xx(99877,1)

CINVESTAV - IPN

Centro de Investigación y de Estudios Avanzados del IPN
Unidad Guadalajara

Implementación de la Opción SACK en el Protocolo TCP.

TESIS QUE PRESENTA
MARCO ANTONIO MORA ARVIZO

PARA OBTENER EL GRADO DE
MAESTRO EN CIENCIAS

EN LA ESPECIALIDAD DE
INGENIERÍA ELÉCTRICA

Guadalajara, Jal., Agosto de 1999

Implementación de la Opción SACK en el Protocolo TCP

Tesis de Maestría en Ciencias
Ingeniería Eléctrica

Por:

Marco Antonio Mora Arvizo

Ingeniero Electrónico
Universidad de Guadalajara 1988 – 1993

Becario del **CONACYT**, expediente no. 96833

Directores de Tesis:

Dr. Manuel Edgardo Guzmán Rentería

CINVESTAV del IPN Unidad Guadalajara, Agosto de 1999.

Agradecimientos:

Agradezco a CONACYT el apoyo económico brindado durante mis estudios de maestría.

Agradezco el apoyo brindado por parte de mis padres y hermanos.

Introducción

Desde que se definió el protocolo TCP en 1981 [12] se han hecho muchos cambios debido a que se han encontrado algunos problemas, así como algunas propuestas para incrementar el desempeño. Entre las propuestas más destacadas están los algoritmos de control de congestión (*Congestion Avoidance*), inicio lento (*Slow Start*), retransmisión y recuperación rápida (*Fast Retransmit and Fast Recovery*) explicados en [10] [9] [14], las implementaciones actuales de TCP deben de tener estos algoritmos como se menciona en [1].

En la actualidad se sigue trabajando en mejoras al protocolo, en parte debido a que las tecnologías en los sistemas de comunicación han evolucionado, ahora se tienen dispositivos más rápidos y anchos de banda mayores, ha sido necesario ir adaptando los protocolos para hacer un mejor uso de los recursos.

El propósito de este trabajo de tesis, es el de evaluar el comportamiento del protocolo TCP haciendo uso de la opción de acuse de recibo selectivo (SACK), dicha extensión se describe en el RFC2018 [13].

SACK es una extensión de TCP que hace uso del espacio reservado para opciones en el encabezado de TCP, esta extensión tiene como finalidad el evitar retransmisiones innecesarias, y como consecuencia tener un incremento en el tráfico cursado o capacidad.

La evaluación de esta extensión se hizo sobre el núcleo 2.0.30 de Linux y un canal simulado de 2048 kbps, con un retardo de 10 ms y 200 ms, y una probabilidad de pérdida que varía entre 0 y 5%.

Índice General

Índice de Figuras

Capítulo 1

Introducción a TCP

1.1 Un Protocolo de Entrega Confiable

En los niveles más bajos, los protocolos de comunicación no proveen una entrega de paquetes confiable, esto es, algunos paquetes pueden perderse, ya sea porque ocurre un error en la transmisión , o cuando los buffers de los dispositivos se llenan, simplemente tiran los paquetes que van llegando. Además las redes que tienen ruteo dinámico, pueden entregar los paquetes en un orden diferente al que se enviaron o con retardo, ya que estos protocolos, como por ejemplo IP [11] se dice que realizan su mejor esfuerzo, pero no garantizan la entrega.

Las aplicaciones de niveles más altos requieren con frecuencia de enviar grandes cantidades de información, y si no se cuenta con un protocolo de transporte que nos garantice esta entrega, el chequeo de errores se puede convertir en algo tedioso y difícil de implementar en la aplicación. Un protocolo que quita este problema de las aplicaciones y utiliza un protocolo de entrega de datagramas (IP), es el protocolo llamado TCP *(Transport Control Protocol Protocolo de Control de Transporte)* pertenece al conjunto de protocolos conocidos como *TCP/IP*

TCP es un protocolo orientado a conexión, esto significa que se tiene un enlace virtual punto a punto por el cual se transfiere un flujo de bytes. Se puede hacer una analogía con una llamada telefónica, en la cual antes de que se comience a transferir información se realiza una petición de uno de los

extremos al otro, una vez que ambos extremos estan de acuerdo, se comienza la comunicación, viéndolo desde el punto de vista del protocolo TCP, antes de transmitir datos, se hace una llamada al sistema operativo indicándole que se quiere hacer conexión con una máquina remota, este envia la petición utilizando como fuente su dirección y un número de puerto (a esta dupla de dirección IP y número de puerto se le conoce como socket [5] [14]) y como destino la dirección IP de la máquina destino y un puerto.

1.1.1 Confiabilidad

Para poder proporcionar confiabilidad, se debe de contar con un mecanismo con el cual el receptor informe al transmisor que paquetes llegaron con éxito. Para lograr esto, el mecanismo que se utiliza es el de acuse de recibo.

La gráfica en la Figura 1.1 muestra la idea básica de este mecanismo, el transmisor envía un paquete, cuando en el otro extremo se recibe, éste envía un paquete informando que se recibió correctamente,[1] y se envía el siguiente paquete. En el caso en que el paquete se pierda (como se muestra en la Figura 1.2), después de un tiempo, un temporizador expira y el paquete se retransmite.[2]

1.1.2 La Ventana Deslizante

Como se vio anteriormente el enviar un paquete y esperar a recibir el acuse de recibo para poder transmitir el siguiente, muestra claramente que se desperdicia ancho de banda, pues hay intervalos de tiempo en los cuales no se transmite, y el canal esta ocioso, por lo tanto el desempeño del protocolo es pobre.

Para combatir el problema mencionado, TCP utiliza una forma en la cual se envía un cierto número de paquetes, y conforme se van recibiendo los acuses de recibo, se van enviando otros paquetes, a este mecanismo se le conoce como ventana deslizante. Para clarificar un poco esta idea, consideremos una ventana de 8 paquetes. El transmisor puede enviar hasta 8 paquetes

[1] Realmente se informa el número del paquete que se espera recibir

[2] Este tiempo se calcula en base a una estimación que se hace del retardo existente en la red.

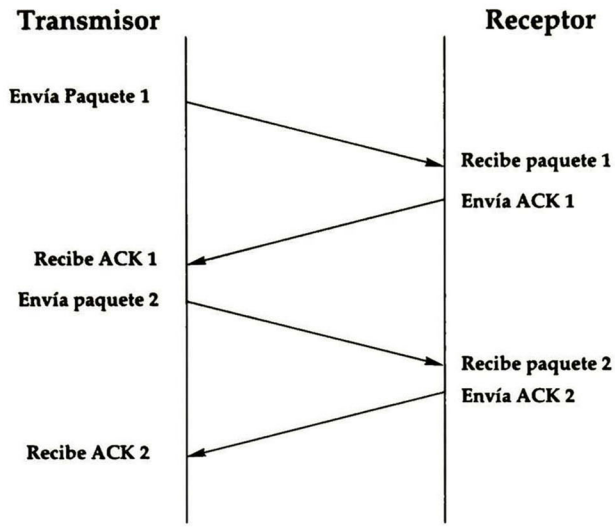

Figura 1.1: Transmisión con ACK simple

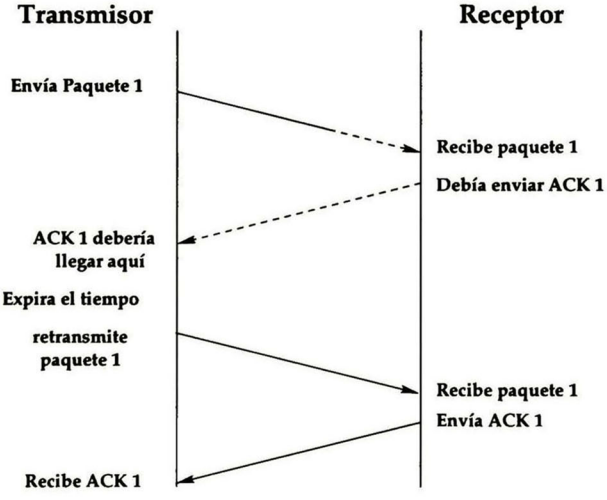

Figura 1.2: Pérdida de paquete

en rápida sucesión y luego espera a recibir un acuse de recibo, cuando envía el octavo paquete, el primero podría ya haber llegado a su destino, y éste haber sido acusado por el receptor,[3] el transmisor al recibir la información mueve la ventana hacia la derecha, dicho de otra manera los paquetes que se encuentran dentro de la ventana pueden ser enviados sin retardo, mientras que los paquetes del extremo derecho fuera de la ventana, no pueden ser enviados hasta que se reciba el acuse del paquete de más a la izquierda dentro de la ventana, al recibir este acuse provoca que se deslice la ventana hacia la derecha y así poder transmitir el paquete que ahora pasa a estar dentro de la ventana.

1.1.3 Acuse Acumulativo

Muchas implementaciones para reducir procesamiento y tráfico, retardan el envio del acuse de recibo. TCP soporta el acuse acumulativo, esto es, el transmisor puede no recibir un acuse por cada paquete que envía, sino que se puede recibir un acuse de un paquete y de los paquetes anteriores que no hayan sido acusados explícitamente, cuando esto ocurre se asume que los paquetes que preceden al que provocó el acuse de recibo llegaron exitosamente, y recorre su ventana tantos paquetes como el número de paquetes que se interpreta que han llegado bien. La Figura 1.3 ilustra el concepto de acuse de recibo acumulativo.

