su tasa de envío tempranamente, mientras que el resto de las variantes comienzan a sufrir una degradación evidente para mayores longitudes de ráfagas. De esta manera, muestra una mayor sensibilidad a este tipo de errores desde un primer un momento. Si además se comparan los gráficos de la evolución del tamaño de la ventana de

congestión, se destaca el mayor tiempo que requiere TCP Vegas para recuperar el tamaño de la ventana, es decir, la tasa de envío, después del evento.

En virtud de la respuesta de TCP Vegas con sus parámetros por defecto, α y β, en este escenario, una nueva serie de ensayos pusieron en evidencia que modificando sus valores se puede mejorar su rendimiento. Al realizar las pruebas para distintos valores se observan mejoras significativas en la respuesta del throughput. Tomando con referencia la respuesta de TCP Reno, se logró encontrar un par de valores de parámetros de Vegas que mejoraban su respuesta, haciéndola comparable a las de Reno en estas condiciones y escenario. Esta respuesta queda en evidencia en los gráficos de las métricas tiempo total de transmisión y throughput promedio, donde se comparan las variantes de TCP Westwood, Reno, CUBIC y Vegas con las dos configuraciones, y se puede apreciar la mejora significativa de TCP Vegas con estos nuevos parámetros en estas condiciones.

Se puede concluir entonces que, para el modelo implementado, los rangos de valores de los parámetros α y β considerados y de las longitudes de los errores en ráfaga introducidos, resulta que α domina la escena y condiciona la respuesta de los parámetros evaluados. El parámetro β está relacionado con la reducción de la ventana de congestión cuando se deduce un crecimiento en las colas de los nodos intermedios y una congestión incipiente. En el escenario analizado, la presencia de un solo flujo TCP Vegas y su naturaleza proactiva no llevan en ningún momento al estado cercano a la congestión de la red, por lo que no se requiere reducir el tamaño de su ventana de congestión.

Por último, cuando distintos flujos compiten por los recursos de una red, analizando los datos y los gráficos obtenidos se muestra que existen variantes del protocolo TCP que se evidencian más agresivas, captando todo el ancho de banda posible para transmitir los datos, mientras que otras muestran ser particularmente vulnerables contra sus congéneres. De esta manera, se pudo detectar que algunas de las propuestas no pueden coexistir con otras, ya sea porque se apropian o ceden prácticamente la totalidad de los recursos disponibles, lo que las hace solo aplicables en escenarios particulares y muy acotados. Esto último no es lo que ocurre en general, dado que las conexiones

extremo a extremo de TCP trascurren en redes heterogéneas que presentan diferentes tipos de enlaces en su camino. Sin embargo, el comportamiento que se observa en la gran mayoría de las confrontaciones, demuestra que los algoritmos de control de congestión negocian su tránsito a un estado estable, distribuyendo de una manera equitativa el ancho de banda disponible. La principal diferencia se da en el tiempo empleado para llegar un estado de equilibrio.

En forma general, las variantes TCP con controles de congestión desarrolladas para redes con altos valores BDP y algoritmos con carácter reactivo, tales como YeAH e Hybla entre otros, presentan una notable agresividad. Esta característica determina la existencia de inestabilidad en la negociación del ancho de banda y explica por qué tardan tiempos significativos en alcanzar un estado equilibrado, como se observa en los casos de Veno vs. Hybla, y Veno vs. YeAH. En las contiendas donde intervienen protocolos como Westwood, también reactivos pero basados en la estimación del ancho de banda, se puede observar que su comportamiento es más amigable a la hora de negociar recursos. De esta manera, la mayoría de los casos llegan a un estado equilibrado en muy poco tiempo.

Es interesante destacar que TCP Vegas resulta particularmente sensible respecto al resto de las variantes. En los ensayos realizados para este trabajo, se pudo observar cómo esta variante resigna rápidamente su participación en el uso del ancho de banda del canal. El motivo de esto es que el algoritmo de control de congestión que posee Vegas no le permite sostener un uso de ancho de banda cuando compite con los restantes protocolos seleccionados para realizar las pruebas. En el otro extremo, protocolos como Hybla, Illinois y aquellos destinados a redes de alta velocidad, tienden a dominar a sus contrapartes en los ensayos e intentan capturar el mayor ancho de banda posible.

Dada la discusión de los resultados de los ensayos donde se introdujeron desconexiones de distinta duración, errores en ráfaga con distintas cantidades de paquetes perdidos y la contienda de flujos TCP por los recursos de la red, se puede concluir que no existe una variante que sobresalga claramente sobre el resto en el escenario analizado. Este hecho queda en evidencia al observar

la cantidad de variantes que se desarrollaron del protocolo TCP. Sin embargo, se pueden evaluar las distintas variantes, para los escenarios y las condiciones planteadas, a lo largo de las pruebas de este trabajo y obtener así, un orden de mérito.

En primera instancia, de acuerdo a lo descrito anteriormente, se descartan las variantes de TCP que no logran compartir equitativamente los recursos de la red (fairness), debido a que a lo largo de la conexión los paquetes atraviesan enlaces heterogéneos y flujos TCP que pueden pertenecer a distintas variantes. Entonces, en primer lugar, se descarta Vegas, pues como se analizó no puede sostener el uso de los recursos de la red excepto contra sí mismo. En el otro extremo se descarta Hybla, pues acapara la mayor parte del ancho de banda, inclusive cuando compite contra sí mismo. Estas dos variantes fueron las que tuvieron los peores valores del índice de equidad. CUBIC, Reno y Westwood fueron los que obtuvieron los valores más altos de la métrica, y levemente menor Compound. En cuanto al tiempo de convergencia las contiendas de Westwood fueron las que menor tiempo requirieron para llegar a un estado de reparto justo del ancho de banda. Lo siguieron CUBIC, Reno y Compound, en ese orden.

Cuando se analizan las desconexiones de distinta duración, las variantes comparadas reducen su throughput promedio en forma similar a medida que se incrementa el tiempo de desconexión. Sin embargo, cuando se analiza el tiempo requerido para completar la trasmisión de los datos, es CUBIC quien menos tiempo requiere en los distintos ensayos, seguido por Compound y Westwood. Por el contrario, es Reno el que más demora en completar la transmisión. En los gráficos de la evolución del número de secuencia de TCP Reno, se observa que TCP Reno requiere mayor tiempo para recuperar la tasa de envío, debido a los crecientes períodos de inactividad producidos por el backoff exponencial que hace crecer los valores de RTO. Esto se produce porque cada timeout en serie reduce su ventana de congestión y pasa a la fase de Slow Start. Dado que las desconexiones también impiden el arribo de los ACK, no se disparan los algoritmos de Fast Retransmit ni Fast Recovery. En estas condiciones y sobre este escenario, su rendimiento es muy similar que al de Tahoe. En cambio, en CUBIC, la ventana de congestión sigue una función

cúbica, lo que le permite ser más escalable y alcanzar más rápidamente la tasa de envío anterior a las desconexiones.

Cuando se analizan los efectos de los errores en ráfaga, el comportamiento es similar en las variantes comparadas. Sin embargo, cuando se observan los gráficos de throughput se puede inferir que Reno y CUBIC se muestran más susceptibles para las distintas ráfagas de errores que Westwood.

Por lo expresado anteriormente, se puede decir que TCP Westwood fue el que presentó el mejor desempeño para los requerimientos que se plantearon en estos ensayos.

Como se describió, este estudio se realizó en base a simulaciones y no en un ambiente real debido a la complejidad de implementación de las versiones de TCP y de los escenarios en diferentes maquetas. Aunque los resultados pueden no reflejar con exactitud los resultados, la utilización de simuladores es un recurso ampliamente extendido en los ambientes de investigación.

En cuanto a los trabajos a futuro, se propone contrastar estas pruebas con las realizadas en un modelo real y comparar los resultados con los obtenidos en este trabajo. También se deben complejizar los modelos de estudio, aumentando la cantidad de nodos y de flujos simultáneos en contienda, introduciendo movilidad a los nodos, incluir distintos tipos de tráficos y enlaces con distintas tecnologías que representen con mayor precisión la heterogeneidad de internet, y extender el análisis a otros protocolos subyacentes de capa de transporte propuestos para resolver los problemas de rendimiento en las redes inalámbricas.

