Universidad Central "Marta Abreu" de Las Villas

Facultad de Ingeniería Eléctrica

Dpto. Telecomunicaciones y Electrónica

TRABAJO DE DIPLOMA

Análisis de los Mecanismos de Control de la Congestión en Redes IP.

Autor: Justo Harvey Ordoñez Molina.

Tutor: Dr. Ing. Félix Álvarez Paliza.

Santa Clara

2008

Año del 50 de la Revolución

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Telecomunicaciones y Electrónica, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

Firma del Autor

Firma del Jefe de Departamento
donde se defiende el trabajo

Firma del Responsable de
Información Científico-Técnica

Pensamiento

La ciencia está en conocer las oportunidades y aprovecharlas.

José Martí

Dedicatoria

- A mis padres: Por ser luz y guía en el sendero de mi vida.
- A mi tía Ana Delia: Por darme aliento para seguir adelante.
- A la memoria de mi Abuelo: Por sembrar en mí este sueño que hoy se hace realidad.

Agradecimientos

- A mis padres por ser el puntal de mi existencia y por todos sus esfuerzos para que pudiera alcanzar la meta.
- A mi abuela por ser para mí como una madre y por todos sus sabios consejos.
- A mi novia por despertar mis sueños y llenarme de esperanzas.
- A Asdrubal por su preocupación y apoyo en todos estos años.
- A mis suegros por brindarme tanto cariño y darme un lugar en su corazón.
- A mi tutor por todas sus enseñanzas.
- A todos mis amigos, que durante estos cinco años han estado a mi lado de forma incondicional.
- A todos aquellos que de una forma u otra han puesto su granito de arena en la realización de este trabajo.

A TODOS MUCHAS GRACIAS

TAREA TÉCNICA

1. Revisión bibliográfica para la construcción del marco teórico de referencia general.
2. Describir los diferentes mecanismos para el control de flujo TCP y el control de la congestión en redes IP.
3. Realizar la modelación y simulación de Redes IP con la herramienta OPNET Modeler.
4. Análisis comparativo de las implementaciones TCP.

Firma del Autor	Firma del Tutor

RESUMEN

Los nuevos servicios y aplicaciones surgidas durante los primeros años del siglo XXI en Internet plantean serios problemas para los mecanismos de control de la congestión, gestión del ancho de banda y otros recursos de la red. En el presente trabajo se analizan las versiones TCP más usadas en las redes IP, con el fin de evaluarlos y analizar sus incidencias sobre el funcionamiento de las mismas, y los compromisos de la QoS (Calidad de servicio) negociadas para el flujo de datos ya establecidos. Se realizan simulaciones de un modelo de red con la herramienta OPNET Modeler, demostrando que la implementación TCP SACK (Selective Acknowledgment, Reconocimientos Selectivos) es la más efectiva, pues expresa un mayor rendimiento en las redes IP.

ÍNDICE

INTRODUCCIÓN……...…….……..1

CAPÍTULO 1. MECANISMOS DE CONTROL DE LA CONGESTIÓN……....3

1.1 LOS PROTOCOLOS DE TRANSPORTE DE INTERNET: TCP Y UDP…….…..4

1.1.1 UDP (User Data Protocol)....…...……...…….……...….…..4

1.1.2 TCP (Transmission Control Protocol)…….....5

1.2 CONTROL DE FLUJO……..9

1.2.1. Mecanismo de Ventana Deslizante……10

1.2.2 Problemas que afectan el desempeño de la red. Soluciones……..10

1.2.3 Estrategias de Retransmisión…….....11

1.3 CONTROL DE LA CONGESTIÓN……..13

CAPÍTULO 2. IMPLEMENTACIONES TCP Y EL CONTROL DE LA

CONGESTIÓN……..……...14

2.1 MECANISMOS PARA EL CONTROL DE LA CONGESTIÓN…….…….15

2.1.1 Algoritmos de Inicio Lento (Show Start) y Prevención de la Congestión (Congestion Avoidance)…….....15

2.1.2 Algoritmos de Retransmisión Rápida (Fast Retransmit) y Recuperación Rápida (Fast Recovery)……17

2.2 PRINCIPALES IMPLEMENTACIONES TCP……..…….....19

2.2.1 TCP TAHOE.....20

2.2.2 TCP RENO 21

2.2.3 TCP NEW-RENO…….……21

2.2.4 TCP SACK(Selective Acknowledgment, Reconocimientos Selectivos)….....23

2.2.5 TCP VEGAS……...…….…….....24

2.2.6 OTRAS VERSIONES TCP 25

2.3 SIMULADOR OPNET MODELER…….....27

CAPÍTULO 3. SIMULACIÓN Y ANÁLISIS DE LOS RESULTADOS……..…..…29

3.1 DISEÑO DE LOS ESCENARIOS……..……...…..29

3.2 ANÁLISIS DE LOS RESULTADOS…….…….....…..32

3.2.1 Análisis de TCP Tahoe…….…….……..32

3.2.2 Análisis de TCP Reno…….…….....34

3.2.3 Análisis de TCP SACK……..….36

3.2.4 Otros Resultados…….……37

CONCLUSIONES……41

RECOMENDACIONES.....42

GLOSARIO 43

REFERENCIAS BIBLIOGRÁFICAS 45

ANEXOS 48

INTRODUCCIÓN

La tendencia actual del crecimiento de las redes de alta velocidades y su interconexión a fin de soportar nuevas y variadas aplicaciones, incrementan el tráfico y la congestión en las mismas, producto a la incorporación de numerosos sistemas o usuarios finales que solicitan dichas aplicaciones.

Los diferentes algoritmos que han surgido con el fin de manejar la congestión en redes IP se han desarrollado en los últimos años. El primer paso del manejo de la misma es su detección. Antiguamente, la detección de la congestión era muy difícil. Una terminación de temporización causada por un paquete perdido pudo deberse a (1) ruido en la línea de transmisión o (2) el descartamiento de paquetes en el enrutador congestionado, la determinación de esta diferencia constituía algo complejo. Hoy día, la pérdida de paquetes por errores de transmisión es poco visible, debido a que los sistemas troncales de larga distancia son de fibra óptica. Todos los algoritmos que emplean el nivel de transporte suponen que las terminaciones de temporización son causadas por congestiones.

Uno de los aspectos a tener en cuenta en las redes IP son los protocolos que utiliza el nivel de transporte. Surgen muchas propuestas para mejorar su eficiencia, como la de TCP (*Transmission Control Protocol*). Este es el protocolo más utilizado en las redes de datos actuales, brinda entrega confiable y ordenada de los datos, control de flujo y control de congestión, prestaciones que no posee el nivel de red y que son requeridas por innumerables aplicaciones.

El hecho de proporcionar una comunicación fiable entre dos hosts implica, entre otras cosas, que se necesita encontrar la forma de controlar que los paquetes enviados lleguen al destino sin extravíos en su trayecto; y proporcionar los medios necesarios para detectar la pérdida para una nueva transmisión. Estas funciones son implementadas por los algoritmos de control de congestión y de recuperación de segmentos perdidos del protocolo TCP: Inicio Lento/Prevención de la Congestión y Retransmisión Rápida/Recuperación Rápida. Las implementaciones TCP (Tahoe, Reno, New-Reno, SACK, Vegas, etc.) utilizadas para

el control de la congestión son basadas en la regulación del tráfico inyectado a la red, para minimizar la intensidad, dispersión y duración de la congestión en las redes IP.

Esta investigación persigue realizar un estudio profundo sobre las mecanismos de control de la congestión en redes IP, mediante un análisis teórico-práctico de las implementaciones TCP simulando un modelo de red con la herramienta OPNET Modeler 8.0, con el fin de evaluarlos y analizar su incidencia sobre el funcionamiento de la misma, así como los compromisos de la QoS (Calidad de servicio) negociadas para el flujo de datos ya establecido.

Los objetivos específicos se resumen como se muestra a continuación:

1. Construir el marco teórico o de referencia con los elementos necesarios que permitan llevar a cabo la investigación, derivados de la consulta de la literatura internacional y nacional más actualizada, y que pueda ser utilizado como documento de referencia con fines docentes, metodológicos e investigativos posteriores en esta temática.
2. Describir el funcionamiento de los mecanismos básicos para el control de la congestión utilizados por el protocolo TCP.
3. Diseñar un modelo de red IP capaz de mostrar el comportamiento de cada implementación TCP frente a una situación de congestión.
4. Analizar las versiones TCP existentes en la actualidad con la simulación de un modelo de red, para identificar las ventajas y problemas que presenta cada implementación.

El trabajo consta de tres capítulos. El primero concentra el basamento teórico de la investigación, En la segunda parte se describen los mecanismos básicos para el control de la congestión y las implementaciones TCP que utilizan estos algoritmos, y en el último capítulo se muestran los resultados del funcionamiento de las versiones TCP y su análisis comparativo, donde queda demostrado la efectividad de la versión TCP SACK.

CAPÍTULO 1. MECANISMOS DE CONTROL DE LA CONGESTIÓN.

Los nuevos servicios y tecnologías surgidos durante los primeros años del siglo XXI en Internet plantean serios problemas para los mecanismos de control de la congestión, gestión del ancho de banda y otros recursos de la red desarrollados [18]. Es más, el escaso conocimiento que se tiene sobre la dinámica del tráfico de Internet, así como la falta de enfoques de análisis y técnicas de ingeniería del tráfico en redes IP, suponen una importante limitación para la estabilidad y seguridad global de la red, el despliegue de nuevos servicios y el uso eficiente de nuevas tecnologías de comunicaciones.

Por otro lado, TCP (*Transmission Control Protocol*, **protocolo de control de la transmisión**) [3] se ha convertido en el factor estándar de los protocolos de transporte, y es empleado por la mayor parte de las aplicaciones, como la navegación Web, la transferencia de ficheros por FTP, el correo electrónico, etc. Todas las previsiones apuntan a que TCP seguirá siendo, en el medio y largo plazo, el protocolo de transporte dominante.

En este capítulo se realiza un análisis teórico sobre el tema, comenzando por una descripción del los protocolos del nivel de transporte de Internet: TCP (*Transmission Control Protocol*, **protocolo de control de la transmisión**) y UDP (*User Data Protocol*, **protocolo de datos de usuario**), se describen los mecanismos básicos para controlar el flujo de datos y para minimizar o evitar la intensidad, dispersión y duración de la congestión, con el fin de evaluarlos y ver su incidencia sobre el funcionamiento de la red teniendo así la base de conocimientos para el entendimiento de los temas que se tratan en los capítulos posteriores.

1.1 LOS PROTOCOLOS DE TRANSPORTE DE INTERNET: TCP Y UDP.

El nivel de red ofrece un servicio poco fiable, pero los usuarios y las aplicaciones necesitan de un servicio que les garantice la recepción y la corrección de todos los datos enviados, de esto se va ha encargar el nivel de transporte que no es solamente otro nivel sino que es el corazón de la jerarquía completa de protocolos. La tarea de este nivel es proporcionar un transporte de datos confiable y económico de la maquina de origen a la maquina de destino, independientemente de la red o redes físicas en uso. Sin el nivel de transporte, el concepto total de los protocolos en niveles tendría poco sentido. Otra manera de ver el nivel de transporte es considerar que su función primaria es mejorar la **QoS** (Quality *of Service*, **calidad de servicio**) proporcionada por el nivel de red. Si el servicio de red es impecable, el nivel de transporte tiene una tarea fácil. Si, por otra parte, el servicio de red es malo, el nivel de transporte tiene que construir un puente sobre el abismo que separa lo que quieren los usuarios de transporte y lo que el nivel de red proporciona. El servicio de transporte puede permitir que el usuario especifique valores preferidos, aceptables y mínimos para varios parámetros de servicios en el momento de establecerse una conexión. Es responsabilidad del nivel de transporte examinar estos parámetros y, dependiendo de los tipos de servicios de red disponibles, determinar si puede proporcionar el servicio requerido [1].

Internet o Redes IP tienen dos protocolos principales en el nivel de transporte, un protocolo orientado a conexiones (TCP) y uno sin conexiones (UDP), tal como se muestra en la Fig.1.1. Cada uno está diseñado para ofrecer un nivel de servicio diferente a las aplicaciones. La TPDU (*Transport Protocol Data Unit*, **unidad de datos del protocolo de transporte**) de TCP se denomina *segmento*, y la de UDP *mensaje* o también *datagrama UDP*.

1.1.1 UDP (User Data Protocol).

El protocolo UDP ofrece a las aplicaciones un mecanismo para enviar datagramas IP (*Internet Protocol*, **protocolo Internet**) en bruto encapsulados sin tener que establecer una

conexión. Muchas aplicaciones cliente-servidor que tienen una solicitud y una respuesta usan el UDP en lugar de tomarse la molestia de establecer y luego liberar una conexión; UDP provee un servicio simple sin control de flujo, continuación de mensajes u otra sofisticación que si ofrece TCP, no emplea acuses de recibos para asegurarse de que llegan mensajes, no ordena los mensajes entrantes, ni proporciona retroalimentación para controlar la velocidad a la que fluye la información entre las maquinas, por lo tanto, los mensajes UDP se pueden perder, duplicar o llegar sin orden, es decir no existe control de flujo ni de congestión. Aplicaciones típicas que usan UDP son: TFTP ("Trivial File Transfer Protocol"), DNS ("Domain Name System"), RPC ("Remote Procedure Call") usado por el NFS ("Network File System"), NCS ("Network Computing System"), SNMP ("Simple Network Management Protocol"), etc. [2].

Este protocolo se describe en el RFC 768 [5] el cual es un estándar oficial, todas las normas oficiales en la comunidad de Internet se publican como una RFC (*Request for Comment*). Además hay muchos RFCs que no son estándares oficiales, pero se publican con fines informativos [3].

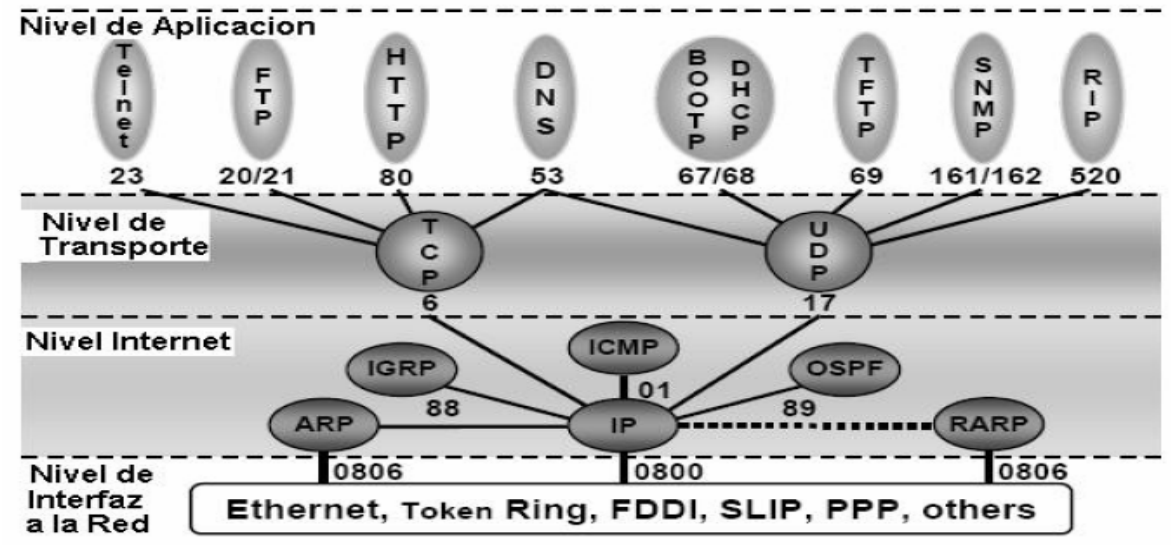


Fig. 1.1 Conjunto de protocolos TCP\IP

1.1.2 TCP (Transmission Control Protocol).

TCP es un protocolo orientado a la conexión, fiable y entre dos extremos, diseñado para encajar en una jerarquía en niveles de protocolos que soportan aplicaciones sobre múltiples

redes. TCP proporciona mecanismos para la comunicación fiable entre pares de procesos en computadoras 'host' ancladas en redes de comunicación de computadoras distintas, pero interconectadas. Se hacen muy pocas suposiciones sobre la fiabilidad de los protocolos de comunicación por debajo del nivel de TCP. Este protocolo sólo supone que puede acceder a un servicio de transmisión de datagramas simple, aunque en principio poco fiable, de los protocolos del nivel inferior. En principio, TCP debería ser capaz de operar encima de un amplio espectro de sistemas de comunicaciones que incluye desde conexiones por cables fijos ('hard-wired conections') hasta redes de intercambio de paquetes o redes de circuitos conmutados [4].