[3]Si el receptor tiene datos que enviar, estos datos son enviados en el mismo paquete que lleva el acuse de recibo, a esto se le conoce como *piggybacking* o montado de acuse de recibo sobre datos

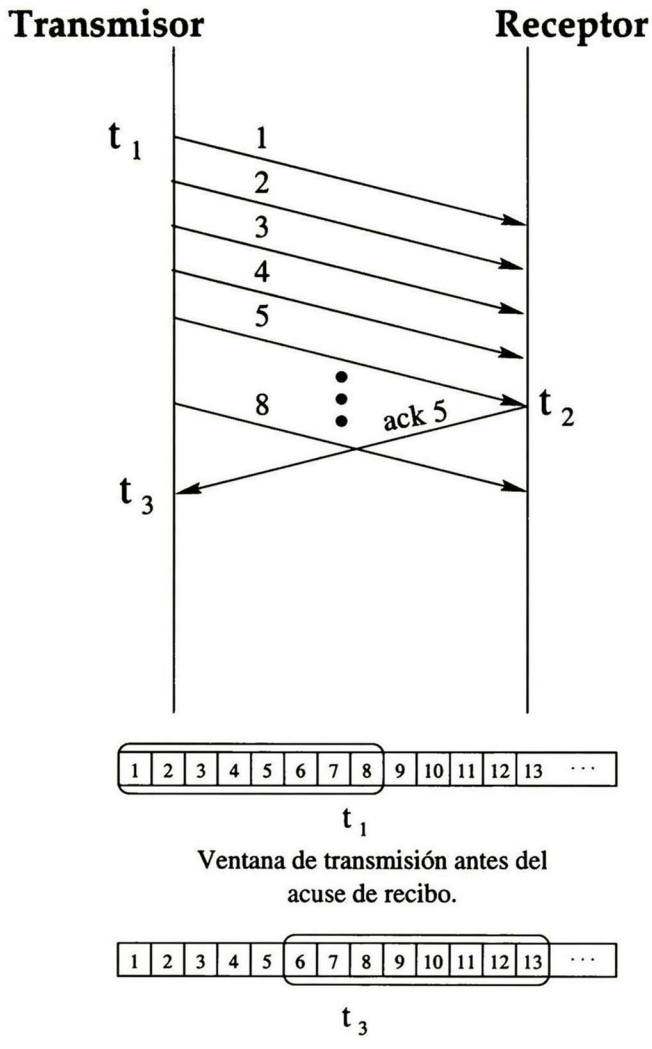

Figura 1.3: Acuse Acumulativo

Capítulo 2

Una Extensión a TCP

2.1 SACK, Una Extensión de TCP

Debido a que TCP utiliza acuse de recibo acumulativo [12], en el cual, si el paquete no corresponde al de más a la izquierda de la ventana, no sera acusado. Provocando que el transmisor tenga que esperar un tiempo de ida y vuelta (*roundtrip-time*) para saber de cada paquete que fue perdido, o hacer una retransmisión innecesaria de segmentos que ya han sido recibidos correctamente.

Con la estrategia de acuse selectivo (*Selective Acknowlegdement SACK*) [13] se soluciona este problema cuando se presenta una pérdida de múltiples segmento. Utilizando este esquema de SACK el receptor puede informar al transmisor de los paquetes que ha recibido satisfactoriamente, y de esta manera el transmisor únicamente reenviará los paquetes que realmente se han perdido.

Algunos protocolos de transporte, como, **NETBLT [4], XTP [15], RDP [16], NADIR [6], VMTP [3]**. han utilizado acuse de recibo selectivo. Algunos experimentos hechos con RDP mostraron que deshabilitando el acuse selectivo incrementa bastante el número de retransmisiones en un canal con pérdidas y un retardo alto.

El esquema SACK fue inicialmente descrito en el RFC 1072 [7] pero nunca fue implementado en Internet por el desacuerdo de que si la opción SACK debía ser implementada en conjunto con el escalamiento de ventana

[8]. Algunas modificaciones se han hecho a este esquema como se describe en RFC 2018.

Esta extensión a TCP utiliza 2 opciones, una que es con la cual ambos extremos se acuerdan el uso de SACK (*SACK-Permitted*). Esta opción es enviada en los segmentos **SYN**. La otra opción es la opción SACK propiamente dicha, Esta segunda opción debe de enviarse únicamente si en la etapa de establecimiento de conexión se acordó su uso por ambos extremos.

La segunda opción que se presenta es enviada desde un extremo TCP que recibió datos no contiguos hacia el transmisor para mostrarle el estado de su buffer.

2.1.1 Opciones en TCP

Las opciones en TCP ocupan el espacio al final del encabezado y deben ser de longitud en múltiplos de 8 bits, existen 2 casos para el formato de una opción.

1. Un simple octeto que indica el tipo de opción.
2. Un octeto de tipo de opción, longitud de la opción y los datos propios de la opción.

La longitud de la opción debe incluir los dos octetos correspondientes a el tipo de opción y longitud, así como los datos de la opción.

El uso de las opciones se explica en [12].

2.1.2 Opción SACK Permitido

Esta opción esta compuesta de 2 bytes que serán enviados en un segmento **SYN** por el extremo TCP que esta capacitado para recibir y procesar esta extensión, cabe remarcar que esta opción **no** debe ser enviada en segmentos **NO-SYN**.

La opción *SACK permitido* como se mencionó anteriormente, es de 2 bytes como se muestra en la figura 2.1. El tipo utilizado es el 4.

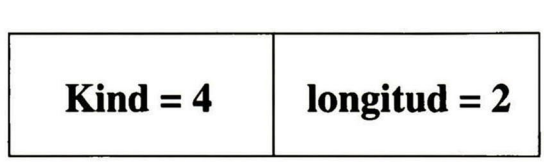

Figura 2.1: Opción que permite el uso de SACK

Esta opción únicamente nos sirve para que ambos extremos acuerden el uso de la opción SACK en los segmentos subsecuentes y que no lleven la bandera SYN encendida.

2.1.3 Opción SACK Propiamente Dicha

Esta opción es utilizada para llevar la información del estado del buffer del receptor hacia el transmisor, la longitud puede ser variable, el tipo utilizado es el 5, como se muestra en la figura 2.2.

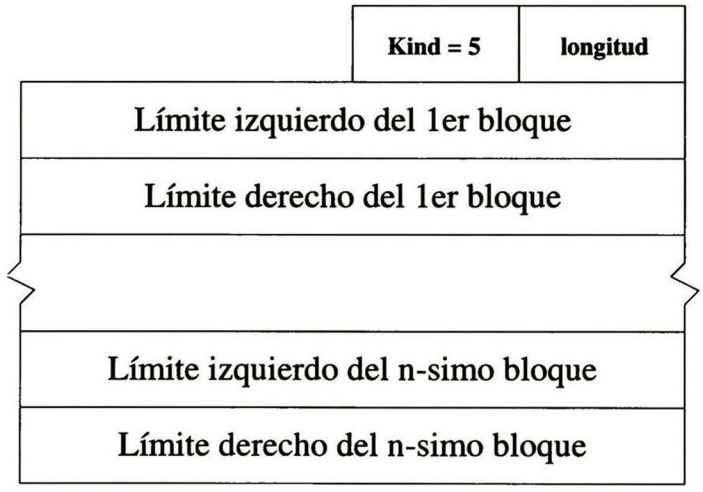

Figura 2.2: Opción SACK

La opción SACK debe ser enviada de el receptor al transmisor, cuando se han recibido y encolado bloques de datos no contiguos, el receptor espera recibir estos bloques de datos retardados o por retransmisión para llenar los

huecos. Cuando estos datos son recibidos, el receptor acusa normalmente los datos avanzando el limite izquierdo de la ventana de secuencia de acuse de recibo y mandando el acuse de recibo en el campo correspondiente en el encabezado TCP.

Cada bloque contiguo que es avisado por la opción SACK es definido por dos enteros sin signo de 32 bits con el orden de byte de red.

- Limite izquierdo del bloque

  Este indica el primer número de secuencia de este bloque.

- Limite derecho del bloque

  Indica el número de secuencia que sigue al último número de secuencia de este bloque.

Cada bloque representa los datos recibidos que están contiguos pero aislados, esto es que los bytes anteriores al limite izquierdo y los posteriores al limite derecho, no han sido recibidos.

La opción SACK que especifica n bloques, tiene una longitud de $8*n+2$ bytes, considerando que se puede tener un máximo de 40 bytes en las opciones y asumiendo que esta opción puede ser utilizada junto con la opción definida en [7], únicamente es posible avisar como máximo 3 bloques no contiguos.

Generación de la opción SACK en el receptor

Se debió haber acordado al inicio de la conexión el uso de la opción SACK por medio del intercambio de la opción *SACK permitido*, cabe remarcar que si no se hizo este acuerdo no debe ser utilizada la opción SACK por ningún motivo durante la conexión.

Una vez acordado el uso de la opción, la opción SACK debe ser incluida en todos los acuses de recibo, los cuales no acusen el número de secuencia más alta en la cola de los datos recibidos. El receptor debe de enviar un acuse por cada segmento válido que le llega y que contiene datos nuevos y

que además pertenecen a un bloque no contiguo, cada uno de estos acuses "duplicados" deberá llevar la opción SACK.

Cuando se va a utilizar la opción SACK se deben de seguir las siguientes reglas:

- El primer Bloque SACK especifica el bloque de datos contiguos que incluye el segmento que genero este acuse, a menos que el bloque avance el número de acuse de recibo en el campo designado para esto en el encabezado.
- El receptor deberá incluir tantos bloques SACK como le sea posible.
- La opción SACK debe de ser llenada, repitiendo los bloques SACK reportados más recientemente que no sean un subconjunto de un bloque SACK ya incluido en la opción SACK que se está construyendo.