Anexo A - Scripts

A.1. Simulaciones

El script que se detalla continuación, es una versión adaptada del original utilizado para las simulaciones incluidas en el presente trabajo. La idea es la de presentar la forma, pero no pretende ser exhaustivo en cuanto a funcionalidades usadas. Algunas de las opciones incluidas contemplan casos que escapan a los objetivos del trabajo. El script que se detalla corresponde al último grupo de simulaciones, que representa el escenario más complejo.

```
# ======================================================================================
#
#         Descripción:
#         Este script ejecuta una simulación de 8 tráficos TCP simultáneos,
#         entre 8 nodos cableados y 8 nodos inalámbricos. Además cuenta con
#         la estación base y un nodo cableado central al que se conectan los
#         nodos cableados.
#
#         Topología:
#         Nodos cableados: 9
#         Nodos inalámbricos: 8
#         Nodos híbridos: 1 (Estación base: interfaz cableada e inalámbrica)
#
#         Variante predeterminada: Westwood
#         Paquetes transmitidos: 3000
#         Error en ráfaga: 15
#
#         Ejecución:
#         ns wcw-8_nodos-TCP.tcl
#
# ======================================================================================
# ____________________________________________________________________________________
#         Definir Opciones
#                   Se definir variables globales al principio del script.
#
#                   Una convención adoptada por muchos investigadores que trabajan
#                   con NS2, es utilizar un array para almacenar los valores.
#
# ____________________________________________________________________________________
global val
# ____________________________________________________________________________________
# Parámetros para interfaces inalámbricas
# ____________________________________________________________________________________
set val(chan)           Channel/WirelessChannel  ;# Tipo de canal
```

```
set val(prop)           Propagation/TwoRayGround           ;# Modelo de propagación de radio
set val(netif)          Phy/WirelessPhy                    ;# Tipo de interfaz de red
set val(mac)            Mac/802_11                         ;# Tipo de MAC
set val(ifq)            Queue/DropTail/PriQueue            ;# Tipo de interfaz de cola
set val(ll)             LL                                 ;# Tipo de capa de enlace
set val(ant)            Antenna/OmniAntenna                ;# Modelo de antena
set val(ifqlen)         50                                 ;# Cantidad máxima paquetes en cola
set val(rp)             DSDV                               ;# Protocolo de enrutamiento
set val(err)                    UniformErrorProc
set val(FECstrength)            1                                  ;# Necesario para introducir errores
# ______________________________________________________________________________________
# Parámetros de la simulación
# ______________________________________________________________________________________
set val(x)                              100                ;#Extensión en X del escenario
set val(y)                              100                ;#Extensión en y del escenario
set val(tr)                             trf.tr             ;#Nombre del archivo de traza
set val(namtr)                          namtrf.nam         ;#Nombre del archivo Network Animator
set val(end_sim)            200.0               ;# Final de la simulación
set val(mobile_nodes)       8                   ;# Número de nodos móviles
set val(wired_nodes)        9                   ;# Cantidad de nodos cableados
set val(bs_nodes)                       1                  ;# Cantidad de estaciones base
# Direcciones de los nodos cableados
set tempW {0.0.0 0.0.1 0.0.2 0.0.3 0.0.4 0.0.5 0.0.6 0.0.7 0.0.8}
# Direcciones de los nodos inalámbricos
set temp {1.1.0 1.1.1 1.1.2 1.1.3 1.1.4 1.1.5 1.1.6 1.1.7 1.1.8}
set vel                                 100.0
set val(delay)                          0.1ms     ;# Setear latencia a 0.1ms
set val(bandwidth)          10Mb        ;# Limitar ancho de banda cableado a 10MB
Mac/802_11 set dataRate_ 1Mb            ;# Limitar ancho de banda inalámbrico a 1MB
set val(TCPvar) "Westwood"
# ======================================================================================
# Programa principal
# ======================================================================================
# ______________________________________________________________________________________
# Initializar variables globales y archivos de traza
# ______________________________________________________________________________________
# Crear objeto simulador
set ns_   [new Simulator]
$ns_ color 1 Blue
$ns_ color 2 Red
# Configuracion de enrutamiento jerarquico
$ns_ node-config -addressType hierarchical
AddrParams set domain_num_ 2                      ; # Numero de dominios
lappend cluster_num 1 2                           ; # Numero de clusters por dominio
AddrParams set cluster_num_ $cluster_num
# Numero de nodos por cluster
lappend eilastlevel $val(wired_nodes) $val(bs_nodes) $val(mobile_nodes)
AddrParams set nodes_num_ $eilastlevel  ; # de cada domino 9 1 8

# Crear archivo de traza inalámbrico
set tracefd     [open $val(tr) w]
$ns_ use-newtrace
$ns_ trace-all $tracefd

# Crear archivo de traza de NAM
set ntf [open $val(namtr) w]
$ns_ namtrace-all-wireless $ntf $val(x) $val(y)

# Procedimiento de salida
proc finish {} {
    global ns_ tracefd ntf val(namtr) val(end_sim)
    puts "Terminando simulación a los $val(end_sim) segundos"

    $ns_ flush-trace
    close $tracefd
    close $ntf
#    exec nam $namtr &
          exit 0
}
```

```
# Definir topografia del objeto
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)

# Crear objeto God (General Operations Director) para nodos con interfaces inalámbricas
create-god [expr $val(mobile_nodes) + $val(bs_nodes)]

# _____________________________________________________________________________
# Crear nodos cableados (DOMINIO 0)
for {set i 0} {$i < $val(wired_nodes) - 1} {incr i} {
          set wnode($i) [$ns_ node [lindex $tempW $i]]
}
# Establecer conexión entre estación base y nodo cableado central
set wnode([expr $val(wired_nodes) - 1]) [$ns_ node [lindex $tempW [expr $val(wired_nodes) - 1]]]
# _____________________________________________________________________________
# Crear nodos inalambricos y estación base
# Crear canal #1
set chan_1_ [new $val(chan)]

# Configuración inalámbricas para estación base
$ns_ node-config -adhocRouting $val(rp) \
                    -llType $val(ll) \
                    -macType $val(mac) \
                    -ifqType $val(ifq) \
                    -ifqLen $val(ifqlen) \
                    -antType $val(ant) \
                    -propInstance [new $val(prop)] \
                    -propType $val(prop) \
                    -phyType $val(netif) \
                    -topoInstance $topo \
                    -wiredRouting ON \
                    -agentTrace ON \
                    -routerTrace ON \
                    -macTrace ON \
                    -movementTrace OFF \
                    -channel $chan_1_
# _____________________________________________________________________________
# Crear BS           (DOMINIO 1)
set BS0 [$ns_ node 1.0.0]
$BS0 random-motion 0

# Definir coordenadas iniciales para el bs (x, y, z=0)
$BS0 set X_ 50.0
$BS0 set Y_ 50.0
$BS0 set Z_ 0.0
# _____________________________________________________________________________
# Configuracion adicional para nodos móviles
$ns_ node-config -wiredRouting OFF \
                 -agentTrace ON \
                 -routerTrace OFF \
                 -macTrace OFF
# _____________________________________________________________________________
# Errores
# _____________________________________________________________________________
# Error en ráfaga
set e_list 5
proc errorModelListWireless { } {
          global val e_list

          set errormodel [new ErrorModel/List]
          set errList $e_list

          if { $errList == 0 } {
                    puts "Simulación sin ráfaga de errores"
          }
          if { $errList == 1 } {
                    $errormodel droplist {1000}
          }
          if { $errList == 5 } {
```

```
                $errormodel droplist {1000 1001 1002 1003 1004}
        }
        if { $errList == 10 } {
                $errormodel droplist {1000 1001 1002 1003 1004 1005 1006 1007 1008 1009}
        }
        if { $errList == 15 } {
                $errormodel droplist {1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014}
        }
        if { $errList == 20 } {
                $errormodel droplist {1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014
1015 1016 1017 1018 1019}
        }

        $errormodel unit packet
        $errormodel FECstrength $val(FECstrength)
        $errormodel datapktsize 1000
    $errormodel cntrlpktsize 80

        #puts "Error list: $e_list"
        return $errormodel
}
#set e_model errorModelListWireless                      ; # Descomentar para activar
# ____________________________________________________________________________________
# Crear nodos inalambricos (DOMINIO 2)
for {set j 0} {$j < $val(mobile_nodes)} {incr j} {
    set node_($j) [ $ns_ node [lindex $temp $j] ]
    $node_($j) base-station [AddrParams addr2id [$BS0 node-addr]]
}

# Definir coordenadas iniciales para los nodos moviles (x, y, z=0)
$node_(0) set X_ 75.0
$node_(0) set Y_ 70.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 75.0
$node_(1) set Y_ 30.0
$node_(1) set Z_ 0.0
# ____________________________________________________________________________________
# ENLACES
# ____________________________________________________________________________________
# Crear enlaces entre nodos cableados y nodos cableado central
for {set j 0} {$j < $val(wired_nodes) - 1} {incr j} {
        $ns_ duplex-link $wnode($j) $wnode([expr $val(wired_nodes) - 1]) val(bandwidth) val(delay) DropTail
}
$ns_ duplex-link-op $wnode(0) $wnode([expr $val(wired_nodes) - 1]) orient right-down
$ns_ duplex-link-op $wnode(1) $wnode([expr $val(wired_nodes) - 1]) orient right-up

# Crear enlaces entre estación base y nodo cableado central
$ns_ duplex-link $wnode([expr $val(wired_nodes) - 1]) $BS0 100Mb 0.001ms DropTail
$ns_ duplex-link-op $wnode([expr $val(wired_nodes) - 1]) $BS0 orient right
# ____________________________________________________________________________________
# ERRORES
# ____________________________________________________________________________________
set noderr1 $wnode([expr $val(wired_nodes) - 1])
set noderr2 $BS0

#Error estadístico
proc errorModelSimpleWired { {e_rate 0.001} } {
        global ns_ noderr1 noderr2

        set error_rate $e_rate                  ;# Error de 0.00001 = 0.001 / (100 - 0)
        # Crear una distribución uniforme de la variable de error
        set loss_random_variable [new RandomVariable/Uniform]
        $loss_random_variable set min_ 0
        $loss_random_variable set max_ 100

        set errormodel [new ErrorModel]
        $errormodel drop-target [new Agent/Null]
        $errormodel set rate_ $error_rate
```

```
          $errormodel set unit pkt
          $errormodel ranvar $loss_random_variable

          puts "Error rate: $error_rate "

          # Insertar modulo de error en conexion
          $ns_ lossmodel $errormodel $noderr1 $noderr2
}
#set we_rate 0.001                              ;# 0.01 -- 1%
#errorModelSimpleWired $we_rate                 ; # Descomentar para activar

# Error en ráfaga
proc errorModelListWired  { {e_list 10} } {
          global ns_ noderr1 noderr2 errList

          set errormodel [new ErrorModel/List]
          set errList $e_list

          if { $errList == 0 } {
                    puts "Simulación sin ráfaga de errores"
          }
          if { $errList == 1 } {
                    $errormodel droplist {1000}
          }
          if { $errList == 5 } {
                    $errormodel droplist {1000 1001 1002 1003 1004}
          }
          if { $errList == 10 } {
                    $errormodel droplist {1000 1001 1002 1003 1004 1005 1006 1007 1008 1009}
          }
          if { $errList == 15 } {
                    $errormodel droplist {1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014}
          }
          if { $errList == 20 } {
                    $errormodel droplist {1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014
1015 1016 1017 1018 1019}
          }

          puts "Error list: $e_list"
          # Insertar modulo de error en conexion
          if { $errList > 0 } {
                    $ns_ lossmodel $errormodel $noderr1 $noderr2
          }
}
set we_list 15
errorModelListWired $we_list                   ; # Descomentar para activar