El protocolo se definió formalmente en el RFC 793 [4]. A medida que pasó el tiempo se detectaron varios errores e inconsistencias, y se cambiaron los requisitos de algunas áreas. Estas clasificaciones y algunas correcciones de fallas se datallan en el RFC 1122 [6]. En el RFC 1323 [7] se presentan las extensiones.

Las conexiones proporcionadas por el servicio de flujo TCP/IP permite la transferencia de datos en ambas direcciones simultáneamente y se le denomina full-dúplex.
TCP proporciona al receptor un medio para controlar la cantidad de datos enviados por el emisor. Esto se consigue devolviendo una "ventana" con cada ACK (acknowledgment, reconocimiento), indicando el rango de números de secuencia aceptables más allá del último segmento recibido con éxito. La ventana indica el número de octetos que se permite que el emisor transmita antes de que reciba el siguiente permiso.
Los usuarios de TCP pueden indicar el nivel de seguridad y prioridad de su comunicación. Se emplean valores por defecto cuando estas características no se necesiten [4].
El servicio TCP se obtiene haciendo que tanto el transmisor como el receptor creen puntos terminales, llamados sockets, que consisten en una concatenación entre la dirección IP, origen o destino, con la aplicación (ports, puerto), también origen o destino, que participan en la comunicación. Puerto es el nombre en TCP de un TSAP (Transport Service Access Point, punto de acceso al servicio de transporte). Todas las operaciones de transferencia se realizan a través de estos sockets. Los números de puerto por debajo de 1024 son denominados puertos bien conocidos (*well-known ports*) y son reservados para servicios estándar, la lista de estos puertos bien conocidos está en el RFC 1700 [2, 8].

La entidad TCP transmisora y receptora intercambia datos en forma de segmentos. Un segmento consiste en una cabecera TCP fija de 20 bytes (más una parte opcional) seguida de ceros o mas bytes de datos. Tras las opciones, si las hay, pueden continuar hasta 65535-20-20=65515 bytes de datos, donde los primeros 20 se refieren a la cabecera IP y los segundos a la cabecera TCP como se muestra en la Fig. 1.3, los segmentos sin datos son legales y se usan para acuses de recibo y mensajes de control [1].

Hay dos límites que restringen el tamaño de segmento. Primero, cada segmento, incluida la cabecera TCP tiene que caber en la carga útil de 65535 byte del IP. Segundo cada red tiene una MTU (*maximum transfer unit*, **unidad máxima de transferencia**) ver Fig. 1.2.

En la RFC 793 [4] se detalla todo lo relacionado sobre la cabecera del segmento TCP, con la explicación de cada uno de sus campos ver Fig. 1.3, en este capítulo se profundiza sobre la gestión de una conexión TCP y su política de transmisión.

Network	MTU (bytes)
Hyperchannel	65535
16 Mbits/sec token ring (IBM)	17914
4 Mbits/sec token ring (IEEE 802.5)	4464
FDDI	4352
Ethernet	1500
IEEE 802.3/802.2	1492
X.25	576
Point-to-Point (low delay)	296

Fig. 1.2. Principales unidades de transferencias (MTUs).

Las conexiones TCP son establecidas y mantenidas a partir de demandas de las aplicaciones y son mantenidas activas hasta que las aplicaciones explícitamente las liberen [9]. O sea que TCP es complejo y aunque describe como los programas de aplicación lo pueden utilizar de forma general, no aclara los detalles de la interfaz entre un programa de aplicación y el mismo.

La razón de esto es la flexibilidad, dado que los programadores implantan TCP/IP en el sistema operativo de una estación y después necesitan emplear la interfaz que proporciona dicho sistema operativo. TCP permite que varios programas de aplicación en una estación se comuniquen de manera concurrente y realiza el de-multiplexado del tráfico TCP entrante entre los programas de aplicación [10, 2].

Es importante recordar que TCP es un protocolo orientado a la conexión y que requiere que ambos puntos extremos estén de acuerdo en participar. Para iniciar la conexión el programa de aplicación en el extremo **Cliente** realiza una función de apertura activa al contactar a su sistema operativo (a través de una Interfaz de Programación de Aplicaciones, API) para establecer una conexión tal como se muestra en la Fig. 1.4 acción 2.

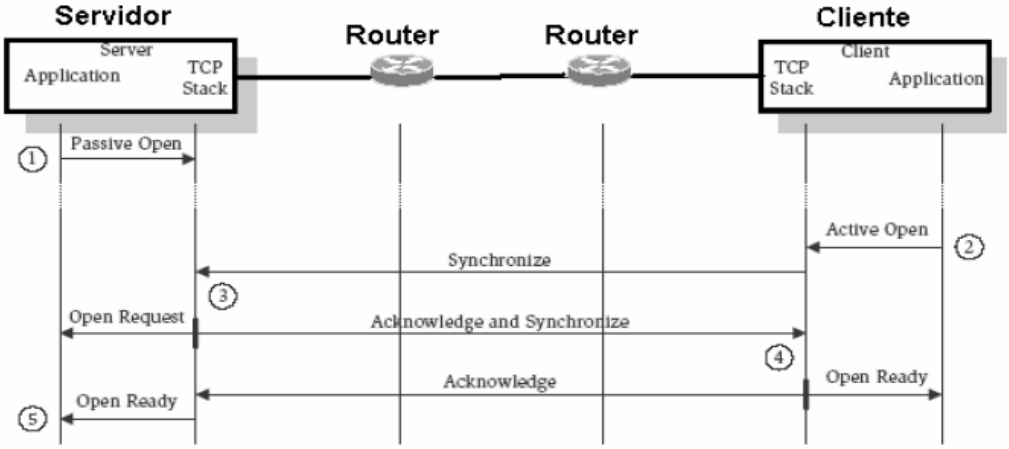


Fig. 1.4. Flujo de mensajes para establecer una conexión TCP.

Dos populares interfaces de programación de aplicaciones (APIs) para aplicaciones que utilizan los protocolos TCP/IP se llaman sockets y TLI (*Transport Layer Interface*; **interfaz de capa de transporte**). El primero es a veces llamada "Berkeley sockets", indicando el lugar en que se desarrolló originalmente. Este último, desarrollado originalmente por AT & T, a veces se llama XTI (X/Open Transport Interface), en reconocimiento de la labor realizada por X / Open, un grupo internacional de fabricantes de computadoras que producen su propio conjunto de normas. XTI es de hecho un superconjunto de TLI. [3].

En el extremo del **Servidor** en que se recibirá la solicitud deberá primero recibir la solicitud de la aplicación a la entidad TCP de apertura pasiva (Ver Fig. 1.4 acción 1) y mantenerse a la espera. Los dos módulos de software TCP se comunican para establecer y llevar a cabo la conexión.

Para establecer una conexión entre las entidades TCP del cliente y el servidor se usa un protocolo de acuerdo de tres vías (three-way handshake) [1].

En la primera etapa (acción 3 Fig.1.4) se envía un segmento que tiene activa la bandera **SYN** en el campo de código de la cabecera TCP. En la segunda etapa el receptor envía un

segmento de respuesta con las banderas **SYN y ACK** activas (acción 4 Fig.1.4), indicando el acuse de recibo del primer segmento **SYN** como el hecho de que continúan con el intercambio. El mensaje final del acuerdo (**ACK**) es solo un acuse de recibo y nada mas se utiliza para informar al destino que ambos extremos están de acuerdo en establecer una conexión, tal como se muestra en la Fig. 1.4. Cuando la bandera **SYN** está activa, el campo de número de secuencia indica el número de secuencia inicial (ISN) y el primer octeto de datos es ISN+1 [2, 9].

Se han ido desarrollando nuevas aplicaciones que requieren de otros protocolos de transporte más eficientes. De ahí que se hayan desarrollado variantes experimentales de TCP como la conocida como T/TCP (Transactional TCP) descrita en las RFC 1379 y 1644. Otra variante propuesta es el SCTP (Stream Control Transmisión Protocol), descrita en la RFC 2960, la cuál ha sido diseñada específicamente para encontrar los requerimientos de transporte de mensajes de señalización en las Redes Telefónicas de Conmutación de Paquetes.

1.2 CONTROL DE FLUJO.

Un flujo de datos envíado sobre una conexión de TCP se entrega de forma fiable y ordenada al destino. La transmisión es fiable gracias al uso de números de secuencia y de acuses de recibo. Básicamente, se le asigna un número de secuencia a cada octeto de datos. El número de secuencia del primer octeto de datos en un segmento se transmite con ese segmento y se le denomina el número de secuencia del segmento. Los segmentos también llevan un número de acuse de recibo que es el número de secuencia del siguiente octeto de datos esperado en la transmisión en el sentido inverso. Cuando el módulo de TCP transmite un segmento conteniendo datos, pone una copia en una cola de retransmisión e inicia un contador de tiempo; si llega el acuse de recibo para esos de datos, el segmento se borra de la cola. Si no se recibe el acuse de recibo dentro de un plazo de expiración, el segmento se retransmite. La llegada del acuse de recibo no garantiza que los datos ya hayan sido entregados al usuario final, sino únicamente que el TCP receptor ha asumido la responsabilidad de hacerlo [4].

1.2.1. Mecanismo de Ventana Deslizante.

La ineficiencia de algoritmos como el explicado anteriormente causa que la mayor parte del tiempo la red no transporte datos sino reconocimientos (ACK). Por ello TCP usa un mecanismo de ventana deslizante para:

• Garantizar la entrega fiable de datos.

• Garantizar que los datos serán entregados en orden.

• Forzar un control de flujo entre el emisor y el receptor.

El control de flujo en el nivel de transporte es muy importante porque las velocidades a las que pueden llegar los datos al receptor son muy variables y de este modo el receptor puede comunicarle al emisor lo que está preparado para recibir. Este control se implementa mediante el uso de la ventana deslizante que también se le conoce como un esquema de asignación de créditos. Para este esquema de créditos cada octeto individual de datos que es transmitido es considerado que tiene un número de secuencia. En adición al dato, cada segmento que es transmitido incluye en su encabezado tres campos relacionados con el control de flujo: Número de Secuencia (SN), Número de Reconocimiento (AN) y la Ventana (W) [2].

1.2.2 Problemas que afectan el desempeño de la red. Soluciones.

Un grave problema que puede afectar el desempeño (*performance*) de la entidad TCP es el síndrome de ventana tonta (*silly windows syndrome*). Este problema ocurre cuando pasan datos a la entidad transmisora en bloques grandes, pero una aplicación interactiva del lado del receptor lee datos de 1 byte a la vez. La solución de Clark [11] es evitar que el receptor envíe una actualización de ventana para 1 byte. En cambio, se le obliga a esperar hasta tener disponible una cantidad decente de espacio, y luego lo anuncia.

Otro caso extremo de baja eficiencia se produce cuando el transmisor envía paquetes de 41 bytes que contienen 1 byte de datos. Una manera de reducir este uso es empleado en el algoritmo de Nagle [12]. Lo que sugirió este señor es muy sencillo: cuando el tráfico de la aplicación llega al TCP en bytes uno por uno se envía el primero en un segmento y se

retienen los demás hasta recibir el ACK correspondiente al byte enviado; una vez recibido el ACK se envía un segmento con todos los bytes que hubiera pendientes y se empieza a acumular de nuevo hasta recibir el siguiente ACK. También se envía un segmento si el número de caracteres acumulados en el buffer es igual a la mitad de la ventana, o al máximo tamaño del segmento. En cierto modo el algoritmo de Nagle sustituye el protocolo de ventana deslizante por un mecanismo de parada y espera.

El algoritmo de Nagle y la solución de Clark al síndrome de la ventana tonta son complementarios. Nagle trataba de resolver el problema causado por la entrega de datos a la entidad TCP desde la aplicación transmisora un byte a la vez. Clark trataba de resolver el problema de que la aplicación receptora toma los datos de la entidad TCP un byte a la vez. Ambas soluciones son válidas y pueden operar juntas. La meta es que el transmisor no envié segmentos pequeños y que el receptor no los pida [1].

1.2.3 Estrategias de Retransmisión.

El tratamiento de la temporización y la retransmisión es fundamental en TCP**,** para llevar a cabo este tratamiento se han establecido las siguientes estrategias:

a) Algoritmo de retransmisión adaptativo.

El ajuste del temporizador de retransmisión, RTO (*Retransmission Time Out*), es especialmente crítico en TCP, al actuar tanto en redes locales, en enlaces punto a punto o en una red tan cambiante como Internet. Debe asegurarse un mecanismo que funcione correctamente en entornos tan diferentes como en los que opera TCP. RTO debe ser suficientemente pequeño como para responder rápidamente a las pérdidas, pero no tanto como para forzar la retransmisión de datos que han sufrido un pico de retardo en la red sin haber llegado a perderse, como sería el caso de congestión. Para adaptarse a los retardos variables característicos de un entorno como Internet, TCP usa un algoritmo de retransmisión adaptativo que monitoriza el retardo en cada conexión y ajusta el valor de RTO de acuerdo con ese valor. La especificación del protocolo sugiere tomar muestras del tiempo de ida y vuelta, RTT (*Round Trip Time*), calculado como la diferencia de tiempo entre la emisión de un segmento y la recepción de su reconocimiento. Con esta

información, TCP puede ajustar dinámicamente una variable que identifique el tiempo medio de ida y vuelta de la siguiente forma:

$$RTT = \alpha * RTT_{anterior} + (1-\alpha) * RTT_{nuevo}$$

Van Jacobson fue quien originó este algoritmo y realizó profundas investigaciones para perfeccionarlo todo esto se puede encontrar en [13] y en la RFC 2898 se añade un refinamiento al cálculo del RTO.

b) Algoritmo de Karn.

Cuando se recibe un reconocimiento de un segmento que ha sido objeto de retransmisión, es imposible determinar si éste corresponde al primer segmento transmitido o a su posterior retransmisión. El algoritmo de *Karn* [14] evita esta ambigüedad. Este algoritmo establece que el cálculo del RTT estimado para determinar RTO debe hacerse ignorando las muestras correspondientes a segmentos retransmitidos.

c) Marca Temporal o *Timestamp*.

Existe una solución alternativa para la medida exacta de RTT que hace desaparecer la ambigüedad, de forma que hace innecesaria la aplicación del algoritmo de *Karn*. Esta solución se basa en aprovechar el ancho de banda para mandar la información de tiempo en cada segmento. De esta forma puede realizarse una medida por cada segmento enviado independientemente de si se trata de un segmento retransmitido o no, obteniendo así más medidas. Esta solución, si bien es adecuada para redes de alta velocidad en las que el ancho de banda no es un recurso escaso.

d) *Backoff* exponencial o RTO exponecial de vuelta atrás.

Cuando en una entidad emisora TCP se vence el tiempo de espera de un reconocimiento positivo por el envío de un segmento, entonces se tiene que retransmitir el segmento de nuevo. La RFC 793 considera que el mismo valor de RTO debe ser utilizado para este nuevo segmento. Sin embargo considerando que el vencimiento del tiempo de espera pudo ser debido a la congestión, producido por el descarte de paquetes o a largos retardos en la trayectoria de ida y vuelta, indica que mantener ese RTO es una mala consideración.

Una política mas sensible dicta que una fuente TCP incrementa sus RTO cada vez que el mismo segmento es transmitido y este proceso es referido como de vuelta atrás (backoff). Una técnica simple para implementar el RTO de vuelta atrás es la de multiplicar el RTO para cada segmento por un valor constante para cada retransmisión:

$$RTO_{nuevo} = ß\ RTO_{anterior} \qquad \text{con} \qquad ß = 2 \text{ (normalmente)}$$

Lo que provoca que el RTO crezca exponencialmente con cada retransmisión, de ahí que se le conozca como el mecanismo de vuelta atrás binario exponencial (backoff).

Hay dos problemas conocidos con el cálculo de RTO especificado en el RFC-793. Primero, la medición exacta de RTT es difícil cuando hay retransmisiones. Segundo, el algoritmo para calcular el tiempo estimado de ida y vuelta suavizado (smoothed round-trip time) es insuficiente [13], porque incorrectamente supone que la variación en los valores de RTT sería pequeña y constante. Estos problemas fueron resueltos por Karn y el algoritmo de Jacobson, respectivamente [15].

1.3 CONTROL DE LA CONGESTIÓN.

Cuando se definió e implementó TCP, las redes existentes presentaban como problema principal una baja fiabilidad, es decir, la presencia de errores era la característica limitante del comportamiento eficiente de la red. Las situaciones de congestión, causa principal del deterioro del comportamiento de las redes actuales, no fueron tenidas en cuenta y, por ello, no se especificó mecanismo alguno para su control. Con el tiempo, sin embargo, se han ido desarrollando una serie de algoritmos con ese propósito.