Interpretación de la Opción SACK

Cuando se recibe un acuse de recibo que contiene la opción SACK, esta debe de ser guardada para futura referencia. Se debe de guardar la referencia más nueva.

Se asume que el transmisor tiene una cola ordenada de retransmisión con los datos que ha transmitido, pero que aun no han sido acusados

En futuras retransmisiones se debe evitar retransmitir bloques que han sido avisados en la opción SACK, Los bloques no debe de ser removidos hasta que el límite izquierdo de la ventana haya avanzado sobre ellos.

Los algoritmos existente para retransmisión rápida (fast-retransmition) y los utilizados para evitar congestión, permanecen sin ningún cambio, esta opción no altera el funcionamiento de éstos.

Capítulo 3

Implementación y Ambiente de Pruebas

3.1 Implementación

El sistema operativo en el que se implementó la extensión de TCP es Linux, principalmente por ser un sistema operativo abierto, de el cual es posible obtener el código fuente. La versión del núcleo en la cual se hizo la implementación es la versión 2.0.30, se seleccionó esta versión por ser la que en el momento que se inicio este trabajo de tesis era la versión estable más reciente. Otro punto que se consideró fue el que no existía ningún resultado reportado de el comportamiento del protocolo TCP utilizando la opción SACK en este sistema operativo.

Los datos reportados de análisis que se encontraron eran los hechos en FreeBSD con SACK [2], la implementación del protocolo TCP en este sistema operativo está basado en una implementación conocida como TCP-Reno, de aquí que la comparación que se hace es básicamente entre TCP-Reno vs TCP-Sack, también existen referencias de otras implementaciones experimentales las cuales comparan TCP-Sack con TCP-Reno y TCP-Tahoe, hechas principalmente en *PSC*[1] y en *LBNL*[2], una cosa que hacia interesante con Linux,

[1] Pittsburgh Supercomputing Center
http://www.psc.edu/networking/index.html

[2] Lawrence Berkeley National Laboratory
http://www-nrg.ee.lbl.gov/nrg.html

era ver el comportamiento que este sistema operativo podía tener, debido a que la implementación de TCP no esta basada ni en la implementación de Reno ni en la de Tahoe.

En Linux no había nada hecho, y una de las cosas que hace diferente a Linux de los demás Unix es que este no tomó las implementaciones de Reno o Tahoe si no que se desarrolló desde cero, por tal razón era interesante ver el comportamiento que tendría en esta implementación, además de que el manejo de los buffers es totalmente distinto a los demás sistemas.

3.2 Buffers para Sockets

Uno de los problemas que se presentan cuando se tienen varias capas de protocolos, y en donde unos utilizan servicios de los otros, es que cada protocolo debe agregar encabezados o información al final de los datos que quieren ser transmitidos, y removidos cuando los datos son recibidos. Esto hace difícil el paso de los buffers de datos entre las capas de protocolos, ya que cada capa necesita saber en donde se encuentra la información de su encabezado o cola.

En Linux para solucionar este problema se utiliza algo que se llama *buffer de sockets (socket buffers)* o *sk_buffs*, para pasar los datos entre las capas de protocolos y los dispositivos de red. Los sk_buffs contiene apuntadores y campos que indican longitudes, para permitir que cada capa de protocolos manipule los datos y la información correspondiente al protocolo por medio de funciones estándar, la figura 3.1 muestra la estructura de datos sk_buff. Cada sk_buff tiene un bloque de datos asociado, el sk_buff tiene cuatro punteros que sirven para manejar la estructura.

head Apunta al comienzo del área de datos en memoria, este se fija cuando se reserva el espacio en memoria.

data Apunta al actual comienzo de los datos del protocolo, este varía de acuerdo a la capa de protocolo en la que se encuentra.

tail Apunta la final de los datos del protocolo, este también varía dependiendo de a que capa de protocolo pertenece actualmente.

end Apunta al final del espacio de datos en memoria, este al igual que le primero se fija cuando se reserva el espacio en memoria.

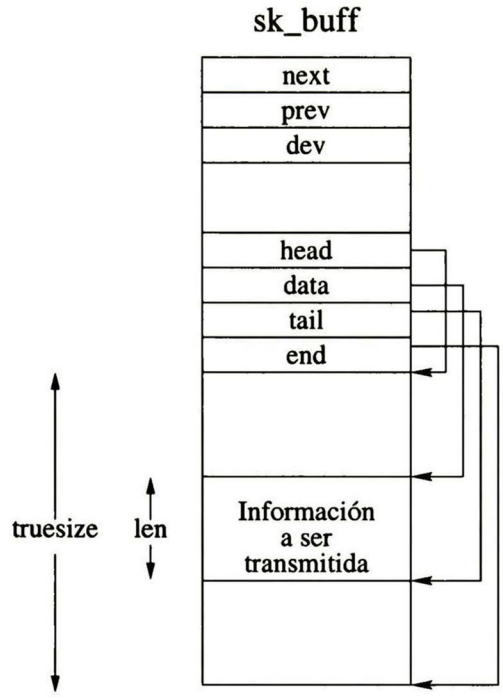

Figura 3.1: Estructura sk_buff

Como se ve en la figura 3.1 hay dos campos de longitud, *len* y *truesize*, los cuales contienen la longitud del paquete del protocolo actual, y el tamaño total del buffer asociado al sk_buff respectivamente. Para el manejo de los sk_buffs existen mecanismos estándar para agregar y remover encabezados y colas de los protocolos, estas funciones manipulan los punteros **data, tail** y la variable **len** en sk_buff.

Estas funciones son:

sk_push Esta función mueve el puntero *data* hacia el principio del área de datos e incrementa el valor de *len*. Es utilizada para agregar datos o encabezado del protocolo al principio de los datos que se quieren transmitir.

sk_pull Esta mueve el puntero *data* del comienzo hacia el final del área de datos y decrementa el valor de *len*. Es usada cuando se extraen datos

o el encabezado del protocolo de los datos que se han recibido.

sk_put Mueve el puntero *tail* hacia el final del área de datos e incrementa el valor de *len*. Se usa cuando se desea agregar datos o información del protocolo al final de los datos a ser transmitidos.

sk_trim Mueve le puntero de el final hacia el comienzo del área de datos, y decrementa el valor de *len*. Es utilizada cuando se desean extraer datos del protocolo del final del paquete recibido.

La estructura sk_buff tiene también punteros que le permiten ser almacenados en listas circulares doblemente ligadas.

3.3 Organización del Núcleo

Dentro de la organización de los fuentes del núcleo, existe una estructura de directorios en donde se encuentra el código referente a los protocolos de red.

TCP se encuentra organizada principalmente en 4 grupos de funciones. Un grupo contiene las funciones utilizadas para la inicialización de conexión, de estructuras de control de conexión y funciones comunes a los otros grupos, los tres grupos restantes abarcan las funciones de entrada, salida y manejo de los temporizadores. Aunque en este directorio se encuentran las funciones que implementan el protocolo TCP, también se hace uso de funciones generales a los protocolos como las funciones de manejo de memoria, como se mencionan en 3.2.

3.4 Implementación de SACK

Para la implementación de SACK fue necesario hacer modificaciones tanto en las funciones de inicialización, como en funciones de entrada y salida, fue necesario incluir, al inicio de una conexión (paquete con la bandera SYN activada) se agregara la petición del uso de SACK, esto es, enviar la opción SACK-permitted, y por consiguiente que al recibir el paquete, la máquina de estados que analiza el encabezado entendiera esta opción, para que al generar

la respuesta (paquete con la bandera SYN y ACK activadas) también se envié esta opción.

Una vez acordado el uso de la opción SACK, se debe de mantener la información más actual del estado del buffer de recepción , para lograr esto se agregaron variables a la estructura que mantiene la información de la conexión (estructura sock), la información es actualizada cada que se recibe un paquete, por ejemplo, si el paquete que llega pertenece a un bloque no contiguo de datos, este se almacena en el buffer de recepción, después de esto se analiza el buffer de recepción a partir del bloque no contiguo para determinar el número de bloques, así como el número de secuencia del inicio y final del o de los bloques. Los números de secuencia de los bloques no contiguos son utilizados por la función que envía información junto con el acuse de recibo, o por la función que envía únicamente el acuse de recibo, para formar la opción SACK.

Por otra parte, es necesario también verificar si el paquete que llega lleva la opción SACK con la información del buffer del receptor en el otro extremo, si es así, es necesario extraer esta información y guardarla para ser usada en futuras retransmisiones, esta información se actualiza por cada paquete que llega con la opción SACK, de esta manera se tiene la imagen lo más recientemente posible del buffer de recepción del otro extremo.

Para generar un núcleo que pueda ser utilizable y que incluya la extensión SACK, y también tener otro que no incluya esta extensión, debido a que se necesitan ambos para hacer las simulaciones y las comparaciones, se hizo necesario hacer uso de las instrucciones para hacer compilación condicional. Además se agregaron las opciones necesarias a los archivos que generan el menú de configuración de las opciones del núcleo

3.5 Modelo de Canal Deseado

Se requiere tener un canal en el cual se pueda controlar la pérdida, este tipo de canal es necesario debido a que la extensión SACK que se desea probar, es útil cuando el canal en el que se mantiene la comunicación, presenta pérdidas. Un objetivo de este trabajo de tesis, es ver el comportamiento del protocolo TCP haciendo uso de la extensión SACK en un canal con diferente probabilidad de pérdida y diferente retardo.