# _____________________________________________________________________________
# COLAS
# _____________________________________________________________________________
# Limite de colas
set QSize0 50
set QSize1 50

#  Máximo buffer de la cola entre enlaces
$ns_ queue-limit $wnode(0) $wnode([expr $val(wired_nodes) - 1]) $QSize0
$ns_ queue-limit $wnode(1) $wnode([expr $val(wired_nodes) - 1]) $QSize0
# _____________________________________________________________________________
# MOVIMIENTO
# _____________________________________________________________________________
$ns_ at 0.1 "$node_(0) setdest 75.0 75.0 $vel"
$ns_ at 0.1 "$node_(1) setdest 75.0 25.0 $vel"
# _____________________________________________________________________________
# TRAFICO
# _____________________________________________________________________________
set n 0
proc trafficTCP { {node1} {node2} {tcpVariant Tahoe} {transType "P"} {begin 0.0} {endTime 10.0} {pkts 3000} {pktSize
1000} {alphaPar 1} {betaPar 3} {tcpTk 0.1} {ovh 0} {winInit 1} {winOp 1} {winCon 4} {mrto 64} } { ; # nodo 1, nodo 2,
tiempo de inicio y cantidad de paquetes
    global ns_ n
```

```
if { $tcpVariant == "Tahoe"} {
          puts "Tahoe"

          set tcp($n) [new Agent/TCP]
          $tcp($n) set tcpTick_ $tcpTk
          $tcp($n) set overhead_ $ovh
          $tcp($n) set windowInit_ $winInit
          $tcp($n) set windowOption_ $winOp
          $tcp($n) set windowConstant_ $winCon
          $tcp($n) set maxrto_ $mrto
} elseif { $tcpVariant == "Reno"} {
          puts "Reno"

          set tcp($n) [new Agent/TCP/Reno]
} elseif { $tcpVariant == "Newreno"} {
          puts "New Reno"

          set tcp($n) [new Agent/TCP/Newreno]
} elseif { $tcpVariant == "Sack"} {
          puts "Sack"

          set tcp($n) [new Agent/TCP/Sack1]
} elseif { $tcpVariant == "Fack"} {
          puts "Fack"

          set ss-div4 1
          set rampdown 1

          set tcp($n) [new Agent/TCP/Fack]
} elseif { $tcpVariant == "Vegas"} {
          puts "Vegas"

          set tcp($n) [new Agent/TCP/Vegas]
          $tcp($n) set timestamps_ true
          $tcp($n) set v_alpha_ $alphaPar
          $tcp($n) set v_beta_ $betaPar
} elseif { $tcpVariant == "Linux-Vegas" } {
          puts "Vegas base on Linux"

          set tcp($n) [new Agent/TCP/Linux]
          $tcp($n) set maxrto_ 120
          $tcp($n) set ts_resetRTO_ true
          $tcp($n) set delay_growth_ false
          $ns_ at $begin "$tcp($n) select_ca vegas"
          # Set global params for TCP
          #$ns_ at $begin "$tcp($n) set_ca_default_param vegas alpha $alphaPar"
          #$ns_ at $begin "$tcp($n) set_ca_default_param vegas beta $betaPar"
          # Set local params only for tcp(n)
          $ns_ at $begin "$tcp($n) set_ca_param vegas alpha $alphaPar"
          $ns_ at $begin "$tcp($n) set_ca_param vegas beta $betaPar"
} elseif { $tcpVariant == "Linux"} {
          puts "Linux"

          set tcp($n) [new Agent/TCP/Linux]
          $tcp($n) set maxrto_ 120
          $tcp($n) set ts_resetRTO_ true
          $tcp($n) set delay_growth_ false
} elseif { $tcpVariant == "Cubic"} {
          puts "Cubic"

          set tcp($n) [new Agent/TCP/Linux]
          $tcp($n) set maxrto_ 120
          $tcp($n) set ts_resetRTO_ true
          $tcp($n) set delay_growth_ false
          $ns_ at $begin "$tcp($n) select_ca cubic"
} elseif { $tcpVariant == "Westwood"} {
          puts "Westwood"
```

```
          set tcp($n) [new Agent/TCP/Linux]
          $tcp($n) set maxrto_ 120
          $tcp($n) set ts_resetRTO_ true
          $tcp($n) set delay_growth_ false
          $ns_ at $begin "$tcp($n) select_ca westwood"
} elseif { $tcpVariant == "Hybla"} {
          puts "Hybla"

          set tcp($n) [new Agent/TCP/Linux]
          $tcp($n) set maxrto_ 120
          $tcp($n) set ts_resetRTO_ true
          $tcp($n) set delay_growth_ false
          $ns_ at $begin "$tcp($n) select_ca hybla"
} elseif { $tcpVariant == "Bic"} {
          puts "Bic"

          set tcp($n) [new Agent/TCP/Linux]
          $tcp($n) set maxrto_ 120
          $tcp($n) set ts_resetRTO_ true
          $tcp($n) set delay_growth_ false
          $tcp($n) set windowOption_ 8
          $tcp($n) set max_ssthresh_ 100
          $tcp($n) set cwnd_range_ 50
          $ns_ at $begin "$tcp($n) select_ca bic"
} elseif { $tcpVariant == "Highspeed"} {
          puts "Highspeed"

          set tcp($n) [new Agent/TCP/Linux]
          $tcp($n) set maxrto_ 120
          $tcp($n) set ts_resetRTO_ true
          $tcp($n) set delay_growth_ false
          $ns_ at $begin "$tcp($n) select_ca highspeed"
} elseif { $tcpVariant == "Veno"} {
          puts "Veno"

          set tcp($n) [new Agent/TCP/Linux]
          $tcp($n) set maxrto_ 120
          $tcp($n) set ts_resetRTO_ true
          $tcp($n) set delay_growth_ false
          $ns_ at $begin "$tcp($n) select_ca veno"
} elseif { $tcpVariant ==  "Illinois"} {
          puts  "Illinois"

          set tcp($n) [new Agent/TCP/Linux]
          $tcp($n) set maxrto_ 120
          $tcp($n) set ts_resetRTO_ true
          $tcp($n) set delay_growth_ false
          $ns_ at $begin "$tcp($n) select_ca illinois"
} elseif { $tcpVariant == "Compound" } {
          puts "Compound"

          set tcp($n) [new Agent/TCP/Linux]
          $tcp($n) set maxrto_ 120
          $tcp($n) set ts_resetRTO_ true
          $tcp($n) set delay_growth_ false
          $ns_ at $begin "$tcp($n) select_ca compound"
} elseif { $tcpVariant == "YeAH" } {
          puts "YeAH"
          set tcp($n) [new Agent/TCP/Linux]
          $tcp($n) set maxrto_ 120
          $tcp($n) set ts_resetRTO_ true
          $tcp($n) set delay_growth_ false
          $ns_ at $begin "$tcp($n) select_ca yeah"
} elseif { $tcpVariant == "HTCP" } {
          puts "HTCP"

          set tcp($n) [new Agent/TCP/Linux]
          $tcp($n) set maxrto_ 120
          $tcp($n) set ts_resetRTO_ true
```

```
                $tcp($n) set delay_growth_ false
                $ns_ at $begin "$tcp($n) select_ca htcp"
        } elseif { $tcpVariant == "LowPriority" } {
                puts "Low Priority"

                set tcp($n) [new Agent/TCP/Linux]
                $tcp($n) set maxrto_ 120
                $tcp($n) set ts_resetRTO_ true
                $tcp($n) set delay_growth_ false
                $ns_ at $begin "$tcp($n) select_ca lp"
        } else {
                puts "Tahoe (default)"
                set tcp($n) [new Agent/TCP]
                $tcp($n) set tcpTick_ $tcpTk
                $tcp($n) set overhead_ $ovh
                $tcp($n) set windowInit_ $winInit
                $tcp($n) set windowOption_ $winOp
                $tcp($n) set windowConstant_ $winCon
                $tcp($n) set maxrto_ $mrto
        }
        $tcp($n) set class_ [expr $n + 1]
        $tcp($n) set fid_ [expr $n + 1]

        $tcp($n) set packetSize_ $pktSize

        set sink($n) [new Agent/TCPSink/Sack1]
                $sink($n) set ts_echo_rfc1323_ true
                $ns_ attach-agent $node1 $tcp($n)
                $ns_ attach-agent $node2 $sink($n)
                $ns_ connect $tcp($n) $sink($n)

        set ftp($n) [new Application/FTP]
                $ftp($n) attach-agent $tcp($n)

                $ns_ at $begin "$ftp($n) start"
                $ns_ at $endTime  "$ftp($n) stop"

        puts "Comenzando tcp($n) a los $begin segundos"

        # CWND
        set outfileCWND($n) [open  "CWND-tcp($n)"  w]
                $ns_  at  $begin  "plotCWND $tcp($n)  $outfileCWND($n)"

        # SEQN
        set outfileSEQN($n) [open  "SEQN-tcp($n)"  w]
                $ns_  at  $begin  "plotSEQN $tcp($n)  $outfileSEQN($n)"
        # GOOD
        set outfileGOOD($n) [open  "GOOD-tcp($n)"  w]
                set oldTimeG [$ns_ now]
                $ns_  at  $begin  "plotGOOD $tcp($n) $sink($n) $outfileGOOD($n) $oldTimeG"
        incr n
}
# ____________________________________________________________________________
# Tráficos

set nodesrc $wnode(0)
set nodedst $node_(0)

# Valores predeterminados para TCP
set pkts 3000
set pktSize 1000
set alphaPar 1
set betaPar 3
set tcpTk 0.1
set ovh 0
set winInit 1
set winOp 1
set winCon 4
set mrto 64
```

```
# Tráfico TCP
#trafficTCP { {node1} {node2} {tcpVariant Tahoe} {transType "p"} {begin 0.0} {endTime 10.0} {pkts 1000} {pktSize 1000}
{alphaPar 1} {betaPar 3} {tcpTk 0.1} {ovh 0} {winInit 1} {winOp 1} {winCon 4} {mrto 64} }

# Trafico 0
set nodesrc $wnode(0)
set nodedst $node_(0)
set startTCP0                5.0
set stopTCP0                 105.0
set transType "t"
set tcpVar $val(TCPvar)
puts "Generando tráfico $tcpVar entre los nodos 0 y 10, de $startTCP0 a $stopTCP0 segundos"
trafficTCP $nodesrc $nodedst $tcpVar $transType $startTCP0 $stopTCP0 $pkts $pktSize $alphaPar $betaPar $tcpTk $ovh
$winInit $winOp $winCon $mrto

# TRÁFICO 1
set nodesrc $wnode(1)
set nodedst $node_(1)
set startTCP1                15.0
set stopTCP1                 115.0
set transType "t"
set tcpVar $val(TCPvar)
puts "Generando tráfico $tcpVar entre los nodos 1 y 11, de $startTCP1 a $stopTCP1 segundos"
trafficTCP $nodesrc $nodedst $tcpVar $transType $startTCP1 $stopTCP1 $pkts $pktSize $alphaPar $betaPar $tcpTk $ovh
$winInit $winOp $winCon $mrto

# TRÁFICO 2
set nodesrc $wnode(2)
set nodedst $node_(2)
set startTCP2                5.0
set stopTCP2                 105.0
set transType "t"
set tcpVar $val(TCPvar)
puts "Generando tráfico $tcpVar entre los nodos 2 y 12, de $startTCP2 a $stopTCP2 segundos"
trafficTCP $nodesrc $nodedst $tcpVar $transType $startTCP2 $stopTCP2 $pkts $pktSize $alphaPar $betaPar $tcpTk $ovh
$winInit $winOp $winCon $mrto

# TRÁFICO 3
set nodesrc $wnode(3)
set nodedst $node_(3)
set startTCP3                15.0
set stopTCP3                 115.0
set transType "t"
set tcpVar $val(TCPvar)
puts "Generando tráfico $tcpVar entre los nodos 3 y 13, de $startTCP3 a $stopTCP3 segundos"
trafficTCP $nodesrc $nodedst $tcpVar $transType $startTCP3 $stopTCP3 $pkts $pktSize $alphaPar $betaPar $tcpTk $ovh
$winInit $winOp $winCon $mrto

# TRÁFICO 4
set nodesrc $wnode(4)
set nodedst $node_(4)
set startTCP4                5.0
set stopTCP4                 105.0
set transType "t"
set tcpVar $val(TCPvar)
puts "Generando tráfico $tcpVar entre los nodos 4 y 14, de $startTCP4 a $stopTCP4 segundos"
trafficTCP $nodesrc $nodedst $tcpVar $transType $startTCP4 $stopTCP4 $pkts $pktSize $alphaPar $betaPar $tcpTk $ovh
$winInit $winOp $winCon $mrto
# TRÁFICO 5
set nodesrc $wnode(5)
set nodedst $node_(5)
set startTCP5                15.0
set stopTCP5                 115.0
set transType "t"
set tcpVar $val(TCPvar)
puts "Generando tráfico $tcpVar entre los nodos 5 y 15, de $startTCP5 a $stopTCP5 segundos"
trafficTCP $nodesrc $nodedst $tcpVar $transType $startTCP5 $stopTCP5 $pkts $pktSize $alphaPar $betaPar $tcpTk $ovh
$winInit $winOp $winCon $mrto
```

```
# TRÁFICO 6
set nodesrc $wnode(6)
set nodedst $node_(6)
set startTCP6                5.0
set stopTCP6                 105.0
set transType "t"
set tcpVar $val(TCPvar)
puts "Generando tráfico $tcpVar entre los nodos 6 y 16, de $startTCP6 a $stopTCP6 segundos"
trafficTCP $nodesrc $nodedst $tcpVar $transType $startTCP6 $stopTCP6 $pkts $pktSize $alphaPar $betaPar $tcpTk $ovh
$winInit $winOp $winCon $mrto

# TRÁFICO 7
set nodesrc $wnode(7)
set nodedst $node_(7)
set startTCP7                15.0
set stopTCP7                 115.0
set transType "t"
set tcpVar $val(TCPvar)
puts "Generando tráfico $tcpVar entre los nodos 7 y 17, de $startTCP7 a $stopTCP7 segundos"
trafficTCP $nodesrc $nodedst $tcpVar $transType $startTCP7 $stopTCP7 $pkts $pktSize $alphaPar $betaPar $tcpTk $ovh
$winInit $winOp $winCon $mrto
# ____________________________________________________________________________________
# Traza de parámetros
# ____________________________________________________________________________________
# CWND
proc plotCWND {tcpSource outfileCWND} {
     global ns_

     set now [$ns_ now]
     set cwnd [$tcpSource set cwnd_]

          #  Mostrar TIME CWND para gnuplot
     puts  $outfileCWND  "$now $cwnd"

     $ns_ at [expr $now+0.1] "plotCWND  $tcpSource  $outfileCWND"
}
# ____________________________________________________________________________________
# SEQN (Sequence number)
proc plotSEQN {tcpSource outfileSEQN} {
     global ns_

     set now [$ns_ now]
     set t_seqno [$tcpSource set t_seqno_]

          #  Mostrar TIME SEQN para gnuplot
     puts  $outfileSEQN  "$now $t_seqno"

     $ns_ at [expr $now+0.1] "plotSEQN  $tcpSource  $outfileSEQN"
 }
# ____________________________________________________________________________________
# Finalizar de la simulacion
# ____________________________________________________________________________________
$ns_ at $val(end_sim) "finish"
$ns_ at [expr $val(end_sim) + 0.01] "puts \"TERMINANDO SIMULACION...\" ; $ns_ halt"

puts "Comenzando simulación..."
$ns_ run
```