El hecho de proporcionar una comunicación fiable entre dos hosts implica, entre otras cosas, que hay que encontrar la forma de controlar que los paquetes enviados lleguen al destino sin perderse en el camino; y sí se pierden, proporcionar los medios necesarios para detectar la pérdida y volver a reenviarlos. Estas funciones son implementadas por los algoritmos de control de congestión y de recuperación de segmentos perdidos del protocolo TCP: Inicio Lento/Prevención de la Congestión y Retransmisión Rápida/Recuperación Rápida. Estos algoritmos se concibieron en [13, 16, 32] La definición de estos cuatros algoritmos entrelazados aparece en la RFC 2581 [17].

CAPÍTULO 2. IMPLEMENTACIONES TCP Y EL CONTROL DE LA CONGESTIÓN.

El uso de Internet aumenta de manera exponencial, también lo hacen los problemas de la congestión en redes IP. Este crecimiento ha originado que los investigadores de los Laboratorios Lawrence Berkeley (**LBL**) se cuestionen sobre los mecanismos empleados por las implementaciones TCP para combatir la congestión. La mayor parte de las mejoras introducidas a este protocolo fueron estimulados por este grupo (LBL), liderado por Van Jacobson, que ha alcanzado casi la condición de superhombre en la creación de redes.

Las primeras implementaciones surgieron en los años 80' por parte de los sistemas BSD (*Berkeley Software Distribution*, **Distribución de Software Berkeley**), y se utiliza para identificar un sistema operativo derivado del sistema Unix nacido a partir de las contribuciones realizadas a ese sistema por la Universidad de California en Berkeley.

En este capítulo se detallan los mecanismos básicos del control de la congestión**:** Inicio Lento (Show Start), Prevención de la Congestión (Congestion Avoidance), Retransmisión Rápida (Fast Retransmit) y Recuperación Rápida (Fast Recovery). Se describen las implementaciones TCP en orden cronológico: TCP Tahoe, TCP Reno, TCP New-Reno, TCP SACK *(Selective Acknowledgment,* **Reconocimientos Selectivos**) y TCP Vegas. Por otra parte se describen las principales características de la herramienta de simulación (OPNET Modeler), con el objetivo de mostrar los escenarios simulados.

2.1 MECANISMOS PARA EL CONTROL DE LA CONGESTIÓN.

Los métodos utilizados por TCP para el control de la congestión están basados en la regulación del tráfico inyectado a la red. Esto supone que se introducen funciones que permiten estudiar la posibilidad de enviar más tráfico a través del enlace, detectar cuándo se ha superado la capacidad del mismo y si se debe disminuir la carga de datos. Estos métodos pueden agruparse en dos categorías en cuanto a su forma de trabajo, *congestion control* (control de congestión) el cual constituye un mecanismo reactivo que busca restituir la normalidad en los enlaces cuando se detecta congestión, y *congestion avoidance* (prevención de la congestión), mecanismo proactivo que busca no alcanzar una situación de congestión [31].

2.1.1 Algoritmos de Inicio Lento (*Slow Start*) y Prevención de la Congestión (*Congestion Avoidance*).

Estos algoritmos se basan en el principio de equilibrio en la conexión: "*la tasa a la cual deberían inyectarse nuevos paquetes en la red es la tasa a la que llegan nuevos reconocimientos*". TCP es un protocolo autosincronizado que utiliza los reconocimientos como marcas para inyectar nuevos paquetes en la red. Cuando no hay segmentos en tránsito, no existen reconocimientos (ACK) que permitan activar tal comportamiento como sucede al inicio de una conexión o tras una expiración del temporizador de retransmisión. Los algoritmos *Slow Start* y *Congestion Avoidance*, son totalmente independientes, en la práctica, se utilizan de manera conjunta, estos deben ser utilizados por la fuente TCP a los efectos de controlar la cantidad de tráfico inyectado en la red [6].

Existen dos variables para el control de la congestión: una ventana de congestión (**cwnd**) que limita la cantidad de datos que TCP puede enviar, y un valor umbral (**ssthresh**) que permite conmutar entre los dos algoritmos los cuales funcionan como se muestra: Inicialmente:

$$\textbf{cwnd} \leq \textbf{2 * SMSS bytes} \qquad \textbf{(2.1)}$$

Donde:

SMSS: (Sender Maximum Segment Size), Tamaño máximo del segmento que el emisor puede transmitir, el tamaño no incluye cabeceras TCP / IP ni Opciones [17])

$$\textbf{ssthresh = Wmáx} \qquad \textbf{(2.2)}$$

Donde:

Wmáx: Tamaño máximo de la ventana de transmisión

El receptor envía un reconocimiento por cada segmento recibido, el emisor enviará 1 segmento durante el primer RTT (Tiempo de ida y vuelta), 2 segmentos durante el segundo, 4 durante el tercero y así sucesivamente, donde se evidencia un incremento exponencial de la ventana, como se muestra en la Fig. 2.1.

- Cuando ocurre un RTO (time out), la mitad del tamaño actual de la ventana se guarda en la variable ssthresh ver ecuación (2.3)

$$\textbf{ssthresh = máx (FlightSize / 2, 2 * SMSS)} \qquad \textbf{(2.3)}$$

Donde:

FlightSize: Cantidad de datos pendientes en la red.

Se reinicia el mecanismo como se muestra en la Fig. 2.1. La llegada de un ACK que reconoce nuevos datos, incrementa el valor de cwnd según el siguiente procedimiento:

Si **cwnd** $\leq$ **ssthresh** se realiza el proceso de inicio lento por lo que se incrementa el valor de la ventana de congestión según ecuación (2.4)

$$\textbf{cwnd =cwnd + SMSS} \qquad \textbf{(2.4)}$$

Si **cwnd** $\geq$ **ssthresh** se realiza el proceso de prevención de la congestión y se observa el incremento de la ventana de congestión en la ecuación (2.5)

$$\textbf{cwnd + = SMSS * SMSS / cwnd.} \qquad \textbf{(2.5)}$$

El algoritmo de Prevención de la Congestión persigue adaptar la transferencia a la situación de la red en cada momento, reaccionando ante posibles estados de congestión, puesto que TCP asume que las pérdidas de paquetes son debidas a dicha situación.

El emisor siempre envía el mínimo entre la ventana de congestión (CongWin) y el tamaño del buffer de recepción (RcvWin)

$$\textbf{LastByteSent - LastByteAcked} \leq \textbf{Mín[CongWin, RcvWin]} \qquad \textbf{(2.6)}$$

Donde:

LastByteSent: Último byte enviado por el emisor.

LastByteAck: Último byte reconocido.

El valor de la ventana de transmisión se determina por el mínimo de dos límites: uno impuesto por el receptor que indica el tamaño del buffer de recepción disponible y el otro impuesto por el transmisor denominado ventana de cogestión [33].

La Figura 2.1 muestra la evolución de la ventana de congestión para los algoritmos de Inicio Lento y Prevención de la Congestión, el umbral de congestión se encuentra en 32 segmentos (línea discontinua), a partir del cual se pasa de la fase de Inicio Lento (crecimiento exponencial) a la fase de Prevención de la Congestión (crecimiento lineal). También se aprecia como tras una retransmisión, el umbral de congestión se reduce a la mitad de la ventana de congestión y se establece otra vez el proceso de inicio lento.

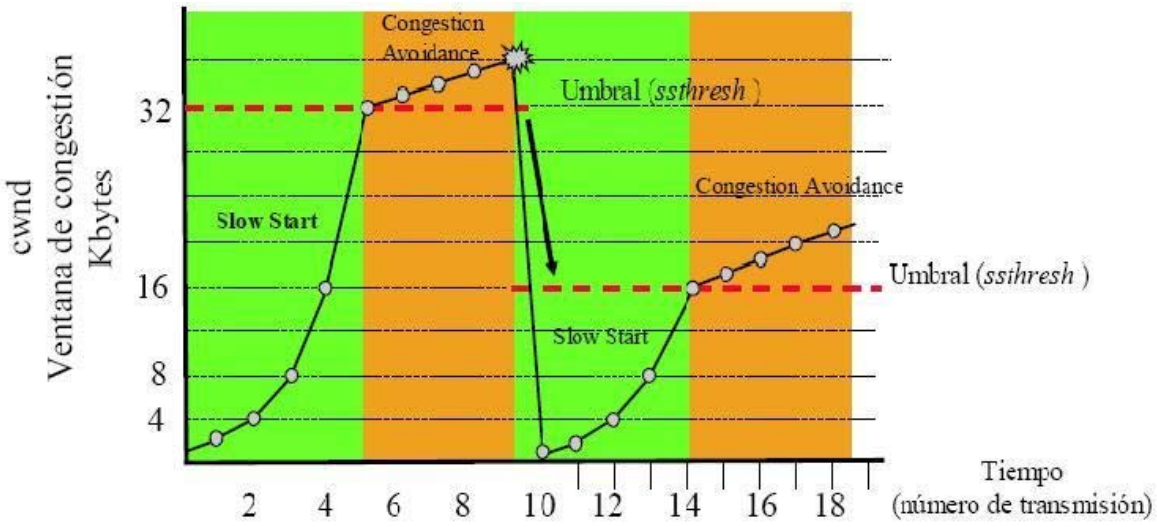


Fig. 2.1 Algoritmos de Inicio Lento y Prevención de la Congestión

Existen propuestas [RFC2414, RFC2415] para solucionar los inconvenientes que presenta este mecanismo en el inicio de la comunicación, permitiendo una apertura más rápida de la ventana de congestión. Las técnicas propuestas tienen importancia en transferencias cortas de datos, donde la fase inicial es relevante en la comunicación de los mismos.

2.1.2 Algoritmos de Retransmisión Rápida (*Fast Retransmit*) y Recuperación Rápida (*Fast Recovery*)

Con la evolución de las redes y los protocolos, se van incorporado nuevos mecanismos, que poseen el propósito de mejorar el comportamiento del protocolo. El mecanismo de

Retransmisión Rápida intenta optimizar y ajustar los algoritmos descritos anteriormente para adecuar el protocolo TCP a situaciones de congestión y pérdidas en general así como su recuperación. La primera de las mejoras propuestas que está presente en la mayoría de implementaciones, aprovecha la recepción de reconocimientos duplicados los cuales contienen el mismo número de secuencia. De esta forma se activa la retransmisión de un segmento perdido antes de la expiración del temporizador de retransmisión (RTO).

La recepción de un ACK duplicado puede estar provocada por las siguientes situaciones:

- La red desordena paquetes y, en consecuencia, es posible que el receptor haya enviado un reconocimiento duplicado ante la llegada de un segmento que no sigue la secuencia normal. Hay que tener en cuenta que todas las implementaciones modernas de TCP generan, inmediatamente, un ACK duplicado al recibir un segmento fuera de orden.
- Se ha perdido algún segmento de datos tanto por errores del canal o por problemas de congestión. En este caso el TCP recibe segmentos fuera de orden y en consecuencia genera ACK duplicados.
- Se ha producido un pico de retardo en la red, es decir, el tiempo de ida y vuelta de un paquete se ha incrementado repentinamente provocando la expiración del temporizador de retransmisión del emisor con la consiguiente retransmisión del segmento conflictivo. Puesto que el segmento en cuestión ya había sido recibido correctamente, la recepción de su réplica provoca la generación de un ACK duplicado. Esta situación se producirá básicamente en casos de congestión.

El emisor, desconoce a cuál de estas razones responde la recepción del ACK duplicado. Experimentalmente se ha comprobado que no resulta apropiado precipitar la retransmisión ante la recepción del primer ni el segundo reconocimiento duplicado en redes que desordenan paquetes, porque puede tratarse de un simple problema de reordenación de rápida solución. Por esta razón, la mayoría de implementaciones TCP activan el mecanismo de Retransmisión Rápida al recibir el tercer ACK duplicado (en total 4 ACK), cuando éste llega, el emisor no espera a la expiración del temporizador sino que retransmite inmediatamente el segmento perdido.

Cuando una entidad TCP retransmite un segmento utilizando el mecanismo de retransmisión rápida, es porque considera que ese segmento está perdido, aún cuando no se ha alcanzado el tiempo de espera del mismo. Por lo que la entidad TCP debe tomar medidas

con vistas a evitar la congestión. Una estrategia es la de combinar el inicio lento combinado con el procedimiento de evitar la congestión utilizado cuando un tiempo de espera se alcanza. Van Jacobson plantea que eso es muy conservador y que en la realidad hay múltiples reconocimientos ACK, de ahí que propone una técnica de recuperación rápida en sustitución del inicio lento exponencial.

La misma es establecida a continuación:

- Establecer ssthresh al valor que resulta de la aplicación de la ecuación **(2.3).**
- Retransmitir el segmento que se supone que se ha perdido y establecer cwnd:

$$\mathbf{cwnd = ssthresh + 3\ SMSS} \qquad \mathbf{(2.7)}$$

  Este incremento se conoce como inflado artificial de cwnd, que refleja el hecho de que los tres ACK duplicados recibidos han abandonado la red y que el receptor de los segmentos que los originaron mantienen a éstos en memoria.
- Para cada nuevo ACK duplicado se incrementa cwnd en el tamaño máximo de un segmento (SMSS), de esta forma se refleja el hecho de que un segmento ha dejado la red.
- Transmitir un segmento nuevo siempre que cwnd o la ventana de transmisión establecida por el receptor lo permita.

A la llegada de un ACK que asienta nuevos datos (no un ACK duplicado), este debe comprender el segmento perdido y todos los enviados después de este y hasta la recepción del tercer ACK duplicado, entonces se establece el valor de cwnd al valor de ssthresh dado en el primer paso (lo que se conoce como desinflado de cwnd), finalizando el algoritmo de Recuperación Rápida

2.2 PRINCIPALES IMPLEMENTACIONES TCP.

Durante el inicio de TCP surgieron tres implementaciones base: **Tahoe**, la de 4.3BSD-Tahoe en 1988, derivado de 4.3BSD(mejor rendimiento de TCP,1986), **Reno**, la de 4.3BSD-Reno en 1990, derivada de Tahoe, usada posteriormente en 4.4BSD-Lite y por último **Vegas**, que supera a Reno en velocidad entre un 37 y 70%. Es de gran importancia la implementación TCP de 4.4BSD-Lite, que apareció en junio de 1994, y se toma como referencia para muchos sistemas actuales, hoy en día, se sigue usando para ilustrar el funcionamiento de un sistema TCP [19].

El Anexo 1 muestra la cronología de las distintas versiones BSD, lo que indica la importancia de las características TCP / IP. Se han desarrollado varias versiones que mejoran a Tahoe, Reno como: New-Reno en versión de Reno con un sistema mejorado del mecanismo de recuperación rápida y SACK (Selective Acknowledgment) que usa confirmaciones selectivas [20].

El comportamiento de TCP se especifica en las RFC (*Request for Comments*). Sin embargo, no hay un TCP "estándar", sino que existen versiones de TCP que emplean los distintos algoritmos para el control de la congestión. En la Fig. 2.2 se muestra un resumen de las versiones TCP más conocidas y los mecanismos de control de la congestión utilizados en cada una de ellas.

2.2.1 TCP TAHOE.

La versión 4.3BSD-Tahoe fue la primera en incorporar los mecanismos de control de congestión y de estimación de RTT (tiempo de ida y vuelta) propuestos por Van Jacobson en 1988. El primero de los mecanismos introducidos fue Inicio Lento entrelazado con Prevención de la Congestión, esta implementación fue mejorada al introducirle otro de los mecanismos básicos ya explicados: Retransmisión Rápida, el cual utiliza los ACK duplicados, así el emisor no espera a la expiración del temporizador (RTO) sino que retransmite inmediatamente el segmento perdido [13].

	TAOHE	RENO	NEW-RENO	SACK
Inicio Lento, Prevención de la Congestión	SÍ	SÍ	SÍ	SÍ
Retransmisión Rápida	SÍ	SÍ	SÍ	SÍ
Recuperación Rápida	NO	SÍ (Reno)	SÍ (New-Reno)	SÍ (Reno ó New-Reno)
Reconocimientos Selectivos	NO	NO	NO	SÍ

Fig. 2.2 Versiones TCP

TCP Tahoe posee los mecanismos básicos del control de la congestión. No obstante, el principal inconveniente que presenta es la utilización del algoritmo Inicio Lento cada vez

que ocurre una pérdida de paquete, provocando un retardo elevado en la transmisión de datos, causa fundamental del bajo rendimiento de la red.