El modelo deseado es el que se presenta en la figura 3.2

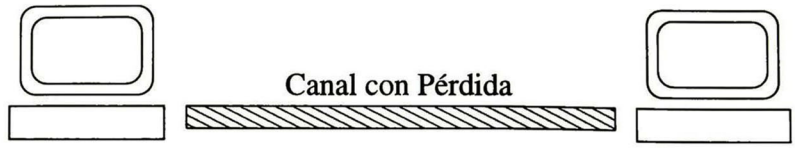

Figura 3.2: Canal Deseado

3.6 Modelo del Canal Usado

El canal que se utilizó para los experimentos, fue un canal que se simuló mediante un par de programas y un esquema como el que se muestra en la figura 3.3

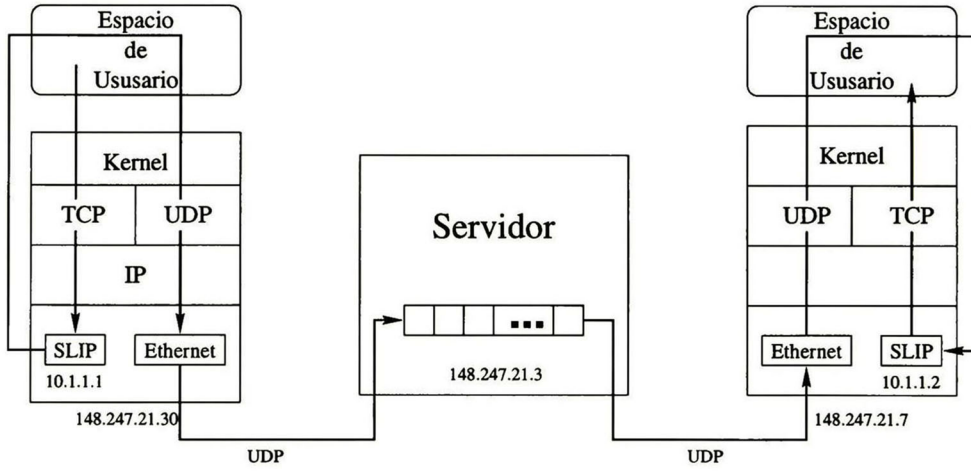

Figura 3.3: Simulación del canal

Se utilizaron 2 computadoras personales, ambas corriendo Linux, y una estación de trabajo Sparc 10 corriendo SunOs como sistema operativo. Las computadoras personales eran las únicas que tenían la implementación de la opción SACK, la estación de trabajo únicamente se utilizó para producir el retardo y controlar el ancho de banda.

Control de Pérdidas y Retardo

En las computadoras personales se corre un programa, que desde el núcleo es visto como una interfaz SLIP y en el espacio de usuario se ve como una pseudo-terminal (pty), esta interfaz SLIP es realmente un pseudo-dispositivo. Así cuando hay paquetes que son enviados desde una aplicación y que deben ir a través de la interfaz SLIP, los paquetes enviados son interceptados por el núcleo y encolados en dicho pseudo-dispositivo, la pty es utilizada por este mismo programa como un descriptor por el cual desencola los paquetes que el núcleo encoló previamente, este descriptor también se usa en sentido contrario, cuando se encolan paquetes desde la pseudo-terminal, son desencolados por el núcleo y pasados a la capa IP.

El programa que hace lo antes mencionado corre en espacio de usuario, y por medio de la función de control de dispositivos (ioctl) crea la interfaz SLIP Dentro de este programa se utiliza una función de generación de números pseudo-aleatorios con una función de distribución uniforme, la salida del generador se utiliza para decidir si el paquete se transmite hacia el programa servidor que se encuentra en la Sparc, o se desecha, creando así las pérdidas.

La función de generación de números pseudo-aleatorios entrega valores entre [0.0 y 1.0), en donde los valores que están ente 0.0 y 0.5 tienen una probabilidad de ocurrencia de 50%, así como los valores entre 0.0 - 0.1 y 0.0 0.01 tienen una probabilidad de ocurrencia de 10% y 1% respectivamente, esta probabilidad de ocurrencia es la que se utiliza para decidir si el paquete es enviado o simplemente se tira.

```
si (valor_aleatorio > pérdida)
        transmite
```

Este programa cliente también escucha en un puerto UDP, todos los paquetes que recibe por este puerto son escritos o encolados hacia el pseudo-dispositivo para que sean tomados por el núcleo y pasados a la capa IP.

En la Sparc corre un programa, que llamaremos servidor, el cual genera 2 procesos, cada uno de ellos escucha en un puerto UDP definido (debido a que el protocolo que se esta probando es TCP, se utiliza UDP para emular el canal y no tenga ninguna influencia sobre los experimentos), los paquetes recibidos son metidos en una cola, existe una cola para cada dirección de la

transmisión, los paquetes se van desencolando de acuerdo al tiempo especificado para el retardo, después es transmitido hacia el otro cliente (no al que envía el paquete que se esta tratando) a un puerto UDP definido. También se utiliza el algoritmo de *leaky-bucket*[3] para limitar el ancho de banda.

Ejemplo

Si se quiere tener una comunicación entre 10.1.1.1 y 10.1.1.2, los paquetes transmitidos desde la máquina con dirección 10.1.1.1 son capturados por el núcleo y encolados en la interfaz SLIP, de aquí son extraídos y enviados por la interfaz Ethernet (en este caso con dirección 148.247.21.30) encapsulado en UDP hacia la máquina con dirección 148.247.21.3 que es donde se encuentra el programa servidor, aquí son metidos en una FIFO y desencolados como se explicó anteriormente y enviados por la conexión Ethernet hacia la otra máquina cliente (en este caso con dirección 148.247.21.7) también por medio de UDP. En esta máquina, como se mencionó anteriormente hay un programa cliente igual al de la máquina que generó el paquete, que escucha en un puerto UDP, al cual transmitió el servidor, este programa encola los paquetes en la interfaz SLIP con dirección 10.1.1.2, para que sean extraídos por el núcleo y pasados a la capa IP, creando de esta manera un túnel. Se puede ver de una manera más clara en la Figura.3.3.

[3]Este algoritmo es utilizado para mantener un flujo de salida constante, independientemente del flujo de entrada

Capítulo 4

Análisis de Resultados

4.1 Características del Canal y Herramientas de Análisis

Las características utilizadas en el canal para la realización de las pruebas, son las de un canal E1[1], con un retardo de 10ms, variando la probabilidad de pérdida entre 0, 1, 3, y 5%.

Para comprobar las características del canal se utilizó el programa pathchar.[2] obteniendo como resultado las siguientes características.

```
1 hops, rtt 31.5 ms (37.5 ms), bottleneck 2.0 Mb/s, pipe 9395
                                bytes
```

Aquí se observa que tenemos únicamente un salto entre fuente y destino con un tiempo de ida y vuelta de 31.5 ms, un ancho de banda máximo de 2 Mb/s y un buffer en el canal de aproximadamente 10 000 bytes.

Para la generación de tráfico se hizo uso del programa ttcp, este programa fue originalmente escrito por Mike Muuss en el Laboratorio de investigación en balística con la finalidad de probar el desempeño de los protocolos

[1] En el canal se considera que se usan las 32 ranuras del E1, teniendo entonces 2048 Mbps

[2] http://www-nrg.ee.lbl.gov/nrg.html

TCP y UDP. En un extremo de la conexión se ejecuta en modo de recepción, en el otro extremo se ejecuta en modo transmisión y es posible indicarle el número de paquetes a transmitir. Se utiliza el programa tcpdump[3] para la captura de paquetes que circulan por la interfaz y para monitorear el puerto, a este programa fue necesario hacer unas pequeñas modificaciones a el código fuente para que mostrara la información correspondiente a la opción SACK. Tcpdump genera un archivo con la captura de los paquetes que cursan por esta interfaz. Los datos obtenidos son analizados por medio de un programa escrito en Perl generando el archivo de entrada necesario para ser graficado, mostrando en el eje de las *x's* el tiempo y en el de las *y's* el número de secuencia. Otro análisis se hace con el programa tcptrace[4] de el cual se obtiene la capacidad así como el archivo necesario para mostrar la gráfica de la capacidad. Para visualizar las gráficas se hace uso del programa xplot[5]

% Perdida	TCP s-SACK	TCP c-SACK	TCP c-SACK/TCP s-SACK
1 %	175418 Kbps	181239 Kbps	1.03
3 %	76489 Kbps	87298 Kbps	1.14
5 %	35525 Kbps	43528 Kbps	1.22

Tabla 4.1: TCP con SACK vs TCP sin SACK

4.2 Análisis de Resultados

Como se mencionó anteriormente, lo que se varió en el canal fue la probabilidad de pérdida, se observó que con un 0% de pérdida el comportamiento con la implementación SACK no disminuía la capacidad debido a el procesamiento que esto implicaba, de tal manera que la capacidad de tráfico cursado era la misma, por lo tanto los resultados obtenidos no se considero para el análisis, únicamente se consideraron los resultados obtenidos con 1%, 3% y 5% de pérdida. Se dividió el número de Kbytes por segundo obtenidos de la prueba

[3]http://www-nrg.ee.lbl.gov/nrg.html
[4]http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html
[5]ftp://mercury.lcs.mit.edu/pub/shep

