SCTP robusto a retransmisiones espúreas

Patricio O. Barletta
pbarletta@dc.uba.ar

Claudio E. Righetti [*]
claudio@dc.uba.ar

Paulo G. Veiga
Departamento de Computación, FCEN UBA
Ciudad de Buenos Aires, Argentina
pveiga@dc.uba.ar

Abstract

The objective of this paper is to contribute to the analysis of degradation of performance in realible transport protocols in the presence of global networks congestion assuming, as the reference model, the case of the SCTP (Stream Control Transmission Protocol).
Beyond the question of congestion, this study addresses packages reordering and *round-trip-time* abrupt changes that are very common in wireless networks. With the current congestion control mechanisms, these underlying network behaviors significantly degrade the performance of the transport protocols.
We propose an adaptive algorithm for the SCTP that will diminish the frequency of the "false timeouts" and the "false fast retransmits". These issues both degrade performance and impede the maximum utilization of resources (principally, the bandwidth and the buffers).We call our SCTP protocol extension "RR-SCTP".

Keywords: SCTP, reordering , RTT abrupt, fast retransmits , fast recovery

Resumen

El objetivo de este artículo es contribuir en los aspectos relacionados con la degradación de performance en los protocolos de transporte confiables, ante el fenómeno de congestión de las redes globales, en particular tomando como modelo de referencia SCTP (*Stream Control Transmission Protocol).*
Al problema de la congestión se suman el reordenamiento de paquetes y el cambio brusco del *round-trip-time*, muy común en redes wireless. Estos comportamientos de la red subyacente producen una seria degradación de la performance de los protocolos de transporte .
Se propone para SCTP un mecanismo adaptativo con el fin de disminuir la ocurrencia de los denominados falsos *timeouts* y falsos *fast retransmits*, que producen degradación en performance y la subutilización de recursos, principalmente de ancho de banda y buffers. La extensión del protocolo SCTP que incluimos la denominamos "*RR-SCTP".*

Palabras claves: SCTP, reordering , RTT abrupto, fast retransmits , fast recovery

[*] Grupo de Redes y Sistemas Globales Escalables.

1 INTRODUCCION

Son anomalías frecuentes el reordenamiento de paquetes y el cambio brusco del round-trip-time (RTT) en las redes IP. Estos comportamientos de la red subyacente producen retransmisiones espúreas que llevan a la degradación de performance de los protocolos de transporte confiable tales como TCP y SCTP. Si bien las estadísticas sobre reordenamiento de paquetes y su influencia en la perfomance de la red han sido causa de debate en el pasado actualmente se consideran las mismas un parámetro de QoS a tener en cuenta en la actual ingeniería de redes. Los primeros trabajos reportados [3] [15] coinciden en que el reordenamiento de paquetes es una propiedad de la redes. Dado que la granularidad del reordenamiento de paquetes excede el umbral de reconocimientos duplicados del algoritmo Fast Retransmit [1] puede causar que el emisor retransmita datos y dispare el mecanismo de control de congestión innecesariamente. Los enlaces inalámbricos presentan variaciones bruscas de RTT debido a los protocolos de nivel de enlace o alocación reactiva de recursos entre otras causas. Estas variaciones bruscas de RTT puede causar la expiración del Timeout de Retransmisión (RTO) produciendo retransmisiones y disparando el mecanismo de control de congestión. El protocolo de transmisión de control de flujo SCTP (Stream Control Transmission Protocol) [7] es un Standard de nivel de transporte propuesto por el IETF e implementando en varias distribuciones de Linux y versiones comerciales de UNIX. SCTP evolucionó de un protocolo de señalización telefónica para redes IP [16]. Provee una conexión full-duplex confiable. Dicha conexión se denomina asociación y provee un novedoso servicio de multihoming, que le permite a los hosts de una asociación tener múltiples direcciones IP. Ofrece multistreaming con lo cual se pueden tener múltiples flujos de datos independientes. SCTP provee un servicio de envío de datos orientado a mensajes a diferencia de TCP, que es orientado a bytes sin ninguna estructura de datos. Los mecanismos de control de congestión implementados son los utilizados en TCP. Proponemos para SCTP un mecanismo adaptativo con el fin de disminuir la ocurrencia de los denominados falsos timeouts y falsos fast retransmits

2 SCTP

La Figura 1 ilustra una generalización del formato del paquete SCTP [7]. Los paquetes siempre comienzan con una cabecera común. El resto consiste de uno o más *chunks*, bloques concatenados que contienen información de datos o control. Los *chunks* de control transfieren la información necesaria para el funcionamiento de una asociación, mientras que los *chunks* de datos llevan información de la capa de aplicación. SCTP provee un diseño flexible que permite agregar nuevos *chunks* de control.