2.2.2 TCP RENO.

TCP Reno surge en el año 1990 se conoce también por el nombre de BSD Network Release 2.0 (BNR2). TCP Reno continuó usando las mejoras incorporadas en TCP Tahoe, así mismo modificó el algoritmo de Retransmisión Rápida para adicionar el mecanismo Recuperación Rápida. Esta nueva implementación previene que al ocurrir una rápida retransmisión no descienda al mínimo el valor de la ventana de congestión y por tanto el rendimiento de la red, permitiendo una recuperación más rápida. Van Jacobson [13] propone mejorar el desempeño de la red si se aplica Inicio Lento solo cuando se retransmite por expiración del temporizador (RTO) y no por 3 ACK duplicados o en el inicio de una conexión [20, 21]. TCP Reno transmite a lo sumo un paquete perdido por RTT. Reno mejora significativamente con respecto a Tahoe cuando un segmento de datos se pierde en una ventana, pero puede afectar el rendimiento cuando hay pérdidas consecutivas de paquetes.

2.2.3 TCP NEW-RENO.

TCP New Reno surgió en abril de 1990 junto a TCP Reno y se estandariza en el RFC 2582 [22]. Implementa una modificación del algoritmo de Rápida Recuperación, que comienza al recibirse 3 ACK duplicados, almacenando en una variable el número de secuencia más alto transmitido. En caso de recibirse ACK de todos los datos enviados por el transmisor, incluso durante la fase de Rápida Recuperación, o de darse un RTO en la retransmisión, pasa a la fase de Prevención de la Congestión. Para comenzar una fase Rápida Recuperación no debe estarse en la misma. Tras la recepción de los tres ACK duplicados ssthresh toma el valor dado por la ecuación 2.3.

Luego retransmite el segmento perdido y cwnd toma el valor dado por la ecuación 2.7.

Para cada ACK duplicado recibido seguirá un criterio similar y transmitirá un nuevo segmento. Ante la llegada de un ACK, puede suceder que sea un ACK de todos los datos pendientes de confirmar, lo que implica un reconocimiento de todos los segmentos enviados entre el que se perdió y la recepción del tercer ACK duplicado. Aquí cwnd toma el valor de la ecuación 2.8:

cwdn = Min{ssthresh, FlightSize + SMSS) **(2.8)**

ó **cwnd = ssthresh** (valor tomado al comienzo de la fase de Recuperación Rápida), esto implica un "desinflado de ventana".

A continuación se sale de la fase de Recuperación Rápida.

Si el ACK no cubre todos los datos, se trata de un reconocimiento parcial, por lo que se retransmite el primer segmento no reconocido y se realiza un "desinflado de ventana parcial" que contemple los segmentos reconocidos y de forma que cuando se salga de la Recuperación Rápida aproximadamente sshthresh datos hayan abandonado la red., el transmisor sigue en la fase de Recuperación Rápida. Por lo tanto, cuando múltiples paquetes son perdidos en una ventana de datos, New Reno retransmite un paquete perdido por RTT hasta que todos los paquetes perdidos de esa ventana de datos han sido retransmitidos. New Reno permanece en Recuperación Rápida hasta que todos los datos pendientes, cuando Recuperación Rápida se inició, son confirmados.

Es posible aplicar técnicas más agresivas de retransmisión ante la llegada de un ACK parcial que impliquen la retransmisión de más de un segmento lo que podría ocasionar que se retransmitan segmentos innecesariamente.

Se consideran dos variantes del New Reno, *Impatient* (Impaciente) y *Slow-but-Steady* (Lento-pero-Constante). En la primera, sólo se pone a cero el RTT luego del primer ACK parcial y de haber varios segmentos perdidos en la misma ventana, luego de expirado el temporizador se comienza la fase de Inicio Lento. En la segunda, se pone a cero el RTT luego de cada ACK parcial por lo que se puede retransmitir un segmento luego de cada RTT [23].

La ausencia de la opción SACK no permite inferir demasiada información en el momento de la recepción de un ACK duplicado. No se puede concluir cuál o cuáles fueron los segmentos que dispararon el duplicado, tampoco permite distinguir entre los casos en los cuales se generó debido a un paquete demorado o perdido o si se trata de un ACK a una retransmisión de un segmento que llegó a destino. Por lo tanto, pérdidas múltiples en una ventana pueden ocasionar múltiples e innecesarios Retrasmisiones Rápidas.

A pesar de que TCP New Reno es capaz de manejar múltiples pérdidas en una ventana, tiene la limitación de permitir solamente el envío de un segmento por RTT. Esto no es muy aceptable en casos en que tanto el retardo como el ancho de banda se incrementan.

2.2.4 TCP SACK (Selective Acknowledgment, Reconocimientos Selectivos).

La pérdida de múltiples segmentos de una ventana de datos puede ser catastrófica para el rendimiento de TCP. El uso del mecanismo denominado ACK acumulativo (Los segmentos recibidos que no estén en el borde izquierdo de la ventana de recepción no serán reconocidos.) ocasiona que el transmisor tome dos caminos, o espere un RTT para saber de segmentos perdidos o realice retransmisiones innecesarias que implicarán múltiples descartes. La opción SACK puede ayudar a sobrellevar estos inconvenientes [24].

TCP SACK brinda un mecanismo para proporcionar más información sobre los segmentos que han sido recibidos correctamente y el transmisor puede conocer cuáles son los segmentos que se han perdido. SACK es un campo opcional en el encabezado TCP, que es enviado cuando se reciben datos fuera de secuencia. Esta opción contiene una lista de los bloques contiguos y aislados de datos recibidos y guardados en cola, con la información necesaria para identificarlos (números de secuencia de 32 bits que ofician de "bordes" en los bloques). Por la definición del encabezado de TCP, no es posible especificar más de tres bloques. De esta forma el extremo transmisor consigue la información de los subsecuentes bloques no recibidos por el destinatario. Puede resultar conveniente el uso de esta técnica en conjunción con la técnica de RTTM (Round Trip Time Measurement, Mediciones del tiempo de ida y vuelta) basada en timestamps [7] de forma de obtener más información sobre segmentos perdidos [25].

La extensión SACK hace uso de dos opciones de TCP. La primera, SACK-permitted, está asociada a la habilitación de la opción SACK y se envía como 2 bytes y solamente en el segmento SYN. La segunda, es la opción en si misma. Que el transmisor haya enviado la opción SACK-permitted en el segmento SYN no implica que el receptor deba utilizarla al recibir bloques discontinuos. No debe usarlo si no lo recibió previamente en el momento del three-way handshake (acuerdo de tres vías). De utilizarse, debería hacerlo mientras no se envía el ACK del número de secuencia más alto de los datos que están en su cola.

Quien recibe el ACK con la opción de SACK debe almacenar esta información. Se asume que el transmisor tiene almacenados de forma ordenada los segmentos que necesitará transmitir. Una posible implementación es que cada segmento almacenado en el transmisor tenga asociada una flag (bandera) SACKed que indique si dicho segmento ha sido incluido

en la opción SACK. Los segmentos que tengan dicha flag encendida "escaparán" a la retransmisión. Lo mismo ocurrirá con aquellos segmentos con número de secuencia mayor al máximo SACKed.

Luego de un RTO de retransmisión, se deberían apagar todas las flags, y se debe retransmitir el segmento más a la izquierda de la ventana. De esta forma se maneja la información para el envío completando los bloques que faltan en la transmisión, y marca los bloques como retransmitidos. Si un bloque retransmitido se pierde entonces pasa a la fase de Inicio Lento.

TCP SACK mejora TCP New Reno, pués permite el manejo de segmentos perdidos en una misma ventana, en una forma más eficiente, permitiendo su retransmisión en un solo RTT. En el caso de TCP New Reno, el protocolo se enterará de los segmentos perdidos a medida que reciba los que le preceden [21].

2.2.5 TCP VEGAS

TCP Vegas [26] tiene grandes cambios en comparación con las primeras versiones de TCP. No está estandarizado en ningún RFC. Utiliza tres técnicas para mejorar el rendimiento, y tratar de evitar la pérdida de segmentos.

La primera mejora es en el algoritmo de estimación del RTT, a partir de registrar el momento de envío de cada segmento y el momento de recibir cada ACK. A partir de esto, si recibe un ACK y la diferencia de tiempos con su segmento correspondiente es mayor al RTO, se retransmite sin esperar a tener los 3 ACK duplicados. Cuando recibe un ACK no duplicado, si es el primero o segundo luego de una retransmisión, calcula el tiempo que hace que envió el tercer segmento en cuestión y de ser mayor que el RTT lo envía nuevamente.

En segundo lugar TCP Vegas busca ser proactivo frente a la congestión y no reactivo como lo es Reno, que espera la pérdida de segmentos a los efectos de determinar el ancho de banda disponible en la red. Para ello calcula la razón de transferencia en la red, y la compara con la esperada. Esto le permite determinar el flujo de datos extra que puede transitar por la red y calcular el adecuado evitando llegar a la congestión. Esta es la fase Prevención de la Congestión. En TCP Vegas los emisores utilizan el RTT para controlar la

congestión, basándose en que cuanto mayor sea, se puede suponer que la red tiene mayor peligro de congestión.

Se define BaseRTT como el RTT de un paquete cuando el flujo no está congestionado, y se inicializa al del primer paquete enviado en la conexión. Puede verse como el mínimo de todos los RTTs.

La tasa esperada es:

$$\textbf{ExpectedRate = CongWindow/BaseRTT} \qquad \textbf{(2.9)}$$

Donde:

CongWindow: Ventana de Congestión.

Se calcula la tasa de envío real, *ActualRate* dividiendo el número de bytes que se transmiten en el tiempo de un RTT por dicho tiempo, y se ajusta la ventana en consecuencia.

Para ello se calcula:

$$\textbf{Diff = ExpectedRate - ActualRate} \qquad \textbf{(2.10)}$$

Donde:

Diff: siempre es positiva ó igual a 0, dado que **ActualRate > ExpectedRate** implica actualizar BaseRTT al último RTT.

Se definen dos límites, $\alpha < \beta$, de forma que:

- Si **(Diff $< \alpha$)** se incrementa la ventana de congestión linealmente durante el próximo RTT
- Si **(Diff > β)** se decrementa la ventana de congestion linealmente durante el próximo RTT,
- Si **($\alpha \leq$ Diff $\leq$ β)** no se cambia la ventana de congestión.

Finalmente, la tercera técnica, busca evitar la congestión en la fase Inicio Lento. Realiza crecimientos exponenciales de cwnd cada RTT alternados; entre ellos cwnd queda fija. Si la tasa de envíos lograda es menor que la esperada, durante un cierto lapso de tiempo, se pasa a un modo de crecimiento lineal y no exponencial [27].

2.2.6 OTRAS VERSIONES TCP.

Existen otras versiones de TCP que enumeramos y explicamos brevemente, ya que no las utilizaremos en nuestro trabajo:

El **TCP Net Reno (Network-sensitiveReno)** [28] es una mejora al protocolo TCP New Reno. Se buscó un conjunto de optimizaciones de TCP sensibilizarlo a la red y mejorar su rendimiento, reduciendo los RTO por retransmisión. En TCP Net Reno, un segmento es enviado para cada ACK duplicado recibido, antes que se dispare la fase de Retransmisión Rápida. Busca ser una mejora en el caso de ventanas de congestión pequeñas [29]. Supongamos que cwnd vale 3, si uno de los segmentos es descartado en la red, a lo sumo 2 ACK duplicados arribarán al transmisor (estos son generados cuando llegan segmentos fuera de orden), por lo tanto no se disparará la fase Retransmisión Rápida y el segmento perdido sólo se enviará nuevamente por RTO. Para ello se propone el algoritmo *Limited Transmit* (Transmision Limitada). Este permite al transmisor, luego de recibir 2 ACK duplicados consecutivos, a enviar hasta 2 segmentos más allá de lo indicado por cwnd y en caso que rwnd (ventana receptora) lo permita. El tamaño de cwnd no debe cambiar. Este algoritmo respeta la "conservación de los paquetes" y puede ser el que motive el tercer ACK duplicado lo que hará que se inicie la fase Retransmisión Rápida.

FAST TCP (Fast Active Queue Management Scalable Transmission Control Protocol), es un algoritmo de control de congestión diseñado por investigadores del Californian Institute of Technology. FAST TCP mide constantemente el tiempo que tardan en llegar los paquetes transmitidos, analizando retrasos y prediciendo la tasa de transferencia que podrá soportar la conexión para no producir pérdidas. Con este sistema, se han llegado a alcanzar en Internet velocidades de 100Gb/seg. En conexiones transatlánticas. Existen especulaciones sobre la capacidad que tendría dicho algoritmo de incrementar las velocidades de transferencia en Internet, sin necesidad de modificar la infraestructura actual [40, 41].

Es un algoritmo basado en ecuaciones, por tanto elimina las oscilaciones a nivel de paquete. Emplea retardo de cola como medida primaria de la congestión, lo cual es más fiable que medir la probabilidad de error. Tiene un comportamiento de dinámica de flujo estable y alcanza un equilibrio imparcial que no penaliza flujos largos.

El mecanismo de control de congestión de FAST TCP tiene cuatro componentes: data control (control de datos), windows control (control de ventanas), burstiness control

(control de ráfagas) y estimation (estimación). Todos ellos son funcionalmente independientes, de forma que pueden ser diseñados por separado e incluso mejorados por separado. A continuación se describe detalladamente cada componente:

Estimation: De la transmisión de cada paquete obtiene información de dos componentes: una sobre el retraso de colas (multi bit) y otra que indica si se ha perdido o no el paquete (1 solo bit).

Windows control: FAST TCP emplea el retardo de cola (como TCP Vegas) como parámetro principal en el ajuste de la ventana. Ajusta el tamaño de ventana de forma brusca, (aumentándolo o disminuyéndolo), cuando el número de paquetes en el buffer es muy lejano a su objetivo, y por otro lado, lo ajusta de forma fina, cuando el número de paquetes en buffer y su objetivo no difiere mucho. En este sentido, FAST es una versión de alta velocidad de TCP Vegas.

Data control**:** este componente selecciona cuál será el siguiente paquete a enviar, y lo hace de entre tres grupos candidatos posibles: 1. Los paquetes nuevos.2. Paquetes que están definitivamente perdidos (se ha recibido un ACK negativo).3. Paquetes de los que no se ha recibido confirmación.

Burstiness control: Este componente rastrea las transmisiones de paquetes de manera fluida para averiguar cual es el ancho de banda disponible. Esto es especialmente importante en redes de un gran ancho de banda con presencia de retardo de paquetes, donde el tráfico puede ser a ráfagas debido a eventos tanto en la red como en las WS (Works Station, estaciones de trabajo).

2.3 SIMULADOR OPNET MODELER.

En la actualidad existen diferentes simuladores, que permiten realizar el análisis del comportamiento de redes de computadoras de forma general ó específica, teniendo en cuenta algunas características. Muchos centros de investigación han diseñado los simuladores para su propio propósito y específicamente para la simulación de un protocolo o de un problema en particular. Además, no todos los simuladores tienen las mismas

ventajas y resaltar que tienen un papel importante dentro del estudio de las redes de computadoras [30].

El simulador **OPNET** fue lanzado en 1987 como la primera herramienta comercial de simulación disponible para las redes de comunicaciones.

OPNET es una herramienta de simulación de eventos discretos, basada en el lenguaje de programación C. Esta brinda una interfaz gráfica para los usuarios, la cual puede usarse en varios modos, facilitando el desarrollo de nuevos modelos y programas de simulación. Ofrece gran variedad de escenarios de simulación para el trabajo y es la parte central del proceso de análisis de resultados. OPNET ha sido diseñado para soportar el modelado y simulación de un gran rango de sistemas de comunicaciones, desde una simple LAN (*Local Area Network,* Red de Área Local) hasta una red global satelital.

Un modelo de OPNET consiste en tres capas: el Modelo de Red, el Modelo del Nodo y el Modelo de Proceso. El modelo de Red define la topología de red. El Modelo del Nodo determina el contenido de los nodos (transmisores, generadores, receptores, etc) y como están conectados los elementos. Algunos de estos elementos o módulos ya están predefinidos. No obstante, los usuarios pueden diseñar sus módulos. En el modelo de procesos está contenida la funcionalidad de un módulo, definida por una máquina de estado finito, las condiciones de transmisión y la ejecución de entrada y salida de datos, todas escritos en lenguaje de programación C [39].

En este capítulo se han analizados los mecanismos de TCP para hacer frente a las pérdidas de segmentos y a la prevención de dichas pérdidas. En las Redes IP se pierden segmentos, que perjudican en gran medida el rendimiento de la red, aunque dependiendo de la versión de TCP y del número de segmentos perdidos en una ventana, el impacto es mayor o menor.