A.2. Métricas

THROUGHPUT

```
if ( <es tráfico TCP> && <pertenece a la capa de transporte> && <es un evento de tipo envío> && <el origen es correcto> && <el destino es correcto> ) {
          ++snt;
          pktSum += pkt_size
          diffTime = time - oldTime
}
if ((diffTime > 0.49) && (diffTime < 0.51) ) {
          if ((pktSum > 0.0) ){
                    th = (pktSum * 0.000008) / 0.5;
                    printf("%.4f %.4f\n", time, th ) > name;
          } else if (pktSum == 0.0) {
                    printf("%.4f %.4f\n", time, 0 ) > name;
          }
          pktSum = 0;
          oldTime = time;
          diffTime = 0;
}
```

GOODPUT

```
    set now [$ns_ now]
    set interval [expr $now - $oldTime]

    # Bytes recibidos por el sumidero TCP
    set good [$tcpDest set bytes_]

          # Calcuar el goodput (en MBit/s)
          if { $interval >= 0.49 && $interval < 0.51} {
                    if { $good > 0.0 } {
                              puts $outfileGOOD "$now [expr ($good*0.000008) / 0.5]"
                    } else {
                              puts $outfileGOOD "$now 0.0"
                    }
                    set oldTime $now
                    # Resetear bytes_ en sumidero
                    $tcpDest set bytes_ 0
                    set goodSum 0
          }
...
if ( <es tráfico TCP> && <pertenece a la capa de transporte> && <es un evento de tipo recibido> && <el origen es correcto> && <el destino es correcto> ) {
          ++rcv;
          pktSum += pkt_size
          diffTime = time - oldTime
}
if ((diffTime > 0.49) && (diffTime < 0.51) ) {
          if ((pktSum > 0.0) ){
                    gd = (pktSum * 0.000008) / 0.5;
                    printf("%.4f %.4f\n", time, gd ) > name;
          } else if (pktSum == 0.0) {
                    printf("%.4f %.4f\n", time, 0 ) > name;
          }
          pktSum = 0;
          oldTime = time;
          diffTime = 0;
}
```