Cada *chunk* contiene una cabecera que identifica su tipo, longitud y cualquier flag especial que el tipo requiera. SCTP posee la flexibilidad de concatenar diversos tipos de *chunks* en un mismo paquete. Una de las características principales de SCTP es *multihoming,* permite a los hosts utilizar más de una dirección IP en una asociación.

Durante una transferencia de datos, se utilizan principalmente dos tipos de *chunks*: a) *data chunk*, usado por el emisor, que contiene los datos. b) *sack chunk*, usado por el receptor, que contiene el acuse de la recepción de los *data chunks*. Todos los *data chunks* son identificados por un número secuencial llamado *transmission sequence number*, *TSN.*

Los *TSN* cuentan *data chunks* y no bytes enviados como lo hace el *sequence number* de TCP .

Cuando un *data chunk* arriba al receptor, este último debe enviar un *sack chunk* ,reportando la recepción. El *cumulative TSN ack* es utilizado de la misma manera que los *acknowledgement number* de TCP. El *ack* de SCTP acusa la recepción de todos los TSN hasta el valor del mismo,

mientras que en TCP se acusa recibo de la recepción de bytes, y no de los segmentos que los transportan.

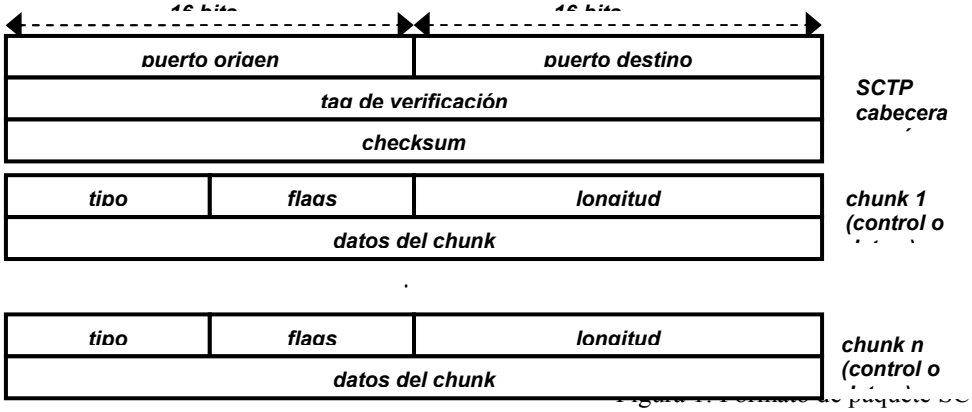

Figura 1: Formato de paquete SCTP

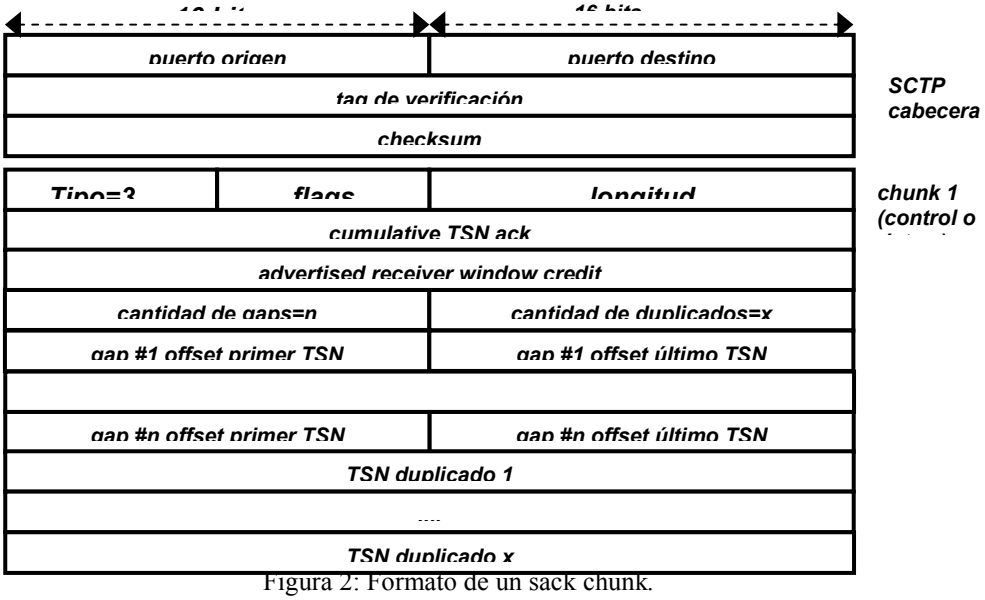

Figura 2: Formato de un sack chunk.