CAPÍTULO 3. SIMULACIÓN Y ANÁLISIS DE LOS RESULTADOS.

El diseño de las grandes redes necesita la posibilidad de simulación y su mantenimiento implica la realización de pruebas. Para gestionar una red compleja y evitar la degradación del sistema y los tiempos de inactividad es importante poseer una visión de la estructura de la red. Un modelo detallado de la red puede contribuir a simplificar y aligerar la resolución de sus problemas. La simulación se encarga de mostrar el movimiento de tráfico ficticio y sus limitaciones. Resulta de gran importancia en las grandes redes, por la incapacidad del ser humano de poder mantener en su mente todo el conglomerado de la red. Se pueden identificar cuellos de botella, concentraciones y desbordamientos, así como definirse futuras ampliaciones de la misma.

OPNET (Optimum *Network Engineering Toll*, Herramienta de Ingeniería de Redes Optimizada) es una sofisticada herramienta para el diseño de redes, que permite establecer todos los detalles de una red, desde su topología o el tipo de conexiones entre los nodos hasta la estructura de los procesos que se ejecutan en cada nodo de la red. Permite ejecutar simulaciones y tomar gran cantidad de datos estadísticos tanto de la red, en conjunto, como de cada elemento por separado.

Las simulaciones realizadas en este capítulo muestran el comportamiento gráfico de las diferentes implementaciones TCP, con el objetivo de ser comparadas y analizar sus principales dificultades y posibles mejoras.

3.1 DISEÑO DE LOS ESCENARIOS.

El escenario básico utilizado para mostrar los resultados de la simulación, fue diseñado en el Editor de Proyectos de la herramienta OPNET Modeler. Inicialmente se creó un proyecto, con un escenario de área (Y: 1.0 Km. X: 1.5 Km.). Luego se prosiguió con la creación y configuración del modelo de red. En el espacio de trabajo del proyecto se añadieron los siguientes objetos: Application Config (Donde se definen y configuran las aplicaciones utilizadas), Profile Config (Donde se definen los perfiles de las aplicaciones) e

ip32_Cloud, este nodo modela una nube IP con soporte de hasta 32 interfaces serie a una tasa seleccionable mediante la que se puede modelar tráfico IP. Los paquetes IP que llegan a cualquier interfaz se encaminan a la de salida adecuada, basándose en la dirección IP destino. El protocolo RIP u OSPF puede ser usado para crear de manera automática y dinámica las tablas de encaminamiento y seleccionar las rutas de manera adaptativa. Esta nube requiere una cantidad fija de tiempo para encaminar cada paquete, determinada por el atributo Packet Latency del nodo; también existen dos subredes como se muestra en la Fig. 3.1.

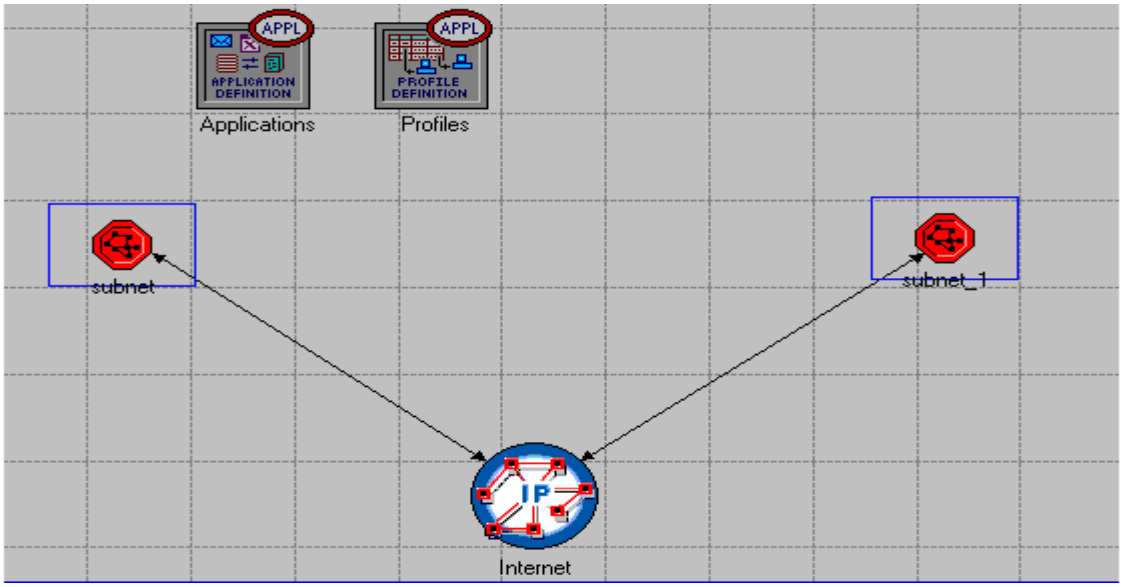


Fig. 3.1 Escenario Básico.

En la subred nombrada subnet existe en su interior una estación de trabajo conectado a un router y en la otra subred (subnet1) está compuesta por un servidor conectado a un router esto se puede observar en los Anexos 2 y 3. Dichos routers se conectan a la nube IP mediante enlaces bidireccionales PPP_DS1 con una velocidad de transferencia de 1.544 Mbps.

La aplicación que se utiliza en la simulación es FTP (Protocolo de Transferencia de Archivos) para ello se configuró dicha aplicación con un tamaño de fichero de 3.0 MBytes como se muestra en la Fig. 3.2.

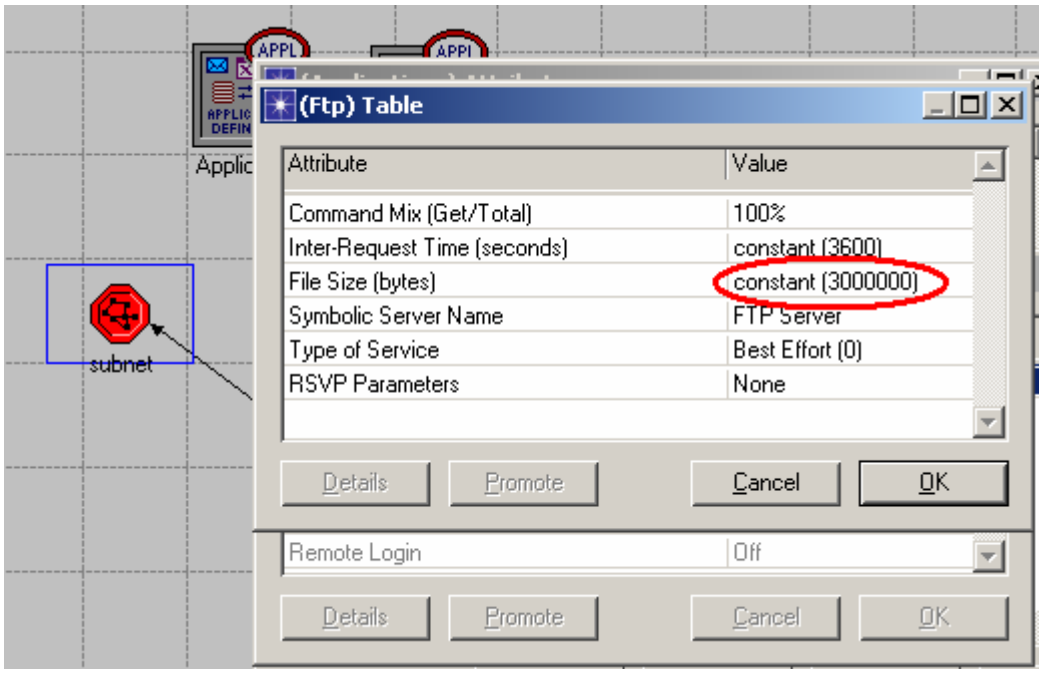


Fig. 3.2 Configuración de la Aplicación FTP.

El servidor y la estación de trabajo también se configuran, los parámetros TCP es un punto clave a la hora de tratar la simulación y analizar las consecuencias de la congestión. Como se muestra en la Fig. 3.3 el Opnet Modeler te facilita la opción de escoger la versión TCP a implantar en la red y configurarla habilitando los mecanismos de control de la congestión que posean dichas implementaciones TCP. Ver Anexo 4.

Como se puede observar en esta figura 3.3 la versión de la herramienta de simulación (Opnet Modeler 8.0) no posee todas las implementaciones TCP explicadas anteriormente, debido a este inconveniente fueron simuladas: TCP TAHOE, TCP TAHOE con el algoritmo de Retransmisión Rápida habilitado, TCP RENO y TCP SACK.

Ya configurada la red, puede ser simulada para un análisis de los resultados. Existen numerosas opciones para seleccionar las estadísticas que se desean visualizar. Lo más importante para comprender los mecanismos de control de la congestión, es analizar el comportamiento del tamaño de la ventana de congestión (Congestion Windows Size).

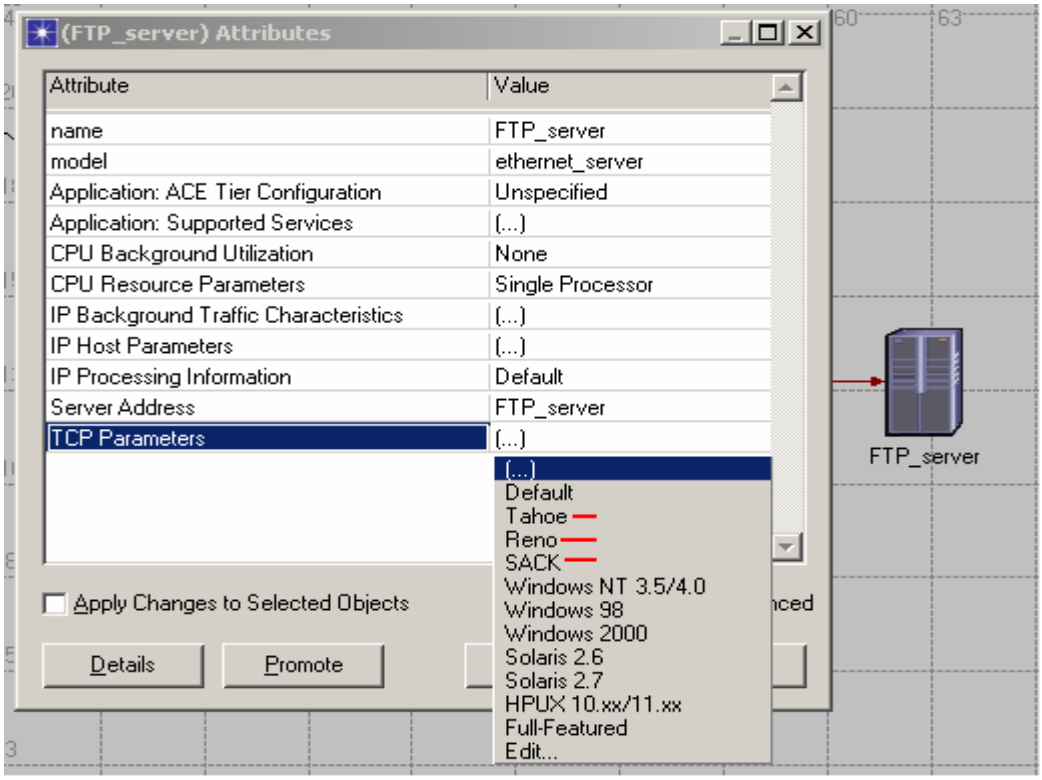


Fig. 3.3 Implementaciones TCP a utilizar.

3.2 ANÁLISIS DE LOS RESULTADOS.

Luego de ser visualizados los resultados obtenidos en la simulación, se analiza el comportamiento de las diferentes implementaciones TCP en orden cronológico, observando las mejoras que se incluyen en cada una de ellas.

3.2.1 Análisis de TCP Tahoe.

En la Fig. 3.4 se muestra la primera implementación TCP simulada, como se observa, no se encuentra habilitado el mecanismo de Retransmisión Rápida, al no recibir reconocimientos duplicados, se activa un temporizador cuando existe una pérdida de paquetes.

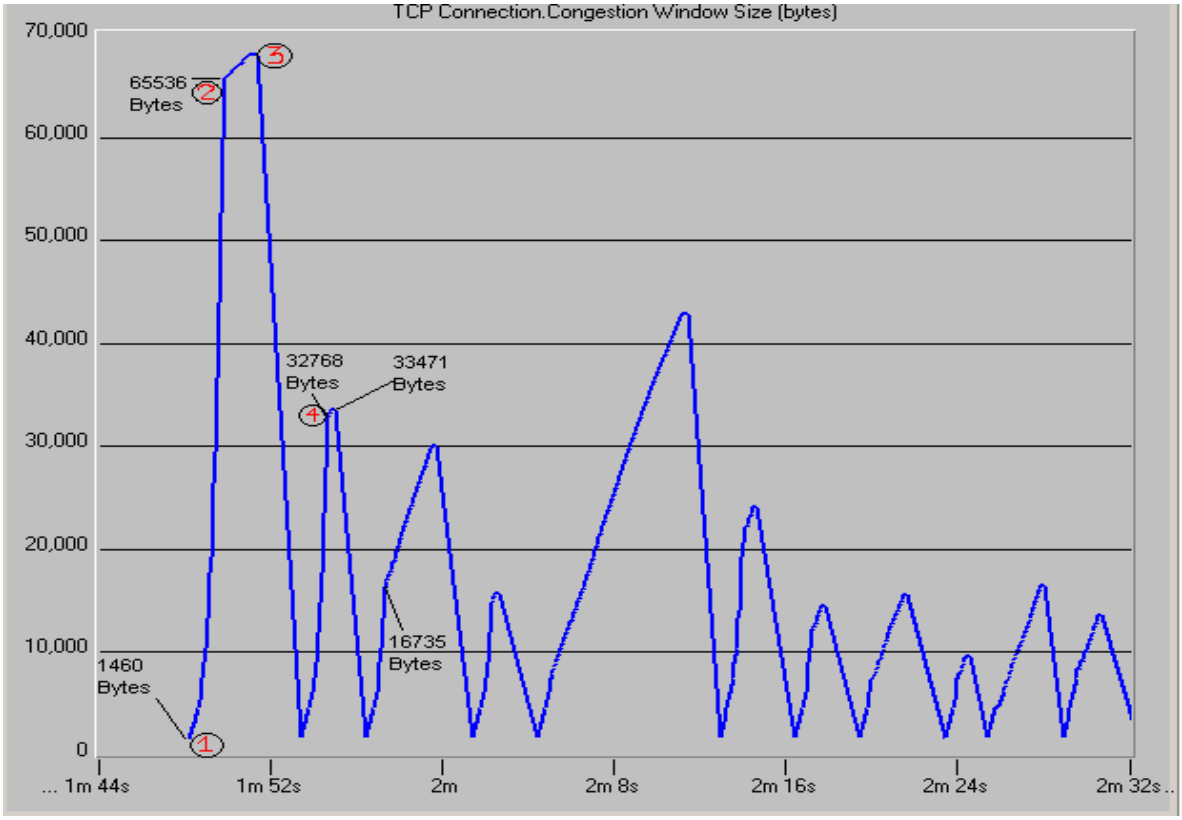


Fig. 3.4 TCP Tahoe (Inicio Lento + Prevención de la Congestión).

Se inicia la ventana de congestión (cwnd) en un SMSS (1460 bytes, punto 1) debido a que la red física que utilizamos es Ethernet, el valor umbral ssthresh inicialmente se fija en el tamaño máximo de la ventana de transmisión (65536 bytes, punto 2) este valor puede variar según la configuración que se desee. Al ocurrir un RTO (punto 3) comienza nuevamente el mecanismo de Inicio Lento (crecimiento exponencial) hasta que cwnd se iguale a ssthresh y continúe con el mecanismo de Prevención de la Congestión con el fin de no saturar la red (punto 4). Si no ocurren más terminaciones de temporización, la ventana de congestión continuará creciendo hasta el tamaño de la ventana del receptor. En ese punto dejara de crecer y permanecerá constante mientras no ocurra más RTO y la ventana del receptor no cambie de tamaño. La desventaja de TCP Tahoe es que toma un intervalo completo RTO para detectar una pérdida de paquete. Cada vez que se pierde un paquete se vacía el buffer. Esto ofrece un alto costo en el ancho de banda y retardo en las transmisiones [34].