TIEMPO TOTAL DE TRANSMISIÓN

```
if ( <es tráfico TCP> && <pertenece a la capa de transporte> && <es un evento de tipo enviado> && <el origen es correcto> && <el destino es correcto> ) {
```

```
{
          if (time < startTime) {
                    startTime = time
          }
          sendTime[pkt_id] = time
}
if ( <es tráfico TCP> && <pertenece a la capa de transporte> && <es un evento de tipo recibido> && <el origen es
correcto> && <el destino es correcto> ) {
          if (time > stopTime) {
                    stopTime = time
          }
          recvTime[pkt_id] = time
}
...
END {
          printf("Start time:  %.4f sec\n", startTime)
          printf("Stop time:  %.4f sec\n", stopTime)
          printf("Total transmission time: %.4f sec\n", (stopTime – startTime))
}
```

END TO END DELAY - LATENCIA

```
if (<es un nodo enviado correspondiente al origen, protocolo e id indicado>) {
          # Registrar tiempo para paquete enviado
          start_time[pkt_id] = time;
}
if ( <es un nodo recibido correspondiente al destino, protocolo e id indicado>) {
          # Registrar tiempo para paquete recibido
          start_time[pkt_id] = time;
          }
. . .
END {
    for ( pkt_id = 0; pkt_id <= max_packet_id; pkt_id++ ) {
                    if (!(pkt_id in start_time)) {
                              continue
                    }
                    start = start_time[pkt_id];
                    end = end_time[pkt_id];
                    packet_duration = end - start;

       if ( start < end ) {
                    printf("%.2f %.2f\n", start, packet_duration) > name;
          }
   }
}
```

VENTANA DE CONGESTIÓN (CWND)

```
     set now [$ns_ now]
     set cwnd [$tcpSource set cwnd_]
```

NUMERO DE SECUENCIA

```
     set now [$ns_ now]
     set t_seqno [$tcpSource set t_seqno_]
```

RTT

```
     set now [$ns_ now]
     set rtt [$tcpSource set rtt_]
```

Anexo B - Contribuciones

B.1. Artículos publicados

- Rodríguez Herlein D.R., Talay C.A., González C.N., Trinidad F.A., Almada M.L., Marrone L.A., **"***Contention Analysis of Congestion Control Mechanisms in a Wireless Access Scenario.***"** In: Pesado P., Aciti C. (eds) Computer Science – CACIC 2018, Communications in Computer and Information Science, vol. 995. Springer, Cham, June 2019, https://link.springer.com/chapter/10.1007/978-3-030-20787-8_18#citeas
- Rodríguez Herlein D.R., Talay C.A., González C.N., Trinidad F.A., Almada M.L., Marrone L.A., "*Burst Error Analysis Introduced in Multiple Traffic of Protocols TCP Reno, Cubic, Westwood and Vegas on a Model of Hybrid Topology*", Journal of Computer Science & Technology (JCS&T), Volume 19, Number 1, p. 32-44, ISSN: 1666-6038, April 2019. Available Online: http://journal.info.unlp.edu.ar/JCST/article/view/1210
- Rodríguez Herlein D.R., Talay C.A., González C.N., Marrone L.A., et al., *"Análisis de rendimiento y equidad en TCP",* XXI Workshop de Investigadores en Ciencias de la Computacion (WICC), Abril 2019, UNSJ, San Juan, Argentina, ISBN 978-987-3619-27-4**,** Available Online: http://sedici.unlp.edu.ar/handle/10915/76952
- F. A. Trinidad, D. Rodriguez Herlein, C Talay, L. Marrone et al., *"Congestion: falso positivo el caso del protocolo TCP en redes inalámbricas"*, Marcela Arpes, Ed., Encuentro en el extremo. Investigaciones científicas en la Patagonia Austral, 1a.Ed., ISBN | 978-987-3714-73-3, Rio Gallegos: Ediciones UNPAedita, 2019.
- D. Rodriguez Herlein, C Talay, C. Gonzalez, L. Marrone et al., *"Analysis of the performance of TCP Vegas and its relationship with alpha and beta parameters in a wireless links network and burst errors"*, II Congreso Argentino de Ciencias de la Informática y Desarrollos de Investigación (CACIDI), pp 1-6, noviembre 2018, Ciudad Autónoma de Buenos Aires, DOI 10.1109/CACIDI.2018.8584350. https://ieeexplore.ieee.org/document/8584350

- D. Rodriguez Herlein, C Talay, C. Gonzalez, L. Marrone et al., "*Un análisis de comportamiento entre distintos mecanismos de control de congestión ensayados sobre una topología mixta*", XXIV Congreso Argentino de Ciencias de la Computación (CACIC), Octubre 2018, Tandil, Buenos Aires, ISBN: 978-950-658-472-6, pág. 725 a 734, Available Online: http://sedici.unlp.edu.ar/handle/10915/73349
- D. Rodriguez Herlein, C Talay, C. Gonzalez, L. Marrone et al., *"Diseño, implementación y ejecución de simuladores de red con variantes del protocolo TCP en distintos entornos"*, V Encuentro de investigadores de la Patagonia Austral, octubre 2018, Rio Gallegos, Santa Cruz, Argentina. ISBN 978-987-3714-56-6
- D. Rodriguez Herlein, C Talay, C. Gonzalez, L. Marrone et al., *"Contienda entre las variantes del protocolo TCP Vegas y Reno por los recursos de la red en un modelo hibrido simple"*, XX Workshop de Investigadores en Ciencias de la Computación (WICC), abril 2018, UNNE, Corrientes, Corrientes, Argentina, ISBN 978-987-3619-27-4, pag. 159 a 163, Available Online: http://sedici.unlp.edu.ar/handle/10915/67063
- F. A. Trinidad, D. Rodriguez Herlein, C Talay y C. Gonzalez, *"Automatización del proceso de simulación para el análisis de rendimiento del protocolo TCP en redes inalámbricas",* ICT-UNPA-186-2018, febrero 2018, ISSN: 1852-4516, Aprobado por Resolución N° 1453/18-R-UNPA
- D. Rodriguez Herlein, C Talay, C. Gonzalez, L. Marrone et al., *"Desempeño de TCP Vegas según parámetros alfa y beta en escenarios híbridos con errores en ráfagas",* 2nd International Conference on Information Systems and Computer Science (INCISCOS), Noviembre 2017, Quito, Ecuador, IEEE Conferences, Pages: 217 – 223, https://ieeexplore.ieee.org/document/8328110/
- D. Rodriguez Herlein, C Talay, C. Gonzalez, L. Marrone et al., *"Consideraciones sobre el comportamiento del protocolo TCP en sus variantes Vegas, Reno, Cubic y Westwood ante errores en ráfaga en una topología híbrida"*, XXIII Congreso Argentino de Ciencias de la Computación (CACIC), Octubre 2017, La Plata, Buenos Aires, ISBN: 978-950-34-1539-9 pág. 874 a 883, Available Online: http://sedici.unlp.edu.ar/handle/10915/63846
- D. Rodriguez Herlein, C Talay, C. Gonzalez, L. Marrone et al., *"Un caso práctico de utilización del simulador ns-2 en el análisis del comportamiento de variantes del protocolo TCP con fines de investigación y enseñanza"* , XII Congreso de Tecnología en Educación y Educación en Tecnología (TE&ET), Junio 2017, La Matanza, Buenos Aires, ISBN: 978-987-4417-04-6, pág. 77-93, Available Online: http://sedici.unlp.edu.ar/handle/10915/63370
- D. Rodriguez Herlein, C Talay, C. Gonzalez, L. Marrone et al., *"Estudio del protocolo TCP en la utilización de redes con conexión Wireless"*, IV Encuentro de investigadores

de la Patagonia Austral, Octubre 2016, Caleta Olivia, Santa Cruz, Argentina. ISBN 978-987-3714-37-5

- D. Rodriguez Herlein, C Talay, C. Gonzalez, L. Marrone et al., *"Explorando posibles mejoras de protocolo TCP en redes móviles"*, XVIII Workshop de Investigadores en Ciencias de la Computación (WICC), Abril 2016, Concordia, Entre Ríos, Argentina, ISBN: 978-950-698-377-2 , pág. 177 a 181, Available Online: http://sedici.unlp.edu.ar/handle/10915/52801

B.2. Actividades de extensión

- XV Jornadas de Informática, *"Herramientas de representación gráficas para el análisis de datos en simulación de redes",* UARG-UNPA, Septiembre 2018.
- XV Jornadas de Informática, *"Importancia de TCP en telecomunicaciones"*, UARG-UNPA, septiembre 2018.
- XVI Semana nacional de la ciencia y la tecnología, *"Principales herramientas disponibles para la instalación y análisis del tráfico en redes de computadoras",* Colegio Industrial N°4 Rio Gallegos, septiembre 2018.
- I Jornadas Binacionales de Investigadores, *"Congestión falso positivo: el protocolo TCP en redes inalámbricas",* UARG-UNPA, noviembre 2017.
- XIV Jornadas de Informática, *"La utilización del ns2 para el análisis de parámetros de dos tipos de topologías comparadas con distintas variantes de protocolo Reno vs. Vegas",* UARG-UNPA, septiembre 2017.
- XIV Jornadas de Informática, *"Una revisión de las versiones de protocolo TCP y los ámbitos en los cuales se aplican",* UARG-UNPA, septiembre 2017.
- XIII Jornadas de Informática, *"El protocolo TCP y su utilización en conexiones wireless",* UARG-UNPA, octubre 2016.

Bibliografía

[1] J. Postel, *Transmission Control Protocol RFC793,* 1981.

[2] «IETF, Internet Engineering Task Force,» [En línea]. Available: http://www.ietf.org. [Último acceso: Marzo 2019].