Los campos *gap ack blocks.* Estos son utilizados para acusar la recepción de datos fuera de orden, ya sea porque hay *data chunks que* pueden arribar al receptor en forma desordenada, o porque pueden perderse. Entonces, si un TSN cae dentro de un *gap ack block* significa que se ha recibido correctamente y que el emisor no necesita retransmitirlo incluso si el *cumulative TSN ack* no acusa su recepción. También puede observase en la Figura 2, que el *sack chunk* contiene una lista de *TSNs* llamada *duplicate TSN.* Por medio de esta lista, el receptor informa al emisor sobre la recepción de *TSNs* duplicados. SCTP es un protocolo de ventana deslizante que regula la tasa de transmisión de datos de la misma manera que lo hace TCP. El valor *advertised receiver window credit* es informado por el receptor al momento de iniciar la asociación y en cada *sack chunk* transmitido.
Para realizar control de congestión SCTP utiliza los mismos mecanismos de TCP: *slow start y congestion avoidance* [1] [18]. SCTP realiza s*low start* cuando el *cwnd* es menor o igual que el *ssthresh*. En contraste, las implementaciones de TCP poseen la libertad de elegir entre s*low start* y c*ongestion avoidance* cuando el *cwnd* es igual que el *ssthresh*. Durante s*low start*, el *cwnd* es incrementado únicamente si se cumplen las siguientes condiciones: que el *ack* recibido asiente nuevos datos, y que todo el *cwnd* este en uso antes de que el *ack* arribe.
De ser así, el *cwnd* es incrementado por el mínimo entre la cantidad de datos asentidos y el MTU (unidad máxima de transferencia). Las variables de *cwnd* y *sshthresh* deben mantenerse por cada una de las direcciones destino presentes en una asociación, con lo cual, los algoritmos de control de

congestión deben ejecutarse por cada una de ellas. Esto se debe a que diferentes direcciones destinos generalmente implican diferentes caminos, diferentes estados de congestión y diferentes *RTT*. Ante la ocurrencia de un t*imeout*, el emisor retransmite todos los *data chunks* que no hayan sido asentidos. Todos los *data chunks* que fueron asentidos por medio de *gap ack blocks* no serán retransmitidos, evitando la retransmisión de los *data chunks* que fueron recibidos fuera de orden.
El calculo del *timeout* de retransmisión (*RTO*) se realiza con el algoritmo utilizado en TCP [1] [10]. El mismo debe realizarse por cada destino que interviene en la asociación, ya que en cada uno de ellos podrían observarse distintos valores del *RTT*.
A lo igual que TCP tiene mecanismo de *fast retransmit* el cual consiste en optimizar el throughput ante pérdidas aisladas. El mecanismo que utiliza el receptor para informar la recepción de paquetes fuera de orden es a través de los *gap ack blocks*. Estos bloques forman parte del *sack chunk* y el receptor está obligado a notificarlos en caso de poseer en sus buffers *TSNs* no contiguos. Es decir, siempre que el receptor posea huecos debido a la recepción de *data chunks* fuera de orden, se ve obligado a informar los mismos en cada *sack chunk* que envía. Cuando un emisor recibe un *sack chunk* que indica que algún TSN ha sido perdido, éste debe esperar por tres reportes más que indiquen la pérdida del mismo TSN (por medio de *sack chucks* subsecuentes), antes de tomar alguna acción. El conteo de los reportes de *TSNs* faltantes se realiza utilizando el algoritmo denominado HTNA (*Highest TSN Newly Acked*) [19].

3 TRABAJOS RELACIONADOS

Mediante la utilización de los reportes de paquetes duplicados (*DupTSN*) de SCTP y presentes como la extensión DSACK [8] en TCP, se propone en [6] un mecanismo para ambos protocolos que utiliza estos reportes para detectar retransmisiones espurias. Se mantiene un seguimiento de los paquetes que fueron retransmitidos durante un evento de retransmisión, asociando a cada uno una marca de si fue reportado o no como duplicado. Si todos los paquetes retransmitidos en una ventana son reportados como duplicados, estamos en condiciones de afirmar que la retransmisión fue espuria. Deja abierta la forma de cómo almacenar y mantener la lista de estos paquetes retransmitidos. En el caso de SCTP, no detalla la implementación del mismo en esquemas multihoming. El núcleo de nuestro algoritmo de detección se basa en los principios delineados en éste trabajo, presentando una versión funcional del mismo que soluciona todos los problemas antes descriptos.
Uno de los primeros trabajos de detección para TCP es el algoritmo Eifel [13] que utiliza la opción de timestamps [11] como mecanismo explicito de señalización para detectar retransmisiones y timeouts espurios. Existen varios inconvenientes que hacen que EIFEL no pueda implementarse fácilmente en SCTP. Por un lado, no existe un mecanismo similar a la extensión de timestamps para TCP. Por otro lado la capacidad que posee SCTP de enviar más de un *data chunk* en un paquete, puede generar falsas detecciones. Por último, EIFEL no fue diseñado para funcionar en un esquema de multihoming ya que esta característica no esta presente en TCP. En [12], que propone una extensión de timestamps para SCTP, con el fin de lograr una versión símil de EIFEL.
Una ventaja de EIFEL, frente a los mecanismos de detección basados en los reportes de duplicados, es que puede detectar retransmisiones espurias más rápidamente. La desventaja es el overhead que corresponde a los 12 bytes de la opción de timestamp que debe ser incluida en cada paquete transmitido.
Otro algoritmo es F-RTO [17] que modifica ligeramente el comportamiento del emisor TCP inmediatamente después de un *timeout*, y luego observa el patrón de los *acks* que recibe. En base a este patrón infiere si la retransmisión fue innecesaria. La ventaja de este algoritmo es que solo necesita ser implementado en el emisor TCP y no introduce ningún overhead. Sin embargo, este