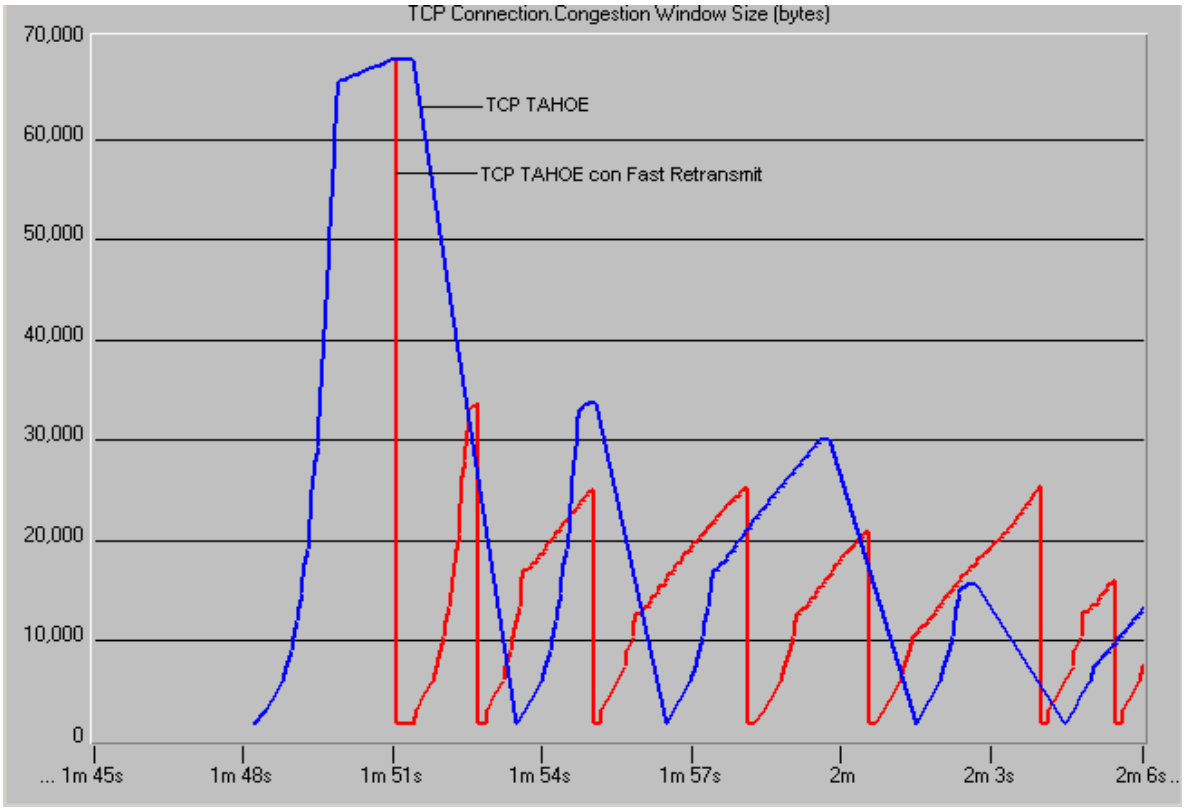


Fig. 3.5 TCP Tahoe y TCP Tahoe + Retransmisión Rápida.

Se puede observar como la mejora de esta versión influye en el rendimiento del protocolo, el surgimiento del mecanismo Retransmisión Rápida incluye los reconocimientos duplicados de esta forma se activa la retransmisión de un segmento perdido antes de que expire su temporizador de retransmisión (RTO) [20]. La incorporación de este mecanismo en TCP Tahoe permite optimizar y ajustar los algoritmos Inicio Lento y Prevención de la Congestión para adecuar el protocolo TCP a situaciones de congestión y pérdidas en general, y ayudarlo así a recuperarse con mayor rapidez de las mismas [21, 35].

3.2.2 Análisis de TCP Reno.

En la Fig. 3.6 se observa el comportamiento del tamaño de la ventana de congestión de TCP Reno el cual es una mejora de TCP Tahoe. TCP Reno continuó usando las mejoras incorporadas en TCP Tahoe, así mismo modificó la fase de Retransmisión Rápida para adicionar el algoritmo de Recuperación Rápida. Se aprecia que con esta mejora después de la pérdida de un paquete no se comienza con Inicio Lento, permitiendo que no se vacíe el

buffer provocando esto, un mejor rendimiento, permitiendo que se recupere con mayor rapidez de una situación de congestión y errores en la red.

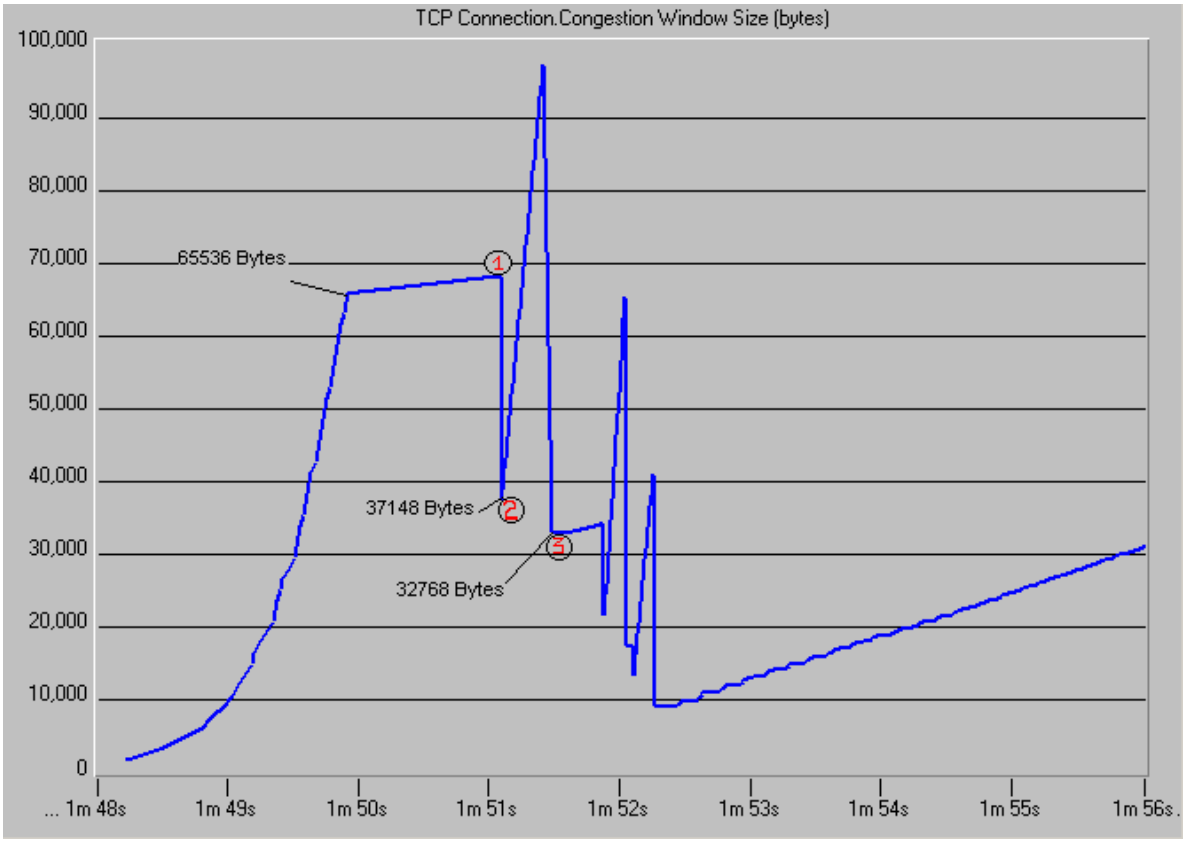


Fig. 3.6 TCP Reno (TCP Tahoe mejorado + Recuperación Rápida).

En presencia de una perdida de paquete (punto 1), cwnd toma el valor de: **ssthresh + 3*SMSS** (punto 2), si ssthresh = 32768 bytes, entonces cwnd = 32768 + 3*1460 bytes ver ecuación (2.7), como muestra la Fig.3.6 cuando llega un ACK simple, que reconoce nuevos datos, debe comprender el segmento perdido y todos los enviados después de este y hasta la recepción del tercer ACK duplicado, entonces se establece el valor de cwnd al valor de ssthresh (32768 bytes, punto 3), mostrando un desinflado de cwnd y finalizando el algoritmo de Recuperación Rápida.

TCP Reno muestra buen rendimiento cuando existe solamente una pérdida de paquete por ventana, si existieran múltiples pérdidas por ventana (pérdidas en sucesivos segmentos) el rendimiento de TCP Reno fuera peor que TCP Tahoe [36, 37], si existen múltiples pérdidas de paquete, entonces la información sobre la pérdida del primero aparece cuando se reciben ACK's duplicados. Pero la información sobre el segundo paquete que fue perdido aparecerá solamente después que el ACK para el primer segmento retransmitido alcanza al emisor

después de un RTT. Estas pérdidas son comunes en enlaces de alta velocidad, donde la pérdida en general contiene varios segmentos [21].

Todas estas simulaciones se realizaron para una pérdida de paquete por ventana, para múltiples pérdidas el comportamiento de TCP Reno fuera diferente y el comportamiento de TCP Tahoe y TCP SACK no varía mucho en este aspecto (Anexos 5, 6, 7).

3.2.3 Análisis de TCP SACK.

TCP SACK también utiliza los algoritmos de Retransmisión y Recuperación Rápida, al igual que TCP Reno. La ventaja de esta versión se halla en que el receptor informa al transmisor de los segmentos que ha recibido correctamente, para que la retransmisión sólo se lleve a cabo para aquellos segmentos que no han sido reconocidos. De este modo, el número de segmentos retransmitidos en la red se reduce significativamente [38].

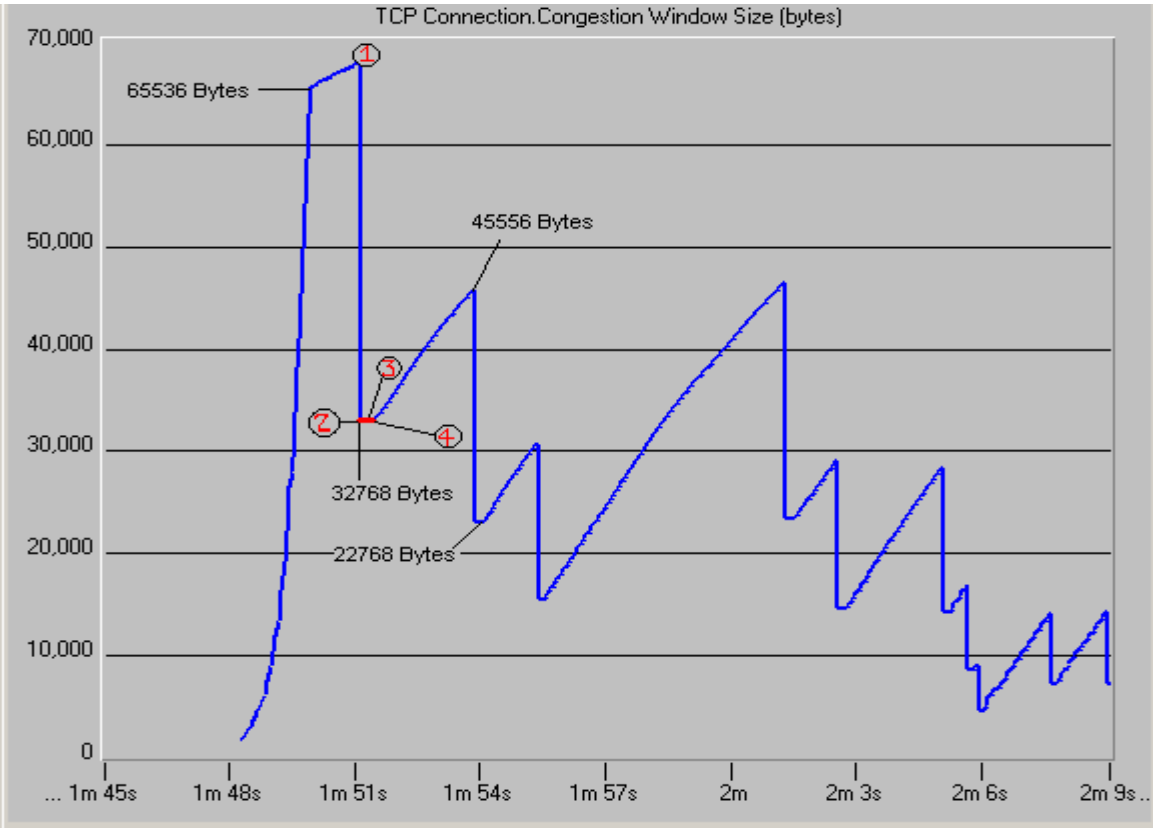


Fig. 3.7 TCP SACK (Selective Acknowledgment, Reconocimientos Selectivos).

Tras la pérdida de un segmento del flujo de datos (punto 1), cuando ocurre la detección de los tres ACK's duplicados, se reduce la ventana a la mitad del número de segmentos en tránsito (punto 2). A medida que llegan los ACK's duplicados con información de

reconocimientos selectivos (SACK), la variable pipe (indica el número de segmentos que se han enviado, pero no han recibido reconocimientos) se va actualizando, pero la ventana de congestión permanece constante (punto 3). Con la información de los reconocimientos selectivos, se retransmiten los segmentos perdidos. Cuando llega el ACK que confirma la recepción de todos los segmentos, se continúa con la fase de Prevención de la Congestión (punto 4). La desventaja de TCP SACK constituye la difícil incorporación de la opción SACK en el protocolo TCP actual, se necesitan campos para reconocer los segmentos selectivos y requiere cambios en el receptor, mientras que todas las implementaciones antes mencionadas solamente requieren los cambios en el lado del emisor.

Investigaciones futuras sobre algoritmos de control de congestión pueden sacar provecho de la información adicional que provee SACK. Una posible área de estudio de este tipo es la modificación del protocolo TCP para entornos inalámbricos o por satélite donde la pérdida de paquetes no es necesariamente una indicación de congestión [24].

Como se ha podido observar todas estas implementaciones utilizan el mecanismo Inicio Lento al comienzo de cada conexión permitiendo el incremento gradual de la cantidad de datos en tránsito y es necesario para conseguir llevar la conexión al estado de equilibrio, seguido se utiliza la fase de Prevención de la Congestión para evitar la saturación de la red.

3.2.4 Otros Resultados.

Luego de analizadas las implementaciones TCP Tahoe, Reno y SACK, se proponen algunas estadísticas como son: *Segment Round Trip Time* (medida en segundos del tiempo de ida y vuelta (RTT) del segmento para una conexión). En la Fig. 3.8 se observa el RTT promedio para todas las implementaciones. Se observa como TCP SACK posee el menor RTT y TCP Tahoe presenta el mayor tiempo de ida y vuelta del paquete, este parámetro influye en el rendimiento de la red.

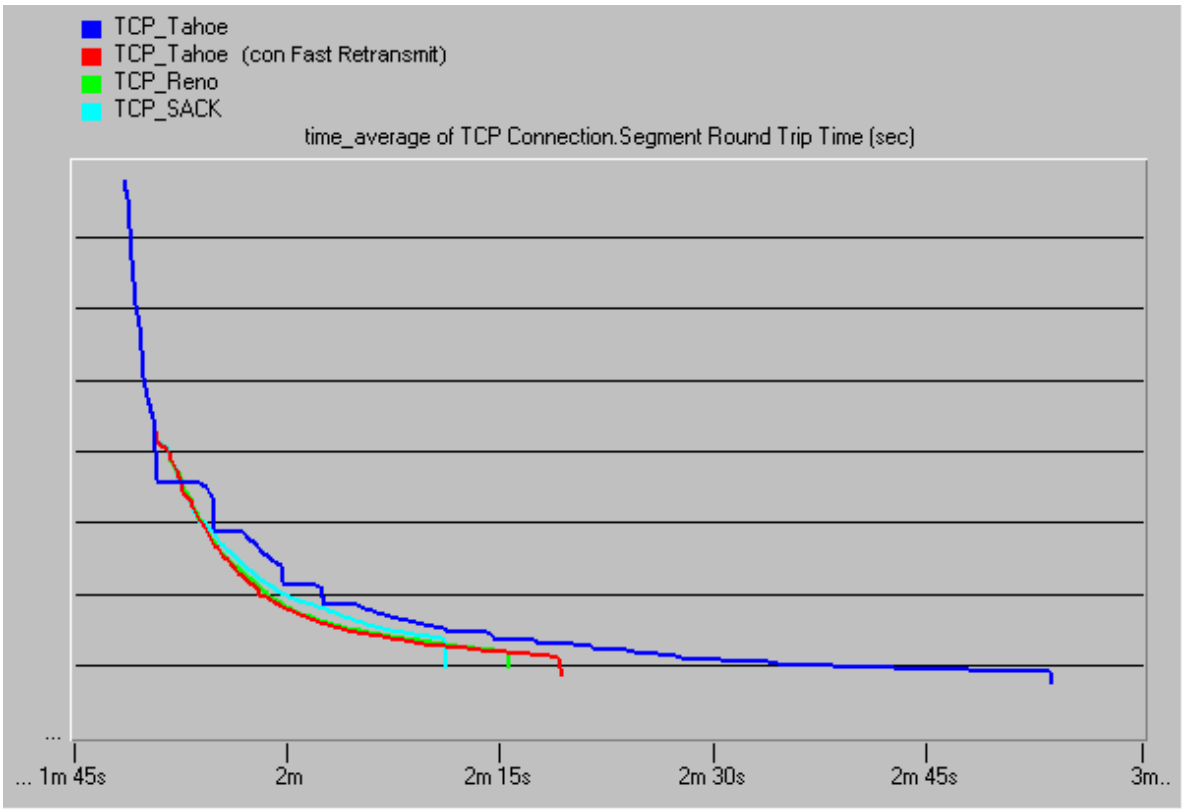


Fig. 3.8 Tiempo promedio de ida y vuelta del segmento.