[3] Paxson, V. , Allman, M., Chu, J. and Sargent, M. , *Computing TCP's Retransmission Timer, RFC6298,* 2011.

[4] Guha, S., ed., K. Biswas, Ford, B., Sivakumar, S. and Srisuresh, P. , *NAT Behavioral Requirements for TCP, RFC 5382,* 2008.

[5] Eggert, L. and Gont, F. , *TCP User Timeout Option, RFC 5482,* 2009.

[6] Gont, F. and Yourtchenko, A., *On the Implementation of the TCP Urgent Mechanism, RFC6093,* 2011.

[7] Kurose, J.F. and Ross, K.W., COMPUTER NETWORKING: A Top-Down Approach (6th. Edition), Boston: Pearson, 2012.

[8] D. E. Comer, Internetworking with TCP/IP. Vol. 1, Boston: Pearson, 2014.

[9] Fall, Kevin R., and W. Richard Stevens, TCP/IP illustrated, Volumen 1 (2nd. Edition), Upper Saddle River, NJ: Addison-Wesley, 2012.

[10] Jacobson. V., Braden. R. and Borman, D., *TCP Extensions for High Performance - RFC1323,* 1992.

[11] W. Stallings, Data and computer communications, (8th. Edition), Upper Saddle River, N.J: Pearson/Prentice Hall, 2007.

[12] Welzl, M., Network congestion control: managing Internet traffic, Chichester, West Sussex, England: Hoboken, NJ: J. Wiley, 2005.

[13] Stevens, W., *TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms - RFC 2001,* 1997.

[14] Allman M., Paxson V. and Blanton E., *TCP Congestion Control - RFC 5681,* 2009.

[15] Khelage,P.B. and Kolekar, U., «Survey and Simulation based Performance Analysis of TCP-Variants in terms of Throughput, Delay and drop Packets over MANETs,» *International Journal of Scientific & Engineering Research,* vol. 5, nº 1, January 2014.

[16] Ashraf, T., Noor-ul-Sabah and Arshad, M.J., «Comparative Study of TCP Protocols: A Survey,» *International Journal of Computer Science and Telecommunications,* vol. 8, nº 1, pp. 19 - 25, 2017.

[17] Esterhuizen, A. and Krzesinski, A.E., «TCP Congestion Control Comparison,» *SATNAC,* September 2012.

[18] Kho, L.C.; Defago, X.; Lim, A. O. and Tan, T., «A taxonomy of congestion control techniques for TCP in wired and wireless networks,» *IEEE Symposium on Wireless Technology and Applications (ISWTA),* pp. 147-152, 2013.

[19] Jacobson, V., «Congestion Avoidance and Control,» *ACM SIGCOMM Computer Communication Review,* vol. 25, nº 1, pp. 157 - 187, 1995.

[20] S. Lar and X. Liao, «An initiative for a classified bibliography on TCP / IP congestion control,» *Journal of Network and Computer Applications,,* vol. vol. 36, p. pp. 126–133, 2012..

[21] Polese, M. et al., «A Survey on Recent Advances in Transport Layer Protocols,» *IEEE Communications Surveys & Tutorials,* 2019.

[22] Ahmad, U., Ngadi, A. I, et al., «Survey on end to end congestion control techniques in different network scenarios,» *Journal of Theoretical and Applied Information Technology,* vol. 77, nº 3, pp. 342 - 364, 2015.

[23] Afanasyev, A., Tilley, N., Reiher, P. and Kleinrock, L., «Host-to-Host Congestion Control for TCP,» *IEEE Communications Surveys & Tutorials,* vol. 12, nº 3, pp. 304-342, 2010.

[24] Molia, H. K., & Agrawal, R., «A conceptual exploration of TCP variants,» *2nd International Conference on "Emerging Technology (ET2ECN),* 2015.

[25] Olasoji, B., Oyenike M., Olanrewaju, O., Adebayo I., «Transmission Control Protocol and Congestion Control: A Review of TCP Variants,» *International Journal of Computer Science and Information Security (IJCSIS),* vol. 14, nº 3, p. 127, 2016.

[26] Chaudhary, P. and Kumar, S., «A Review of Comparative Analysis of TCP Variants for Congestion Control in Network,» *International Journal of Computer Applications,* vol. 160, nº 8, 2017.

[27] Singh and Meenu, , «A survey on congestion control mechanisms in packet switch networks,» de *International Conference on Advances in Computer Engineering and Applications*, Ghaziabad, 2015.

[28] V. Jacobson, «Congestion avoidance and control,» *ACM SIGCOMM,* p. pp. 314–329, 1988.

[29] V. Jacobson, «Modified TCP congestion avoidance algorithm,» *email to the end2end list,* April 1990.

[30] Allman, M., Paxson, V. and Stevens, W., «RFC2581 - TCP congestion control,» 1999.

[31] Floyd, S. and Henderson, T., *RFC2582 - the NewReno modification to TCP's fast recovery algorithm,* 1999.

[32] Floyd, S. , Henderson,T. and Gurtov, A., *RFC3782 - the NewReno modification to TCP's fast recovery algorithm,* 2004.

[33] Mathis, M., Mahdavi, J., Floyd, S., and Romanov, A., *RFC2018 - TCP selective acknowledgment options,* 1996.

[34] Mathis, M. and Mahdavi, J., «Forward acknowledgement: refining TCP congestion control,» de *Proc. conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, New York, NY, USA,, 1996.

[35] Brakmo, L. and Peterson, L., «TCP Vegas: end to end congestion avoidance on a global Internet,» *IEEE J. Sel. Areas Commun,*

vol. 13, nº 8, p. 1465–1480, 1995.

[36] Kaur, H. and Singh, G., «TCP Congestion Control and Its Variants,» *Advances in Computational Sciences and Technology,* vol. 10, nº 6, pp. 1715-1723, 2017.

[37] Ghassan A. A., Mahamod I. & Kasmiran J., «Influence of Parameters Variation of TCP-Vegas in Performance of Congestion Window over Large Bandwidth-Delay Networks,» de *17th Asia-Pacific Conference on Communications (APCC)*, Sabah, Malaysia, 2011.

[38] Rodgers, A. and Kuznetsova, T., «TCP Reno and Vegas co-existence,» *Semantic Scholars,* 2015.

[39] Tsang, E.C.M. and Chang, R.K.C., «A Simulation Study on the Throughput Fairness of TCP Vegas,» de *9th IEEE International Conference on Networks*, Bangkok, Thailand,, 2001.

[40] Hasegawa, G., Kurata, K., and Murata, M., «Analysis and improvement of fairness between TCP Reno and Vegas for deployment of TCP Vegas to the Internet,» *Proc. IEEE ICNP,* p. 177–186., 2000.

[41] Srijith, K., Jacob, L. and Ananda, A., «TCP Vegas-A: Improving the performance of TCP Vegas,» *Computer Communications,* vol. 28, nº 4, p. 429–440, 2005.

[42] Luo, Y., Yin, M., Jiang, H. and Ma, S., «An improved congestion avoidance control model for TCP Vegas based on Ad Hoc networks,» de *26th Chinese Control and Decision Conference (CCDC)*, Changsha, 2014.

[43] Fu, C. P. and Liew, S. C. , «TCP Veno: TCP enhancement for transmission over wireless access networks,» *IEEE J. Sel. Areas Commun,* vol. 21, nº 2, 2003.

[44] R. Braden, *RFC1122 - Requirements for Internet Hosts - Communication Layers,* 1989..

[45] V. Paxson, «End-to-end Internet packet dynamics,» *SIGCOMM Computer Communication Review,* vol. 27, nº 4, p. 139–152, 1997.

[46] Ludwig, R. and Gurtov, A., *The Eifel Response Algorithm for TCP - RFC4015,* 2005.

[47] Wang, F. and Zhang, Y., «Improving TCP performance over mobile adhoc networks with out-of-order detection and response,,» de *Proceedings of the 3rd ACM international symposium on mobile ad hoc networking & computing*, New York, NY, 2002.

[48] Bohacek S, Hespanha J, Lee J, Lim C, and Obraczka K.,, «TCP PR: TCP for persistent packet reordering,» *International Conference on Distributed Computing Systems,* vol. 23, p. 222–233, 2003..

[49] Zhang, M. , Karp, B. , Floyd, S. and L. Peterson, «RR-TCP: a reordering robust TCP with DSACK,» *International Computer Science Institute, Tech. Rep.,* 2002.

[50] Venkataramani, A., Kokku, R. and Dahlin, M., «TCP Nice: A Mechanism for Background Transfers,» *Operating Systems Review,* nº 36, p. 329–344, 2002.

[51] Kuzmanovic, A. and Knightly, E. W., «TCP-LP: a distributed algorith for low priority data transfer,» de *IEEE INFOCOM*, 2003.

[52] Hrituparna, P. and Priyanka, S., «A Survey: High Speed TCP Variants in Wireless Networks,» *International Journal of Advance Research in Computer Science and Management Studies,* vol. 1, nº 7, pp. 142-148, 2013.

[53] Alrshah, M.A., Othman, M., Alib, B. and Hanapia, Z.M., «Comparative study of High-speed Linux TCP Variants over High-BDP Networks,» *Journal of network and computer applications,* pp. 43, 66 - 75, 2014.

[54] S. Floyd, *HighSpeed TCP for large congestion windows - RFC3649,* 2003.

[55] S. Floyd, *Limited slow-start for TCP with large congestion windows - RFC3742,* 2004.

[56] T. Kelly, «Scalable TCP: improving performance in highspeed wide area networks,» *Computer Communications Review,* vol. 32, nº 2, 2003.