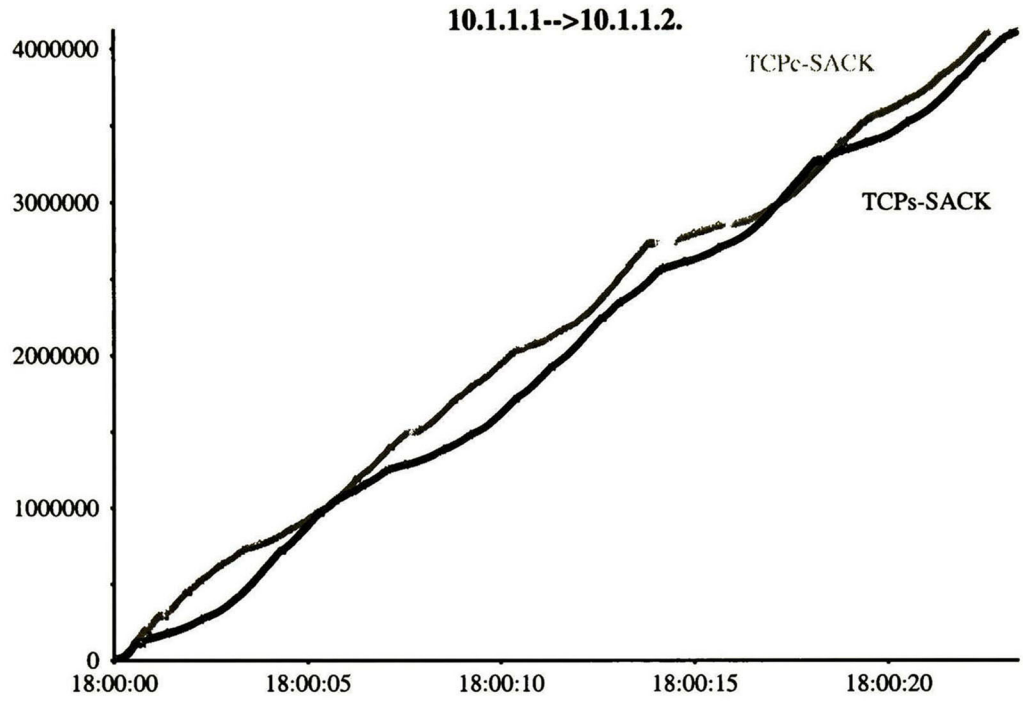

Figura 4.1: Comparación con pérdida de 1%

haciendo uso de la opción SACK, entre los Kbytes por segundo de los experimentos hechos sin la opción SACK, para tener como resultado el porcentaje de la ganancia obtenida, Los resultados arrojados son mostrados en la tabla 4.1.

Como se observa en la tabla 4.1 se ve claramente que cuando se tiene una pérdida es de 1% la ganancia que se obtiene es muy poca, de apenas un 3%. Cuando existe una pérdida de 3% la ganancia que se obtiene es de un 14% que ya se podría considerar importante. Pero cuando la pérdida que se tiene en el canal es de un 5%, se puede apreciar que la ganancia que se tiene es de un 22% lo cual ya es considerable. Lo anterior es representado de una forma gráfica para que pueda ser apreciado más claramente.

La figura 4.1 muestra la comparación cuando la probabilidad de pérdida es de 1%, el eje de las x's se presenta el tiempo, y el eje de las y's representa

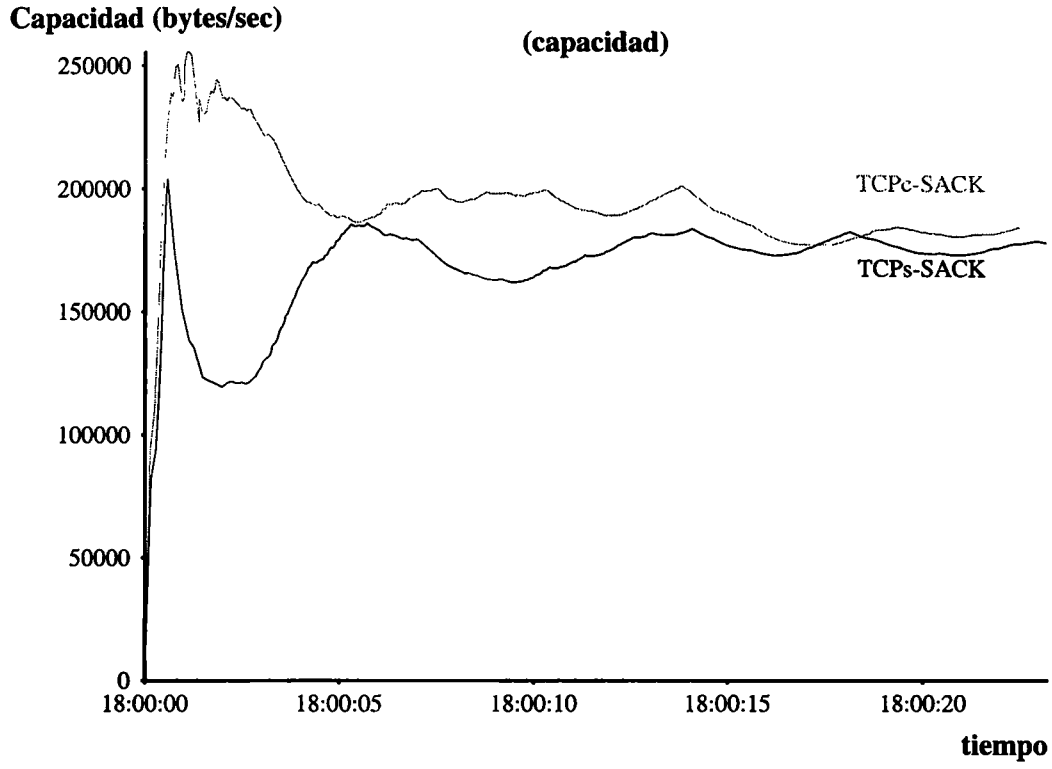

Figura 4.2: Capacidad con 1% de pérdida

en número de secuencia[6], en este tipo de gráficas si la pendiente es más pronunciada significa que nuestra transmisión es mejor pues nos indica que el tiempo requerido es menor, de lo anterior obtenemos que la ganancia que se obtiene en este caso no es muy significativa, como se menciono anteriormente. En la gráfica 4.2 se presenta una comparación de la capacidad con la misma probabilidad de pérdida.

Analizando los resultados obtenidos con un 3% de pérdida, de la tabla 4.1 vemos el incremento en la capacidad, esto expresado de una manera gráfica como se observa en la figura 4.3, en esta gráfica se ve claramente que la ganancia obtenida en más significativa en comparación a la obteni-

[6]el número de secuencia se genera aleatoriamente en el inicio de la conexión, y se incrementa de acuerdo al número de bytes transmitidos

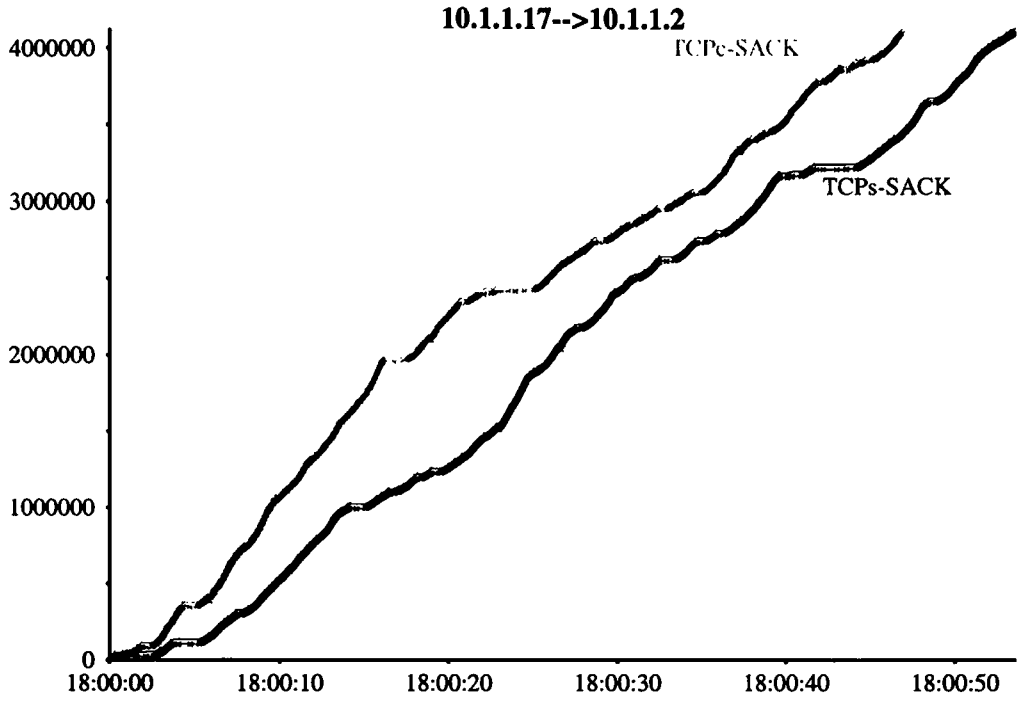

Figura 4.3: Comparación con 3% de pérdida

da con un 1% de pérdida, se puede apreciar como la gráfica que muestra el comportamiento que utiliza la opción SACK presenta una pendiente más pronunciada, y como se menciono anteriormente esto indica un mejor desempeño.

En la figura 4.4 se muestra la comparación de la capacidad, en esta gráfica se puede observar que la linea que representa la conexión utilizando SACK termina un poco antes que no hace uso de esta opción, esto significa que para un mismo número de paquetes a transmitir correctamente (esto es sin contar los retransmitidos) se requirió de un menor tiempo.