Otro resultado importante es el (Download Response Time) Tiempo de Respuesta FTP en la estación del cliente, que muestra el tiempo transcurrido entre el envío de una solicitud y la recepción de la respuesta del paquete por la aplicación FTP en este nodo, medido a partir del tiempo en que la aplicación del cliente envía una solicitud al servidor hasta el tiempo en que recibe una respuesta del paquete. En la Fig. 3.9 se observa que TCP SACK posee el menor tiempo de respuesta, mientras que TCP Tahoe lo supera para este caso. Para el caso de pérdidas consecutivas de segmentos, TCP Reno es más lento que TCP Tahoe, lo que constituye una contradicción pues Reno fue concebido como una mejora con respecto a Tahoe, esto se debe a que Reno está optimizado para redes con una pequeña proporción de descartes de paquetes. Se ha demostrado que TCP SACK es la versión de TCP que mejor se adapta las redes IP. Ver Anexo 8.

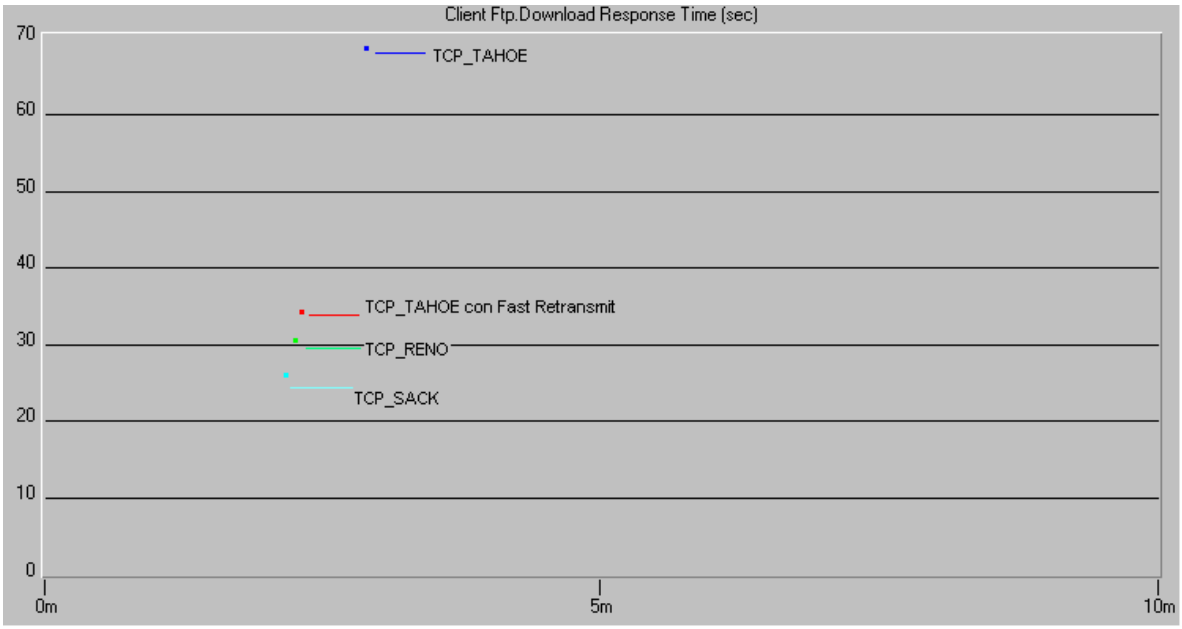


Fig. 3.9 Tiempo de Respuesta.

TCP Tahoe es el que presenta más deficiencias ante las pérdidas de paquetes, mientras que TCP SACK puede detectar y retransmitir el paquete perdido con mayor rapidez que los time out de TCP Tahoe y TCP Reno, también cuenta con un menor número de retransmisiones, ya que, no tiene que vaciar el buffer cada vez que ocurra una pérdida de paquete. TCP SACK es superior que TCP Reno porque permite el manejo de segmentos perdidos consecutivamente en una misma ventana. Para una pérdida de paquete por ventana, TCP Reno presenta mejor comportamiento que TCP Tahoe, aplica la fase de Recuperación Rápida que permite la recuperación de segmentos en sustitución del inicio lento exponencial.

TCP New-Reno, TCP Vegas, aunque no pudieron ser simuladas por incapacidad de la versión de la herramienta utilizada se hace importante destacar que se ha demostrado que TCP New-Reno es mucho más eficiente que TCP Reno, ya que detecta múltiples pérdidas de paquetes por ventana, pero a la vez TCP SACK es mas eficiente que New-Reno, porque permite la retransmisión de paquetes perdidos consecutivamente en un solo tiempo de ida y vuelta (RTT). Según investigaciones y comparaciones de estas implementaciones se ha demostrado que TCP Vegas es el más efectivo entre los mencionados, porque puede detectar y retrasmitir el paquete perdido mucho mas rápido que los RTO de TCP Tahoe, también tiene pocas retransmisiones debido a que no tiene que vaciar el buffer siempre que

exista una perdida de paquete, es el mejor en la prevención de la congestion, la modificación de los algoritmos Inicio Lento y Prevención de la Congestión permite la medida exacta del ancho de banda disponible, por lo tanto utiliza recursos de la red eficientemente. TCP Vegas supera a TCP Reno también, porque no tiene que esperar 3 ACK duplicados, así ocurre la retransmisión mucho más rápido, con respecto a TCP SACK hay mejoras en cuanto a la eficiente estimación de la congestión por medir el cambio en el rendimiento en lugar de pérdidas de paquetes, lo que da lugar a una mejor utilización del ancho de banda y contribuye a minimizar la congestión.

Este tema requiere de estudios investigativos más profundos, con el estudio queda demostrado que para las Redes IP la versión TCP SACK es la más efectiva. En la actualidad existen versiones de esta herramienta que permiten simular estos escenarios mostrando mayores estadísticas y agregando variantes TCP como lo es New-Reno, logrando una mejor comprensión del funcionamiento de las implementaciones que no fueron simuladas.

CONCLUSIONES

Como resultado de la investigación se concluye que:

1. La construcción del marco teórico en esta temática derivado de la consulta de literatura internacional y nacional más actualizada, puede ser utilizado como documento de referencia con fines docentes, metodológicos e investigativos.
2. TCP seguirá siendo a largo plazo el protocolo de transporte dominante para las redes IP, porque permite una comunicación fiable de datos, aplicando métodos para el control de flujo y control de la congestión como son: Inicio Lento con Prevención de la Congestión y Retransmisión Rápida con Recuperación Rápida
3. Con los resultados obtenidos de la simulación, se ha demostrado que TCP SACK es la versión que mejor se adapta a las redes IP en cuanto al rendimiento y al retardo, pues permite la retransmisión de paquetes perdidos consecutivamente en un mismo RTT.
4. El OPNET Modeler 8.0 cumple con las expectativas necesarias para investigaciones sobre temas de congestión en redes IP, pues constituye una herramienta de modelación y simulación que permite graficar su comportamiento.
5. Existen implementaciones TCP, que no fueron simuladas, por incapacidad de la versión de la herramienta utilizada. El análisis teórico de TCP Vegas, demostró que supera todas las implementaciones estudiadas.

.

RECOMENDACIONES

1. Profundizar en el estudio de la herramienta OPNET Modeler y utilizar eficientemente las potencialidades que este brinda.
2. Indagar en las implementaciones TCP que no se simularon y continuar investigaciones sobre nuevas versiones que surjan en el mundo de la Telemática.
3. Abordar esta temática con mayor profundidad en las asignaturas de Telemática de la carrera de Ingeniería de Telecomunicaciones y Electrónica.

GLOSARIO

IP. (Internet Protocol). Protocolo Internet.

TCP. (Transmission Control Protocol). Protocolo de Control de la Transmisión.

UDP. (User Data Protocol). Protocolo de Datos de Usuario.

QoS. (Quality of Service). Calidad de Servicio.

TPDU. (Transport Protocol Data Unit). Unidad de Datos del Protocolo de Transporte.

TSAP. (Transport Service Access Point). Punto de acceso al servicio de transporte.

TLI. (Transport Layer Interface). Interfaz de capa de transporte.

RFC (Request for Comment). Solicitud de Comentarios.

API. (Applications Programming Interface). Interfaz de Programación de Aplicaciones.

ACK. (Acknowledgement). Reconocimiento.

W. (Windows). Ventana

BIT. (Binary Digit). Dígito binario.

BYTE. Un conjunto de ocho bits, considerados como una unidad de almacenamiento de la información.

LAN. (Local Area Network). Red de Área Local.

MSS. (Maximum Segment Size). Tamaño Máximo de Segmento.

SMSS (Sender Maximum Segment Size). Tamaño máximo del segmento que el emisor puede transmitir

MTU. (Maximum Transfer Unit). Unidad de Transferencia Máxima.

SYN. (Synchronous) Sincronización de los números de secuencia.

TCP/IP. (Transmission Control Protocol/Internet Protocol). Protocolo de Control de Transmisión/Protocolo Internet.

WS. (Work Station). Estación de Trabajo.

Hosts. Computadoras.

SACK. (Selective Acknowledgment). Reconocimientos Selectivos.

BSD. (Berkeley Software Distribution). Distribución de Software Berkeley.

RTT. (Round Trip Time). Tiempo de ida y vuelta de un segmento.

RTTM (Round Trip Time Measurement). Mediciones del tiempo de ida y vuelta.

RTO.(Retransmission Time Out). Temporizador de Retransmisión.

FTP. (File Transfer Protocol). Protocolo de Transferencia de Archivos.

OPNET. (Optimum Network Engineering Toll). Herramienta de Ingeniería de Redes Optimizada.

REFERENCIAS BIBLIOGRÁFICAS

1] Andrew S. Tanenbaum, "Computer Networks", 4th Ed, Prentice Hall, Upper Saddle River, New Jersey, Capítulo 6, págs 479-545, 2003.

[2] Dr. Félix Alvarez Paliza, "Arquitectura TCP\IP Nivel de Transporte", Departamento de Telecomunicaciones, UCLV, 2007

[3] R. Stevens,"TCP/IP Illustrated, Volume 1: TheProtocols", Addison-Wesley, 1994.

[4] J. Postel, "Transmission Control Protocol", STD 7, RFC-793, September 1981.

[5] J. Postel, "User Datagram Protocol", RFC-768, August 1980.

[6] Braden R., "Requirements for Internet Hosts --Communication Layers", STD 3, RFC 1122, October 1989.

[7] Jacobson V., Braden R., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

[8] Reynolds, J., and J. Postel, "Assigned Numbers", RFC 1700, October 1994.

[9] Farrel, Adrian, “The Internet and its Protocols a Comparative Approach”, Capitulo 7 págs. 307-364.

[10] Microsoft Corporation, "Microsoft Windows Server 2003 TCP/IP Implementation Details", Published: June 2003, Updated: December 2007

[11] Clark, D., "Window and Acknowledgement Strategy in TCP", RFC 813, July 1982.

[12] J. Nagle, "Congestion Control in IP/TCP", RFC 896, January 1984.

[13] Jacobson, V., "Congestion Avoidance and Control", ACM SIGCOMM-88, August 1988.

[14] P. Karn and C. Partridge "Round Trip Time Estimation", ACM SIGCOMM-87, August 1987.

[15] Braden, R., "Requirements for Internet Hosts -- Communication Layers", STD 3, RFC-1122, October 1989.

[16] Jacobson, V., "Modified TCP Congestion Avoidance Algorithm", end2end-interest mailing list, April 30, 1990. ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail.

[17] Allman, M., Paxson, V. and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[18] Sally Floyd., "Congestion Control Principles". RFC 2914, Network Working Group, Best Current Practice., September 2000.

[19] Stevens, W. Richard; Wright, Gary R., TCP/IP Illustrated: The Implementation, Vol.2, Pearson Education, (1995)

[20] Moraru, Bogdan; Copaciu, Flavius; Lazar, Gabriel; Dobrota, Virgil (2002): "Practical Analysis of TCP Implementations: Tahoe, Reno, New-Reno",

[21] Fall, K. y S. Floyd, "Comparisons of Tahoe, Reno, and Sack TCP", December 1995.

[22] Floyd, S. and T. Henderson, "The New Reno Modification to TCP's Fast Recovery Algorithm", RFC 2582, April 1999.

[23] Kevin Fall and Sally Floyd, “Simulation-based Comparisons of Tahoe, Reno and SACK TCP”, Computer Communication Review, July 1996.

[24] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, “TCP Selective Acknowledgment Options”, RFC 2018, October 1996.

[25] M. Mathis, J. Mahdavi, S. Floyd and M. Podolsky, “An Extension to the Selective Acknowlegdment (SACK) Option for TCP”, RFC 2883, July 2000.

[26] Chunlei Liu, Wireless Network Enhancement L. Brakmo, S. O’Malley and L. Peterson, “TCP Vegas: New Techniques for Congestion Detection and Avoidance”, 1994

[27] Lawrence S. Brakmo and Larry L. Peterson., “TCP Vegas: end to end congestion avoidance on a global Internet”. IEEE Journal on Selected Areas in Communications, October 1995.

[28] Dong Ling and H.T.Kung, “TCP Fast Recovery Strategies: Analysis and Improvements”, 1998

[29] M. Allman, H. Balakrishnan and S. Floyd, “Enhancing TCP's Loss Recovery Using Limited Transmit Computting TCP’s Retransmission Timer”, RFC 3042, January 2001.

[30] Nielsen//NetRatings Inc., Notas de prensa, 2001. Accesibles en http://209.249.142.22/press_releases.asp?country=north+america.

[31] Chunlei Liu, "Wireless Network Enhancement Using Congestion Coherence, Faster Congestion Feedback, Media Access Control and AAL2 Voice Trunking – Dissertation", The Ohio State University, 2001.

[32] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC 2001, January 1997.

[33] M. Hassan y R. Jain, "High Performance TCP/IP Networking: Concepts, Issues, and Solutions", Prentice-Hall, 2003.

[34] B. Sikdar, S. Kalyanaraman and K. S. Vastola, "Analytic Models and Comparative Study of the Latency and Steady-State Throughput of TCP Tahoe, Reno and SACK", Dept. of ECSE, Rensselaer Polytechnic Institute Troy, NY 12180 USA, 2007.

[35] A. Kumar, "Comparative performance analysis of versions of TCP in a local network with a lossy link", IEEE/ACM Trans. on Networking, vol. 6, no. 4, Aug 1997.

[36] O. González de Dios, I. de Miguel, V. López Álvarez, R. J. Durán, N. Merayo, J. F. Lobo Poyo, "Estudio y simulación de TCP en redes de conmutación óptica de ráfagas (OBS)", XV Jornadas Telecom I+D, Madrid, November 2005.

[37] M. N. Akhtar, M. A. O. Barry and H. S. Al –Raweshidy, "Modified Tahoe TCP for Wireless Networks Using OPNET Simulator", Department of Electronic Engineering, University of Kent, U.K., November 2006.

[38] G. Corral and A. Zaballos, "Simulation-based study of TCP flow control mechanisms using OPNET Modeler", 2006.