[57] D. Leith, «H-TCP: TCP congestion control for high bandwidth-delay product paths,» IETF Internet Draft, 7th. April 2008. [En línea]. Available: https://tools.ietf.org/html/draft-leith-tcp-htcp-06. [Último acceso: Marzo 2019].

[58] Caini, C. and Firrincieli, R., «TCP Hybla: a TCP enhancement for heterogeneous networks,» *International J. Satellite Communications and Networking,* vol. 22, p. 547–566, 2004.

[59] Xu, L., Harfoush, K. and Rhee, I. , «Binary increase congestion control for fast, long distance networks,» *IEEE INFOCOM,* vol. 4, p. 2514–2524, 2004.

[60] Rhee, I. and Xu, L., «CUBIC: a new TCP-friendly high-speed TCP variant,» *SIGOPS Operating Systems Review,* vol. 42, nº 5, p. 64–74, 2008.

[61] Wei, D.X., Jin, C., Low, S.H. and Hegde, S., «FAST TCP: motivation, architecture, algorithms, performance,» *IEEE/ACM Trans. Netw,* vol. 14, nº 6, p. 1246–1259, 2006.

[62] Marfia, G., Palazzi, C., Pau, G., Gerla, M., Sanadidi, M. and Roccetti, M., «TCP Libra: Exploring RTT-Fairness for TCP,» *UCLA Computer Science Department, Tech. Rep. ,* 2005.

[63] Sing, J. and Soh, B., «TCP New Vegas: Improving the Performance of TCP Vegas Over High Latency Links,» *4th IEEE International Symposium on Network Computing and Applications (IEEE NCA05),* p. 73–80, 2005.

[64] Shimonishi, H. and Murase, T., «Improving efficiency-friendliness tradeoffs of TCP congestion control algorithm,» *IEEE GLOBECOM,,* 2005.

[65] Kaneko, K., Fujikawa, T., Su, Z., and Katto, J., «TCP-Fusion: a hybrid congestion control algorithm for high-speed networks,» de *PFLDnet, ISI*, Los Angeles, California, 2007.

[66] King, R., Baraniuk, R, and Riedi, R., «TCP-Africa: an adaptive and fair rapid increase rule for scalable TCP,» *IEEE INFOCOM,* vol. 3, p. 1838–1848, 2005.

[67] Tan, K., Song, J., Zhang, Q. and M. Sridharan, , «A compound TCP approach for high-speed and long distance networks,» *IEEE INFOCOM,* 2005.

[68] Liu, S., Basar, T. and Srikant, R., «TCP-Illinois: A loss and delay-based congestion control algorithm for high-speed networks,» *First International Conference on Performance Evaluation Methodologies and Tools ,* 2006.

[69] Baiocchi, A., Castellani, A.P.and Vacirca, F., «YeAH-TCP: yet another highspeed TCP,» de *PFLDnet, ISI,*, Los Angeles, California, 2007.

[70] Marrone, L., Barbieri, A. and Robles, M., Tecnologías Wireless y Movilidaden IPv4/IPv6, La Plata, Bs. As.: Editorial de la Universidad Nacional de La Plata (Edulp), 2011.

[71] Qiu, R.C., Hu, Z., Li, H. and Wicks, M.C., Cognitive Radio Communication and Networking: Principles and Practice, John Wiley & Sons, 2012.

[72] Agrawal, D.P. and Zeng, Qing-An , Introduction to Wireless and Mobile Systems (3rd. Edition), Stamford, CT: Cengage Learning, 2011.

[73] Saidul Huq, K.M. and Rodriguez, J., Future Wireless Systems, Chichester, UK: John Wiley & Sons, Ltd., 2017.

[74] Ho L., «An investigation of improving the traditional TCP over MANETs,» de *Thesis Master of Computer and Information Sciences*, Aukland University of Tecnology, 2012.

[75] Dalal. K., Chaudhary, P. and Dahiya. P., «Performance Evaluation of TCP and UDP Protocols in VANET Scenarios using NCTUns-6.0 Simulation Tool,» *International Journal of Computer Applications ,* vol. 36, nº 6, pp. 6 - 9, 2011.

[76] Shahdad, S.Y., Sabahath, A. and Parveez, R., «Architecture, Issues and Challenges of Wireless Mesh Network,» de *International Conference on Communication and Signal Processing*, India, 2016.

[77] Miriam Carlos-Mancilla, Ernesto López-Mellado, and Mario Siller, «Wireless Sensor Networks Formation: Approaches and Techniques,» de *Journal of Sensors*, Hindawi Publishing Corporation, 2016.

[78] Bellalta, B., «IEEE 802.11ax: High-efficiency WLANS,» *IEEE Wireless Communications,* vol. 23, nº 1, pp. 38 - 46, 2015.

[79] «Official IEEE Standards Association Website,» [En línea]. Available: https://standards.ieee.org/standard/. [Último acceso: Septiembre 2019].

[80] «Wi-Fi Alliance Official industry association web site,» [En línea]. Available: http://www.wi-fi.org/. [Último acceso: Septiembre 2019].

[81] Rodriguez, F., «Análisis del desempeño de redes 802.11,» *", Universidad de la República (Uruguay). Facultad de Ingeniería,* Nov 2015.

[82] Ali Abedi,, «Evaluating and Characterizing the Performance of 802.11 Networks,» *UWSpace,* 2017.

[83] Pokhrel, S.R., «Modeling and Performance Evaluation of TCP over Last-mile Wireless Networks,» *PhD Thesis, Swinburne Universty of Technology,* 2017.

[84] Leung K.-C. & Li V. O. K., «Transmission Control Protocol (TCP) in Wireless Networks: Issues, Aproaches, and Challenges,» *IEEE Comunications Surveys and Tutorials,* vol. 8, nº 2, pp. 64-79, 2006.

[85] Molia, H.K. and Kothari, A.D., «TCP Variants for Mobile Adhoc Networks: Challenges and Solutions,» *Wirel. Pers. Commun,* vol. 100, nº 4, p. 1791–1836, Jun. 2018.

[86] Kanellopoulos, D., «Congestion control for MANETs: An overview,» *ICTExpress,* 2018.

[87] Hala Elaarag, «Improving TCP Performance over Mobile Networks,» *ACM Computing Surveys,* vol. 34, nº 3, p. 357–374, September 2002.

[88] Teja F.R., Vidal, L and Alves, L., «TCP sobre enlaces wireless – Problemas y algunas posibles soluciones existentes,» *Curso de posgrado y actualización, Instituto de Ingeniería Eléctrica, Facultad de la República,* marzo 2004.

[89] Ramakrishnan, K., Floyd, S. and Black, D., *The addition of explicit congestion notification (ECN) - RFC3168,* 2001.

[90] Bagal, P., Kalyanaraman, S., Packer, B. and Ny, T., «Comparative study of RED, ECN and TCP Rate Control,» *Dept of ECSE, RPI,* pp. 1 - 27, 1999.

[91] Fairhurst G. and Wood L., *Advice to link designers on link Automatic Repeat reQuest (ARQ), RFC3366,* 2002.

[92] Grazia, C.A., Patriciello, N., Klapez, M. and Casoni, M. , «A cross comparison between TCP and AQM algorithms: Which is the best couple for congestion control?,» *14th International Conference on Telecommunications (ConTEL),* p. 75–82, June 2017.

[93] Bakre, A. and Badrinath, B., «I-TCP: Indirect TCP for Mobile Hosts,» *Department of Computer Science, Rutgers University, Tech. Rep. DCSTR-314,* 1994.

[94] Brown, K. and Singh, S. , «M-TCP: TCP for mobile cellular networks,» *SIGCOMM Computer Communication Review,* vol. 27, nº 5, pp. 19 - 43, 1997.

[95] Kim, B.H., «Techniques for End-to-End Tcp Performance Enhancement Over Wireless Networks,» *Publicly Accessible Penn Dissertations,* 2016.

[96] Mascolo, S., Casetti, C., Gerla, M., Sanadidi, M.Y. and Wang, R., «TCP Westwood: Bandwidth estimation for enhanced transport over wireless links,» *ACM MOBICOM,* p. 287–297, 2001.

[97] Kliazovich, D., Granelli, F. and Miorandi, D. , «Logarithmic window increase for TCP Westwood+ for improvement in high speed, long distance networks,» *Computer Networks,* Vols. %1 de %252,, nº 12, p. 2395– 2410, 2008.

[98] Grieco, L.A. and Mascolo, S. , «Performance evaluation and comparison of Westwood+, New Reno and Vegas TCP congestion control,» *ACM Computer Communication Review,* vol. 342, 2004.

[99] Wang, R., Valla, M., Sanadidi, M., Ng, B. and Gerla, M. , «Efficiency/friendliness tradeoffs in TCP Westwood,» *Seventh International Symposium on Computers and Communications,* p. 304–311, 2002.

[100] Wang, R. Valla, M. Sanadidi, M. and Gerla, M. , «Adaptive bandwidth share estimation in TCP Westwood,» *IEEE GLOBECOM,* vol. 3, p. 2604–2608, 2002.

[101] Yang, G., Wang, R., Sanadidi, M. and Gerla, M. , «TCPW with bulk repeat in next generation wireless networks,» *IEEE International Conference on Communications,* vol. 1, p. 674–678, 2003.

[102] Shimonishi, H., Sanadidi, M. and Gerla, M. , «Improving efficiencyfriendliness tradeoffs of TCP in wired-wireless combined networks,» *IEEE ICC ,* vol. 5, p. 3548–3552, 2005.