Al igual que en los dos casos anteriores, se compara los resultados con un 5% de pérdida, en la figura 4.5 se muestra esta comparación, aquí también se observa claramente el comportamiento cuando se utiliza SACK.

La gráfica 4.6 muestra la comparación de la capacidad, en esta gráfica

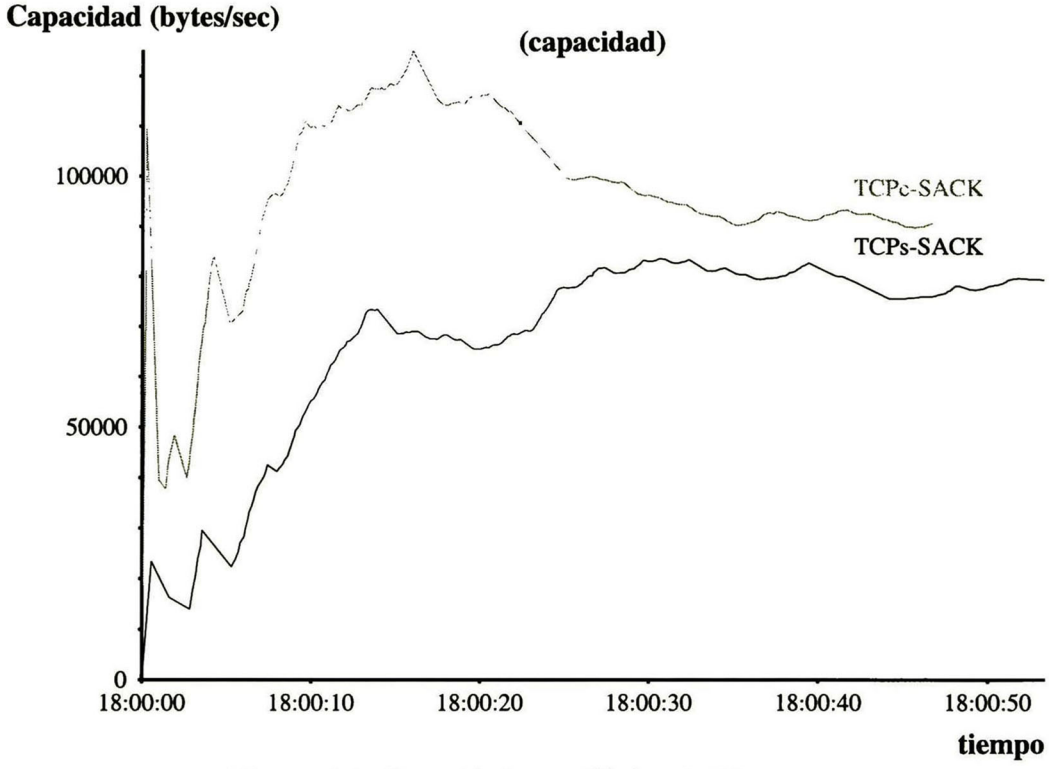

Figura 4.4: Capacidad con 3% de pérdida

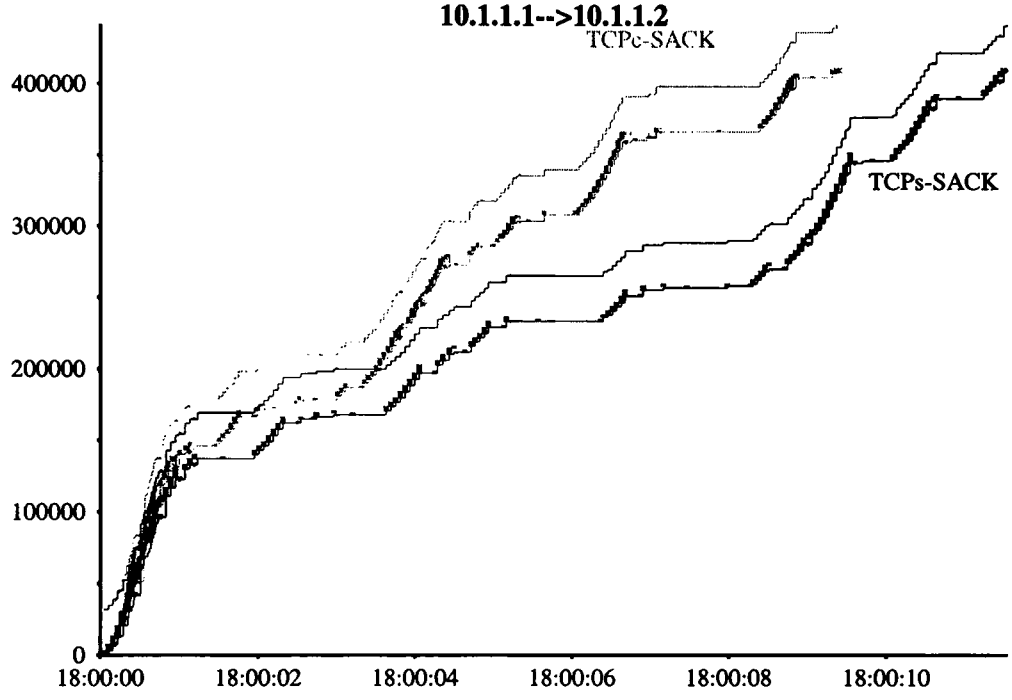

Figura 4.5: Comparación con 5% de pérdida

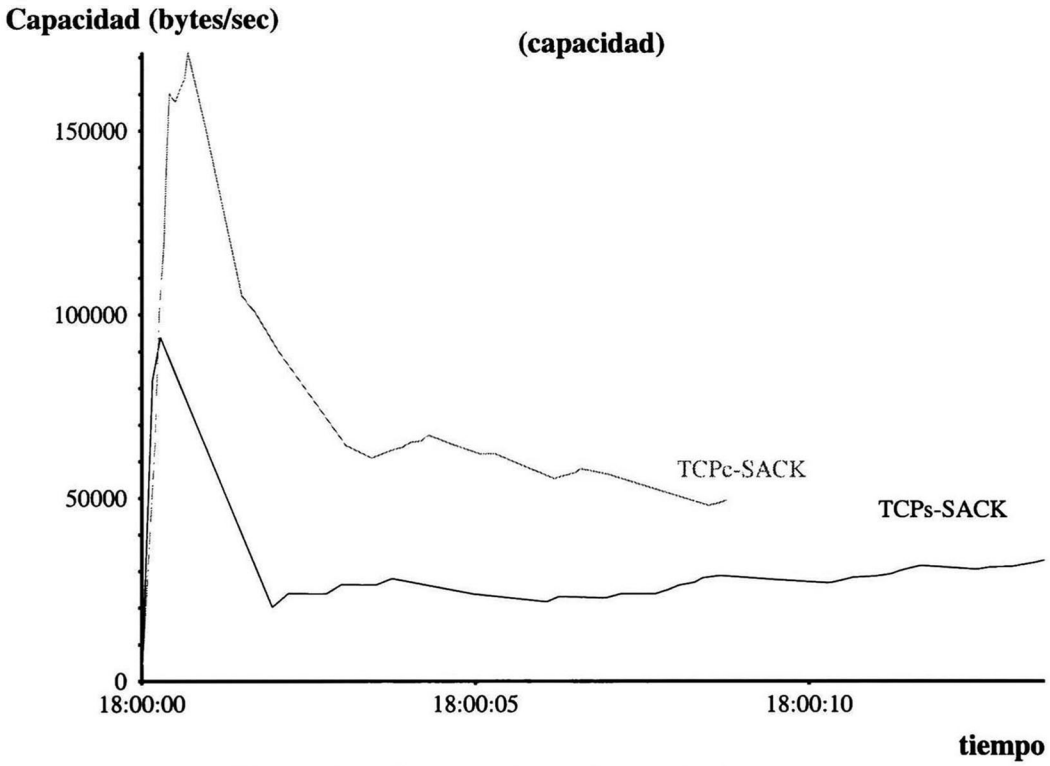

Figura 4.6: Capacidad con 5% de pérdida

se observa que el tiempo requerido para la transmisión de los paquetes fue significativamente menor.

4.3 Canal con Mayor Retardo

Se realizaron experimentos simulando un canal E1 con un retardo de 200 ms, este canal podríamos considerarlo como un canal E1 satelital, en este caso también se realizaron las pruebas con un 1,3 y 5% de pérdidas. para obtener el porcentaje de ganancia entre ambas implementaciones se realizó el mismo procedimiento utilizado para la comparación el la simulación mencionada anteriormente, teniendo como resultado los valores mostrados en la tabla 4.2.

% Perdida	TCP s-SACK	TCP c-SACK	TCP c-SACK/TCP s-SACK
1 %	25373 Kbps	31023 Kbps	1.22
3 %	9548 Kbps	11750 Kbps	1.23
5 %	6058 Kbps	7601 Kbps	1.25

Tabla 4.2: TCP con SACK vs TCP sin SACK

Analizando los resultados con las 3 probabilidades de pérdida vemos que la ganancia obtenida cuando se hace uso de la opción SACK es aproximadamente la misma en los 3 casos.

En las Figuras 4.7 y 4.8 vemos la gráfica de la comparación cuando se tiene un 1% de pérdida y la gráfica de la capacidad con la misma probabilidad de pérdida.

Con una probabilidad de pérdida de 3% se tienen los resultados mostrados en la gráfica 4.9 y en la gráfica 4.10 que muestran el comportamiento en el tiempo contra número de secuencia y la capacidad, respectivamente.