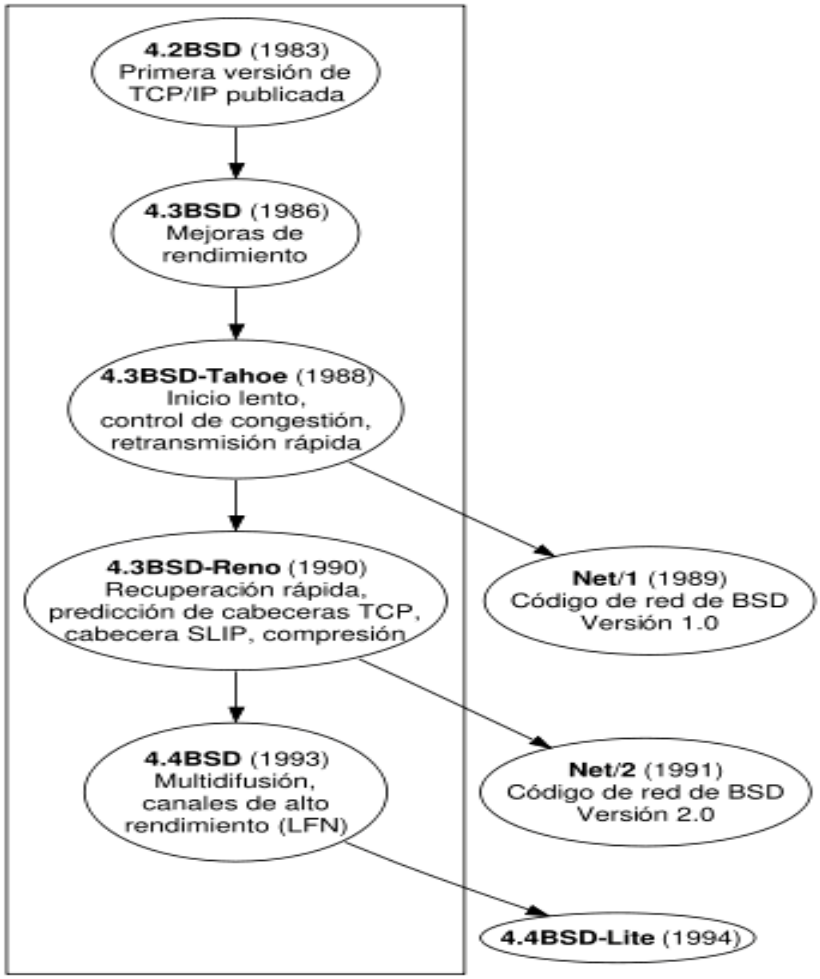
[39] OPNET Technologies, Inc., "Intro_Modeler_8.0_Lab_Manual.doc", disponible con la distribución del software, (2001).

[40] Hyojeong Choe and Steven H. Low. Stabilized Vegas. In Proc. of IEEE Infocom, April 2003. http://netlab.caltech.edu.

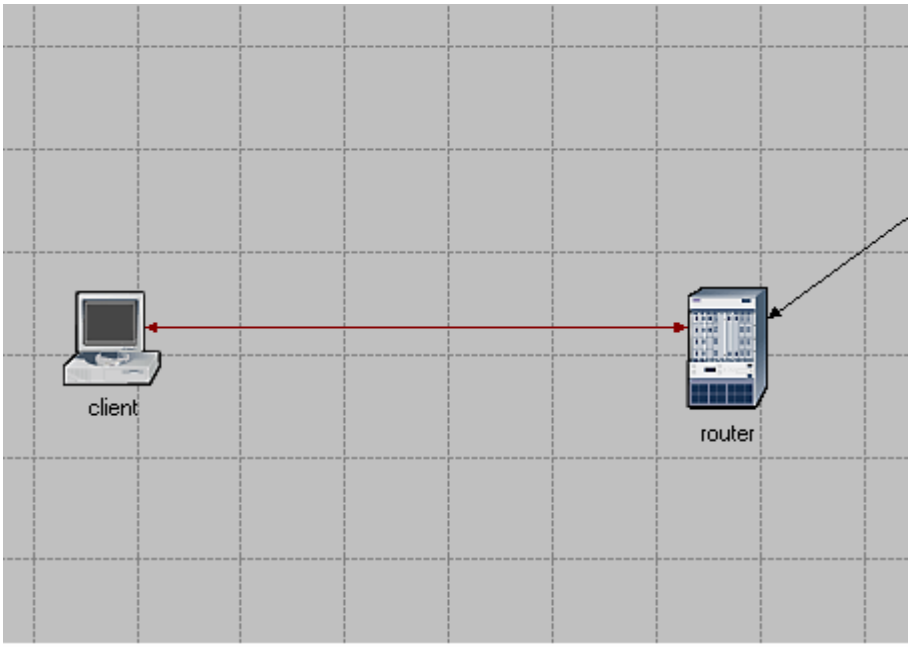
[41] Cheng Jin, David X. Wei, and Steven H. Low. TCP FAST: motivation, architecture, algorithms, performance. In Proceedings of IEEE Infocom, March 2004. http://netlab.caltech.edu.

ANEXOS

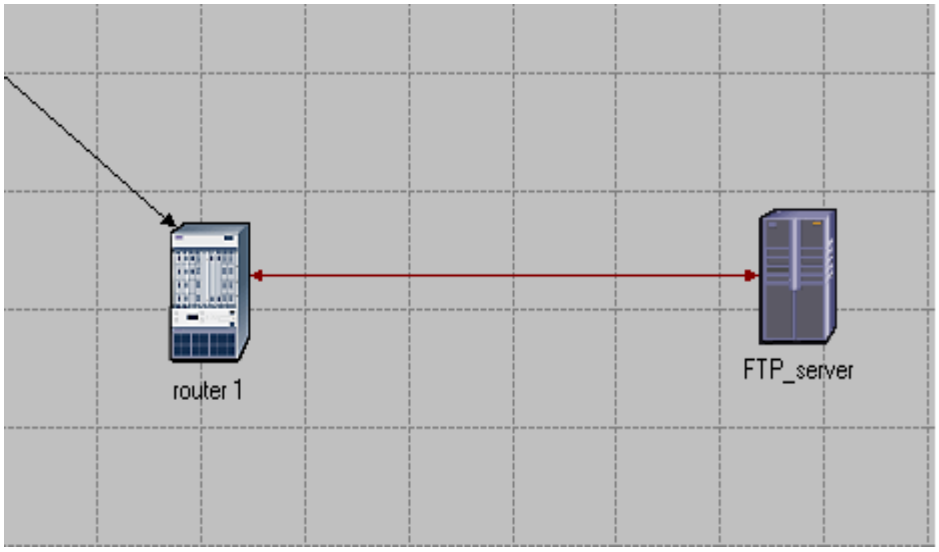
ANEXO 1: Algunas versiones de BSD, y características TCP ofrecidas.



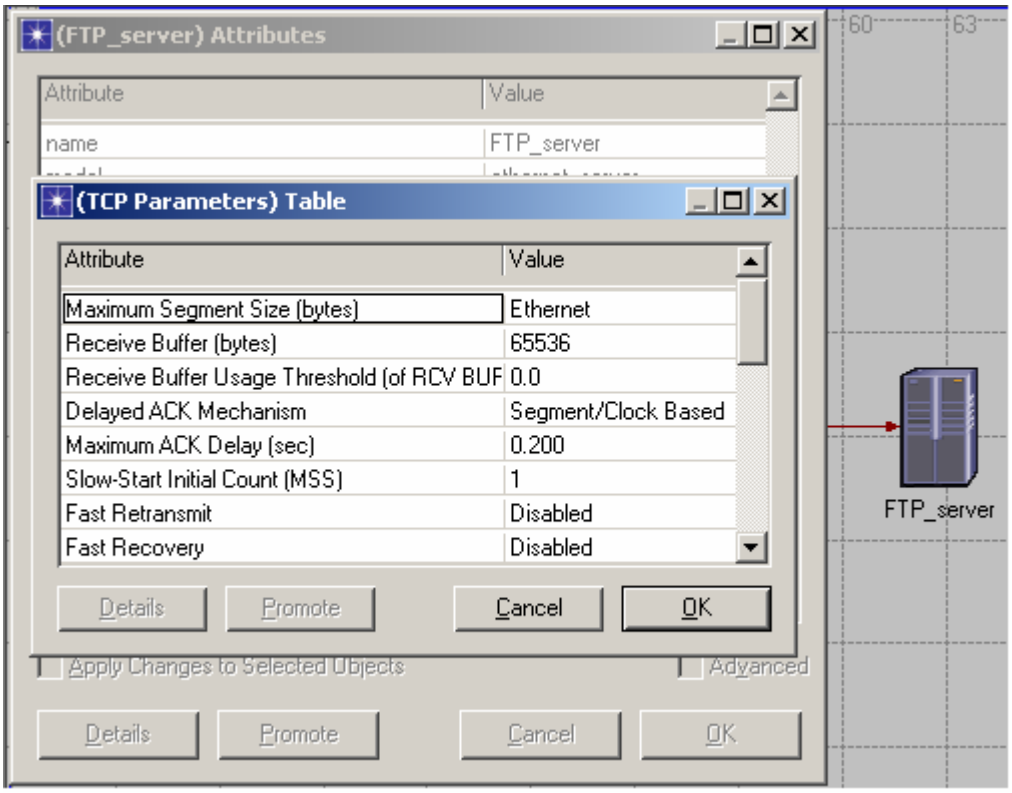
ANEXO 2: SUBRED



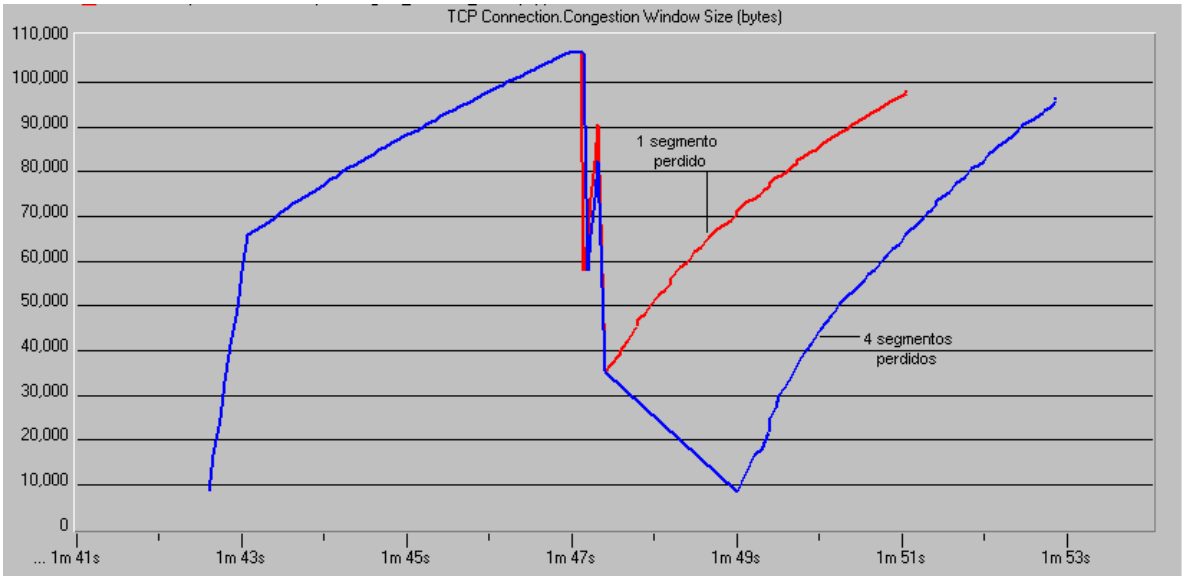
ANEXO 3: SUBRED 1.



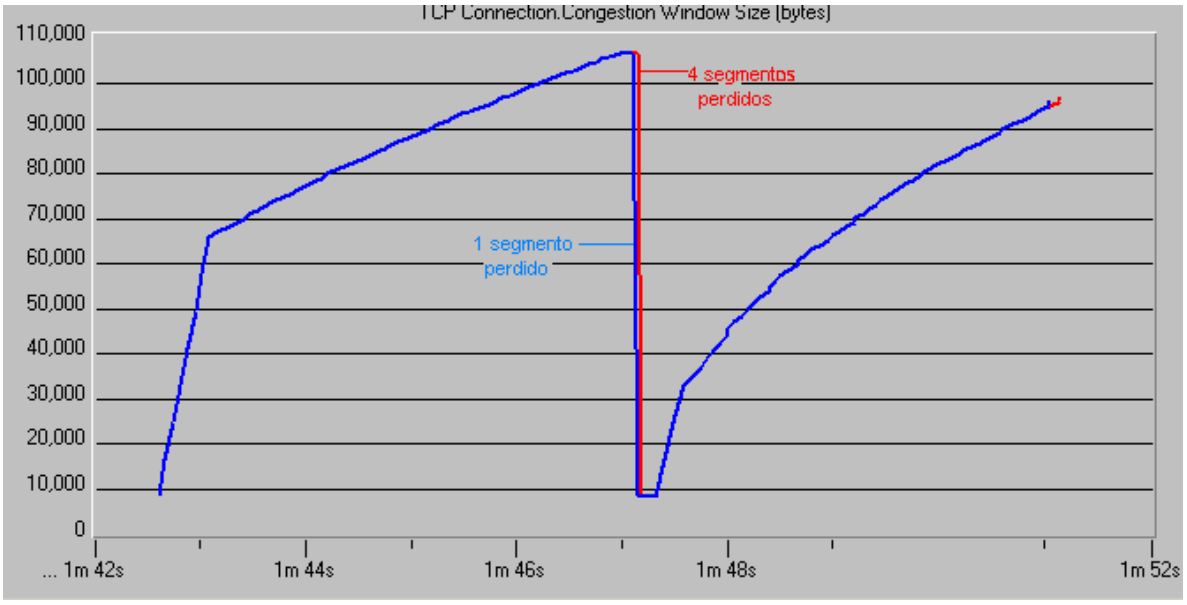
ANEXO 4: Configuración de los Parámetros TCP.



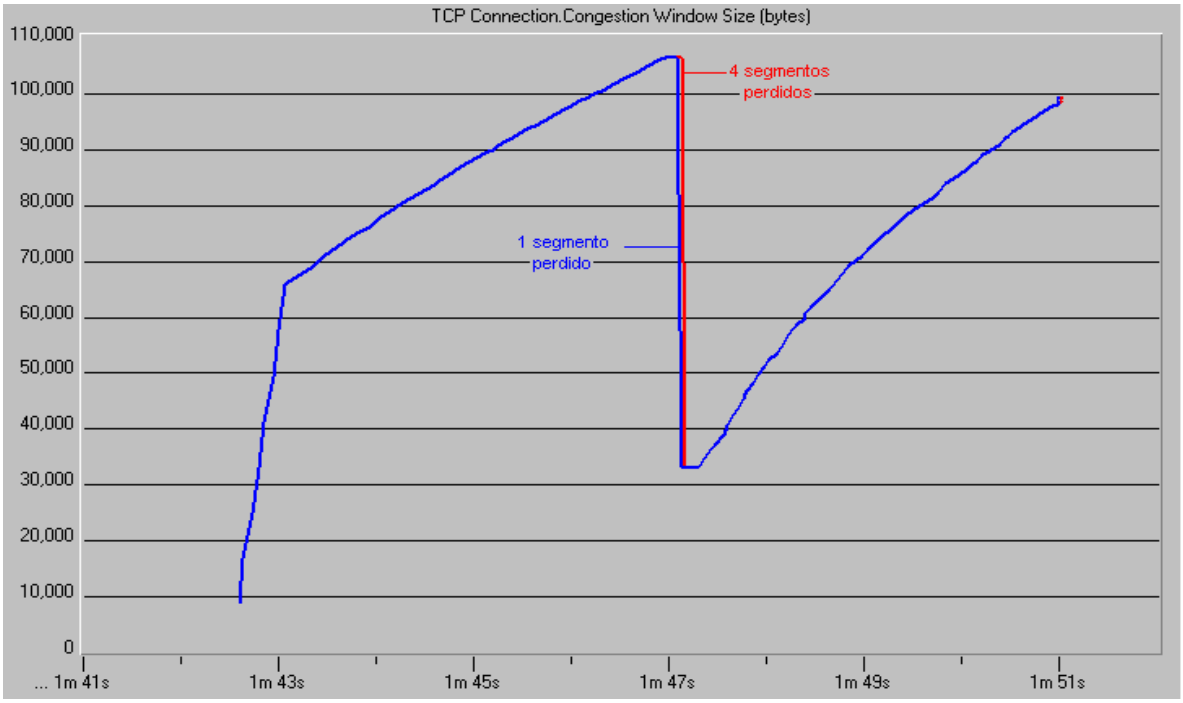
ANEXO 5: Comparación de la Ventana de Congestion en TCP Reno



ANEXO 6: Comparación de la Ventana de Congestion en TCP Tahoe.



ANEXO 7: Comparación de la Ventana de Congestion en TCP SACK.



ANEXO 8: Comparación de las Demoras de Respuestas del Servidor FTP para 4 pérdidas consecutivas de segmentos en una misma ventana.