[103] Wang, R., Yamada, K., Sanadidi, M. and Gerla, M., «TCP with senderside intelligence to handle dynamic, large, leaky pipes,» *IEEE J. Sel. Areas Commun,* vol. 23, nº 2, p. 235–248, 2005.

[104] García Dávalos, A., Escobar Paz,L. M. et.al., «Método de evaluación y selección de herramientas de simulación de redes,» *Universidad Icesi, Revista Sistemas y Telemática,* vol. 9, nº 16, pp. 55-71, 2011.

[105] Issariyakul, T., Hossain, E., Introduction to Network Simulator NS2, Boston, MA: Springer US, 2012.

[106] «NS3 Documentation: A Discrete-Event Network Simulator,» [En línea]. Available: http://www.nsnam.org/documentation. [Último acceso: Marzo 2019].

[107] «EstiNet,» [En línea]. Available: https://www.estinet.com/ns/. [Último acceso: Marzo 2019].

[108] «OMNET++ Documentation: Discrete Event Simulator,» [En línea]. Available: http://www.omnetpp.org/documentation. [Último acceso: Marzo 2019].

[109] «Start with J-Sim,» [En línea]. Available: https://sites.google.com/site/jsimofficial/start-with-j-sim. [Último acceso: Marzo 2019].

[110] «Documentation - The official guide and reference for GNS3,» [En línea]. Available: https://docs.gns3.com/. [Último acceso: Marzo 2019].

[111] «NetSim Knowledge base,» [En línea]. Available: https://support.tetcos.com/support/home. [Último acceso: Marzo 2019].

[112] Calle, M.A., Tovar, J.D., Castaño Pino, Y.J., and Cuéllar, J.C., «Comparación de Parámetros para una Selección Apropiada de Herramientas de Simulación de Redes,» *Información Tecnológica,* vol. 29, nº 6, 2018.

[113] «GAWK: Effective AWK Programming: A User's Guide for GNU AwK,» The Free Software Foundation (FSF), Octubre 2014. [En línea]. Available: gnu.org/software/gawk/manual/. [Último acceso: Septiembre 2016].

[114] «XGRAPH - General Purpose 2-D Plotter,» [En línea]. Available: http://www.xgraph.org/. [Último acceso: Septiembre 2016].

[115] Martínez Bonastre, O. and Palau Salvador, C., Introducción a la programación de protocolos de comunicaciones con Network

Simulator 2, (ECU) EDITORIAL CLUB UNIVERSITARIO, 2000.

[116] «The ns Manual (formerly ns Notes and Documentation),» The VINT Project, 5 Nov. 2011. [En línea]. Available: https://www.isi.edu/nsnam/ns/doc/. [Último acceso: Septiembre 2016].

[117] «The Perl.org free online Perl books,» Perl.org, [En línea]. Available: https://www.perl.org/books/library.html. [Último acceso: Agosto 2016].

[118] Islam, Md Shohidul, «Performance Measurement and Improvement of TCP: An In-Depth Performance Analysis of TCP Over Wired and Wireless Network Using NS-2,» *Saarbrücken: VDM Verlag Dr. Müller,* 2010.

[119] Trinidad, F., Rodriguez Herlein, D., Talay, C. et al., «Automatización del proceso de simulación para el análisis de rendimiento del protocolo TCP en redes inalámbricas,» *ICT-UNPA-186-2018, ISSN: 1852-4516, Resolución N° 1453/18-R-UNPA,* 2018.

[120] Aad, Imad, Mohammad Hossein Manshaei and JeanPierre Hubaux, ns-2 for the impatient, Lausanne, Switzerland: EPFL, 2009.

[121] Chung, J., Claypool, M. et al., «NS by Example,» 2016. [En línea]. Available: http://nile.wpi.edu/NS. [Último acceso: Abril 2017].

[122] «UIT-T SG 12,» [En línea]. Available: https://www.itu.int/md/T17-SG12-190507/sum/es. [Último acceso: Mayo 2019].

[123] «(ITU), International Telecommunication Union,» [En línea]. Available: http://www.itu.int. [Último acceso: Marzo 2019].

[124] «IETF, IPPM WG,» [En línea]. Available: https://datatracker.ietf.org/wg/ippm/about/. [Último acceso: Mayo 2019].

[125] «ITU-T - Publications - Recommendations - Y Series : Y.1540,» Diciembre 2002. [En línea]. Available: https://www.itu.int/rec/T-REC-Y.1540/recommendation.asp?lang=es&parent=T-REC-Y.1540-201912-I. [Último acceso: junio 2019].

[126] « ITU-T - Publications - Recommendations - G Series : G.1020,» Julio 2006. [En línea]. Available: https://www.itu.int/rec/T-REC-G.1020/recommendation.asp?lang=en&parent=T-REC-G.1020-200607-I. [Último acceso: Junio 2019].

[127] Paxson, V., Almes, G., Mahdavi, J. and Mathis, M. , *Framework for IP Performance Metrics - RFC2330,* 1998.

[128] Almes, G., Kalidindi, S. and Zekauskas, M. , *A One-way Packet Loss Metric for IPPM - RFC2680,* 1999.

[129] Almes, G., Kalidindi, S., Zekauskas, M., and Morton A. (Ed.), , *A One-Way Loss Metric for IP Performance Metrics (IPPM) - RFC7680,* 2016.

[130] Almes, G., Kalidindi, S. and Zekauskas, M., *A One-way Delay Metric for IPPM - RFC2679,* 1999.

[131] Almes, G., Kalidindi, S. and Zekauskas, M., *A Round-trip Delay Metric for IPPM - RFC2681,* 1999.

[132] Demichelis, C. and Chimento, P. , *IP Packet Delay Variation Metric for IP Performance Metrics (IPPM) - RFC3393,* 2002.

[133] Paxson, V. and Mahdavi, J. , *IPPM Metrics for Measuring Connectivity - RFC2678,* 1999.

[134] Koodli, R. and Ravikanth, R., *One-way Loss Pattern Sample Metrics - RFC3357,* 2002.

[135] Morton, A., Ciavattone, L., Ramachandran, G., Shalunov, S. and Perser, J. , *Packet Reordering Metrics - RFC4737,* 2006.

[136] Chimento, P. and Ishac, J., *Defining Network Capacity - RFC5136,* 2008.

[137] Mathis, M. and Allman, M. , *A Framework for Defining Empirical Bulk Transfer Capacity Metrics - RFC3148,* 2001.

[138] Mathis, M., «Model based metrics for bulk transport capacity,» *IETF IP Performance Working Group Internet-Draft,* February 2017.

[139] Sawarkar, A. and Saraswat, H., «Performance Analysis of TCP Variants,» *International Journal of Computer Science and Network Security,* vol. 16, pp. 102-106, 2016.

[140] Harjinder Kaur et al., «Measuring performance of variants of TCP congestion control protocols,» *Indian Journal of Computer Science and Engineering (IJCSE),* vol. 8, nº 3, Jul 2017..

[141] Fabini, J. and Morton, A., *Advanced Stream and Sampling Framework for IP Performance Metrics (IPPM) - RFC7312,* 2014.

[142] Constantine, B., Forget, G., Geib, R. and Schrage, R., *Framework for TCP Throughput Testing - RFC6349,* 2011.

[143] Mathis, M. and Heffner, J. , *Packetization Layer Path MTU Discovery, RFC4821,* 2007.

[144] Bradner, S. and McQuaid, J., *Benchmarking Methodology for Network Interconnect Devices - RFC2544,* 1999.

[145] Rodriguez Herlein, D., Talay, C. , Marrone, L. et al., «Consideraciones sobre el comportamiento del protocolo TCP en sus variantes Vegas, Reno, Cubic y Westwood ante errores en ráfaga en una topología híbrida,» de *XXIII Congreso Argentino de Ciencias de la Computación (CACIC)*, La Plata, Buenos Aires, Octubre 2017.

[146] Rodriguez Herlein, D., Talay, C., Marrone, L. et al., «Desempeño de TCP Vegas según parámetros alfa y beta en escenarios híbridos con errores en ráfagas,» de *2nd International Conference on Information Systems and Computer Science (INCISCOS)*, Quito, Ecuador, Noviembre 2017.

[147] Rodriguez Herlein, D., Talay, C., Marrone, L. et al., «Analysis of the performance of TCP Vegas and its relationship with alpha and beta parameters in a wireless links network and burst errors,» de *II Congreso Argentino de Ciencias de la Informática y Desarrollos de Investigación (CACIDI)*, Ciudad Autónoma de Buenos Aires, Noviembre 2018.

[148] Rodriguez Herlein, D., Talay, C., Marrone, L. et al., «Un análisis de comportamiento entre distintos mecanismos de control de congestión ensayados sobre una topología mixta,» de *XXIV Congreso Argentino de Ciencias de la Computación (CACIC)*, Tandil, Buenos Aires, Octubre 2018.

[149] Rodriguez Herlein, D., Talay, C., Marrone, L. et al., «Contention Analysis of Congestion Control Mechanisms in a Wireless Access Scenario,» de *Computer Science – CACIC 2018, Communications in Computer and Information Science, vol. 995*, Springer, June 2019.

[150] Rodriguez Herlein, D., Talay, C., Marrone, L. et al., «Burst Error Analysis Introduced in Multiple Traffic of Protocols TCP Reno, Cubic, Westwood and Vegas on a Model of Hybrid Topology,» *Journal of Computer Science & Technology (JCS&T),* vol. 19, nº 1, pp. 32 - 44, April 2019.