Por último se muestran gráficamente los datos obtenidos en las simulaciones cuando se tiene un 5% de pérdidas

4.4 Conclusiones

Considerando los resultados anteriormente mostrados, se puede concluir que cuando el retardo es pequeño, 10 ms, si la probabilidad de pérdida del canal es muy pequeña, aproximadamente un 1%, el uso de TCP con SACK no significa una ganancia considerable, sin embargo cuando la probabilidad de pérdida es mayor, arriba del 3%, el uso de la opción SACK nos da una ganancia significativa en la capacidad de bytes de datos efectivos por segundo que pueden ser transferidos.

Cuando el retardo es grande, en este caso considerado de 200 ms, los resultados nos muestran que el uso de la opción SACK nos presenta una ganancia, no importando la probabilidad de pérdida.

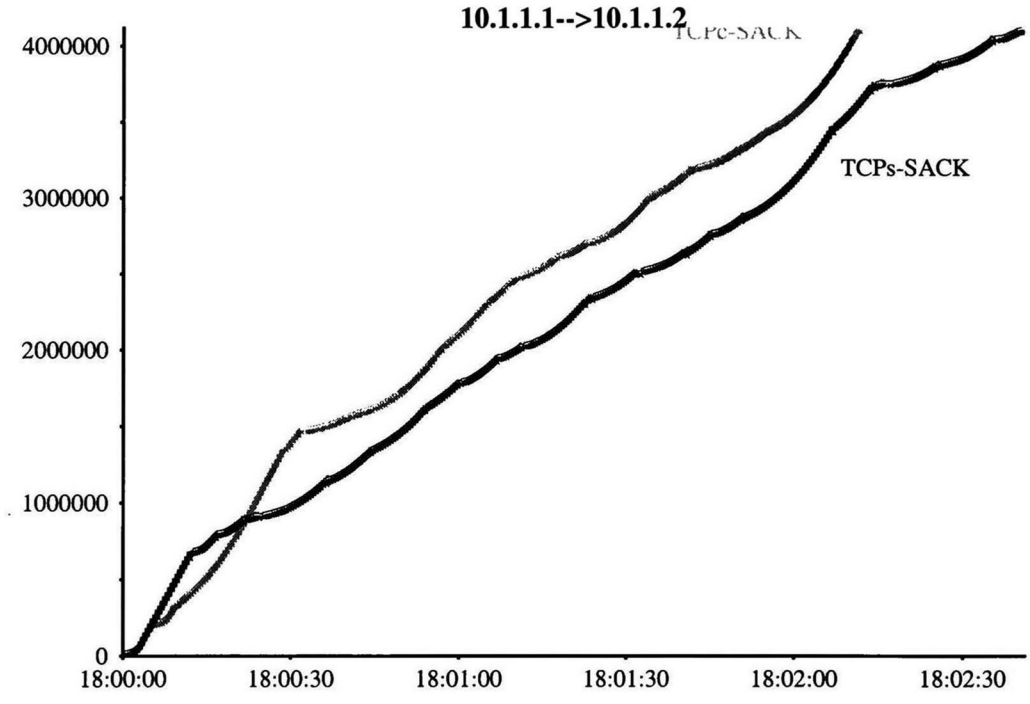

Figura 4.7: Comparación con 1% de pérdida

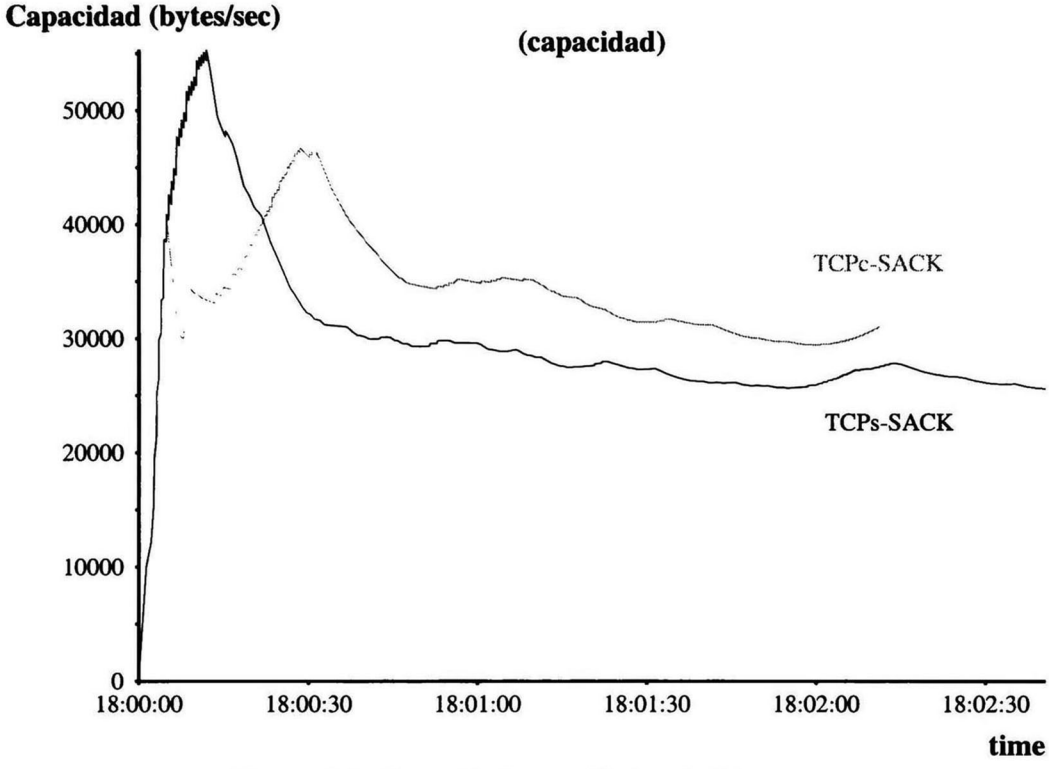

Figura 4.8: Capacidad con 1% de pérdida

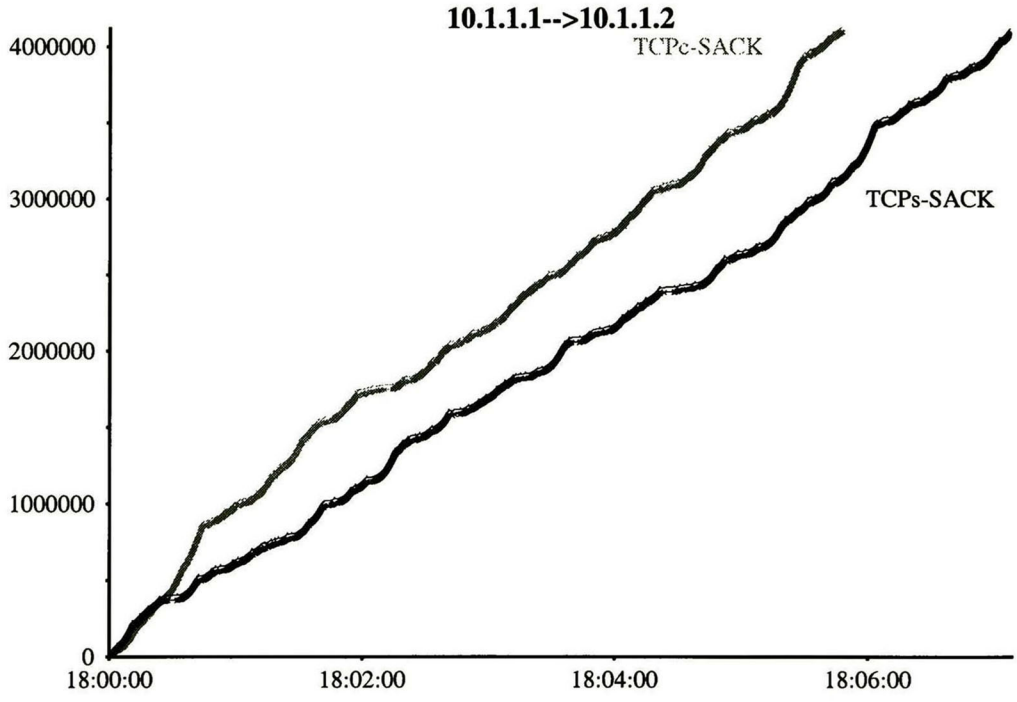

Figura 4.9: Comparación con 3% de pérdida

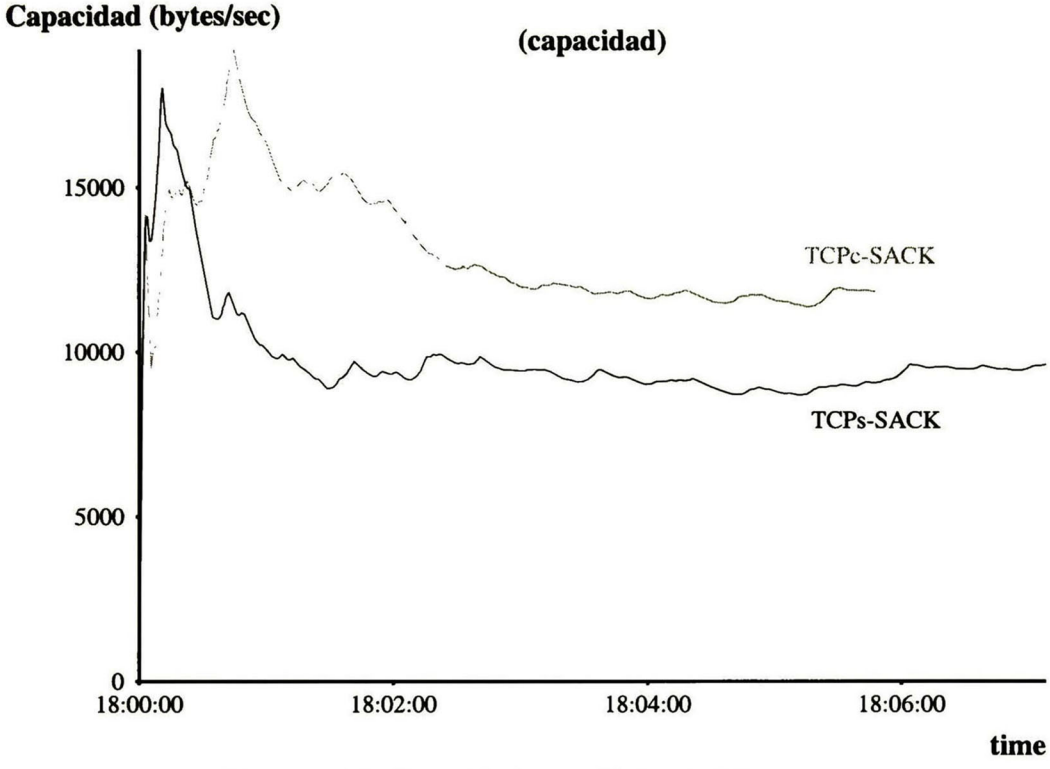

Figura 4.10: Capacidad con 3% de pérdida

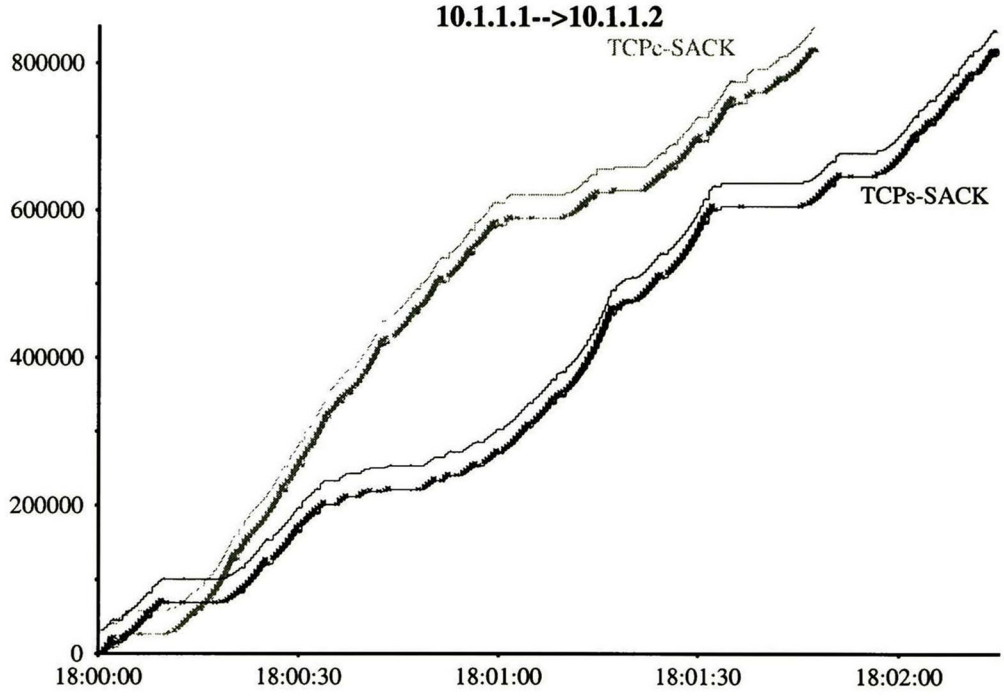

Figura 4.11: Comparación con 5% de pérdida

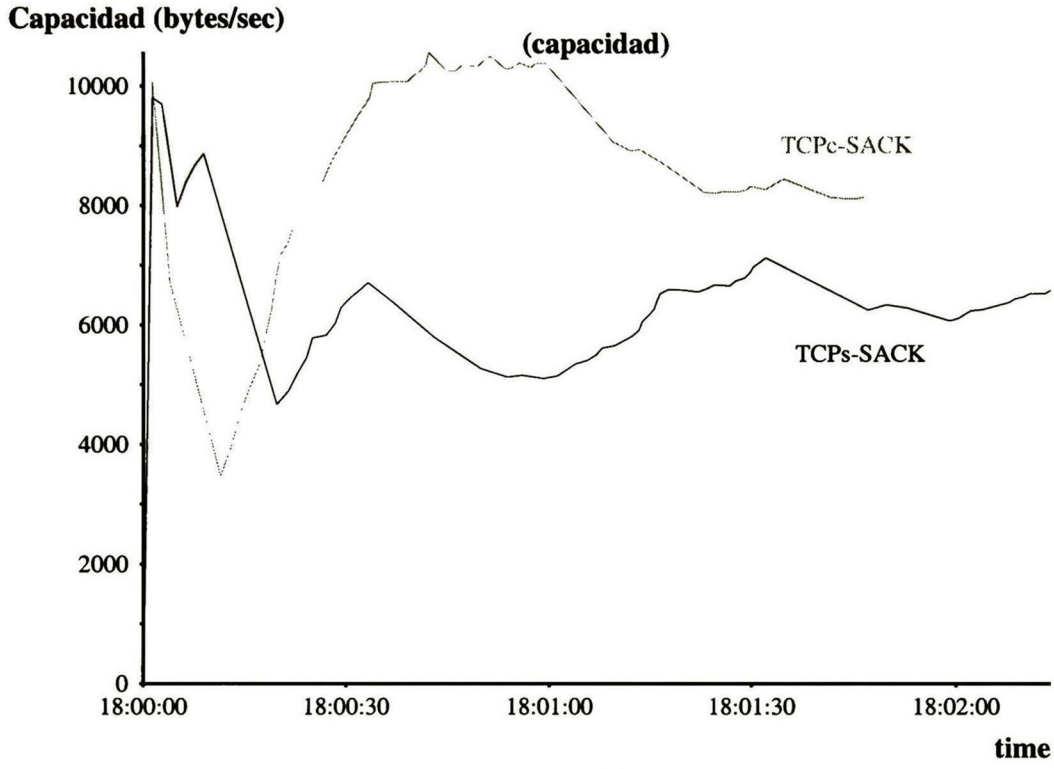

Figura 4.12: Capacidad con 5% de pérdida

Bibliografía

[1] B. Braden. *Requirements for Internet Hosts Communication Layers, RFC 1122.* DDN Network Information Center, Oct 1989.

[2] B. Bruyeron, R. Hemon. *Experimentation with TCP Selective Acknowledgment.* UCLA Internet Research Laboratory, 1997.

[3] D. Cheriton. *VMTP: Versatile Message Transaction Protocol, RFC 1045.* Stanford University, 1988.

[4] M. Clark, D. Lambert and L. Zhang. *NETBLT: A Bulk Data Transfer Protocol, RFC 998 MIT.* DDN Network Information Center, 1987.

[5] D. E. Comer. *Internetworking with TCP/IP vol I: Principles, Protocols, and Architecture.* Prentice Hall, third edition, 1995.

[6] C. Huiteman and I. Valet. *An Experiment on High Speed File Transfer using Satellite Links.* 7th Data Communication Symposium, Mexico, 1981.

[7] R. Jacobson, V. Braden. *TCP Extensions for Long Delay Paths, RFC 1072.* DDN Network Information Center, Oct 1988.

[8] R. Jacobson, V Braden and D. Borman. *TCP Extensions for High performance, Draft.* DDN Network Information Center, Feb 1997.

[9] V. Jacobson. *Modified TCP Congestion Avoidance Algorithm.* end2end-interest mailing list, ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail, Apr 1990.

[10] V Jacobson. *Congestion Avoidance and Control.* Computer Communication Review, vol 18 no. 4, ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z, Aug 1988.

[11] J.Postel. *Internet Protocol, RFC 791.* DDN Network Information Center, Sep 1981.

[12] J.Postel. *Transmission Control Protocol, RFC 793.* DDN Network Information Center, Sep 1981.

[13] F. J. Mathis M., Mahdavi J. and R. A. *TCP Selective Acknowledgment Option, RFC 2018.* DDN Network Information Center, 1996.

[14] R. Stevens. *TCP/IP Ilustrated vol 1, The Protocols.* Addison-Wesley Professional Computing Series, 1994.

[15] B. Strayer, T. Dempsey and A. Weaver. *XTP - The Xpress Transfer Protocol.* Addison-Wesley publishing Company, 1992.

[16] R. Velten, D. Hinden and J. Sax. *Reliable Data Protocol, RFC 908.* DDN Network Information Center, 1984.

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
UNIDAD GUADALAJARA

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, aprobó la tesis: "Implementación de la opción SACK en el protocolo TCP" del señor Marco Antonio Mora Arvizo el día 20 de Agosto de 1999.

Dr. Manuel Edgardo Guzmán Rentería
Investigador Cinvestav 3A
CINVESTAV DEL IPN
Guadalajara.

Dr. Deni Librado Torres Román
Investigador Cinvestav 2 A
CINVESTAV DEL IPN
Guadalajara.

M. en C. Joaquín García Luna Bustamante
Gerente de cuenta.
CISCO Systems de México, S.A. de C.V.