algoritmo es una heurística que no es robusta a ciertas patologías en la red como ser el reordenamiento de ciertos paquetes claves.
Hay trabajos que intentan disminuir el impacto del problema de reordenamiento de paquetes, de forma tal de evitar futuros *fast retransmits* espurios. El mecanismo más simple propone el incremento del *dupthresh* en una constante K [5] después de que una retransmisión espuria es detectada. Un enfoque diferente es el propuesto por [15], donde el *fast retransmit* puede ser demorado por cierta cantidad de tiempo después de que *dupthresh* reportes de duplicados se han recibido. Otra estrategia se basa en mantener un histograma para almacenar las longitudes de reordenamiento que sufren los paquetes de una conexión [20]. En vez de actualizar el *dupthresh* directamente utilizando alguna medida estadística, la elección de este valor se basa en la utilización de un percentil que se adapta por medio de una función de costos.

4 RR-SCTP

RR-SCTP detecta y corrige los problemas causados por el reordenamiento de paquetes y los cambios abruptos de *RTT* con el fin de mejorar el throughput de una asociación. Consta de tres algoritmos independientes: detección de Fast Retransmit y Timeout espurios, de Recovery y el de adaptación dinámica del dupthresh. El algoritmo de detección es el encargado de detectar aquellas retransmisiones que hayan sido espurias, ya sea por Fast Retransmit o por Timeout, para que luego el algoritmo de Recovery puede reestablecer la tasa de transferencia. Por último, el algoritmo de adaptación dinámica del dupthresh es el encargado de adaptar el valor del *dupthresh* en base a mediciones que realiza en la red, para poder así evitar futuras retransmisiones espurias.

4.1 Deteccion de Fast Retransmit y Timeout espurios

El algoritmo de detección se basa en la utilización del reporte de duplicados contenido dentro de los *sack chunks*. Esta información que SCTP provee no es utilizada en las implementaciones actuales. Cuando en una asociación SCTP ocurre un *Timeout*, se deben reenviar todos aquellos paquetes que no hayan sido asentidos previamente. En cambio, durante *fast retransmit* se retransmiten aquellos *TSNs* que hayan sido reportados como faltantes más de tres veces y que no hayan sido retransmitidos por *fast retransmit.* Definimos como evento de retransmisión la ocurrencia *de un fast retransmit o un timeout.* El algoritmo de detección propuesto lleva el registro de los *TSNs* retransmitidos utilizando una lista, que denominamos *TSN* ***retransmision list (Trl).*** Si para cada *TSN* contenido dentro de la *Trl* se recibe un reporte de duplicado, se puede concluir que todos los *TSNs* originales llegaron a destino, al igual que sus retransmisiones.
Por lo tanto, el *evento de retransmisión* fue espurio ya que ninguno de los paquetes originales fue perdido en tránsito. Se limita la detección a si el último evento que causó la retransmisión fue o no espurio. Nuestro algoritmo tiene en cuenta dos premisas: i) necesidad de conocer cuales son los *TSNs* correspondientes a la última retransmisión ii) se deben mantener en la *trl* todos aquellos *TSNs* para los cuales aún es posible que se reciba un reporte de duplicado.
Para conocer que *TSNs* corresponden a la última retransmisión, alcanza con agregarle a cada uno, una marca que indique si el mismo fue agregado en la *trl* durante el último *evento de retransmisión*. Con ésta modificación, al llegar un reporte de duplicado para un *TSN*, si existe más de una copia del mismo, se elimina primero alguna copia que no corresponda a la última retransmisión. Cuando la *trl* no contenga *TSNs* correspondientes a la última retransmisión, significa que la misma fue espuria. En cuanto a la segunda premisa, el mantener un histórico de los *TSNs* retransmitidos introduce un nuevo contratiempo relacionado con el control de crecimiento de la *trl*. Todo *TSN* para el cual no se reciba reporte de duplicado, no será removido de la misma, con su consecuente crecimiento. Para evitar que la *trl* aumente de tamaño indefinidamente, se eliminan todos los *TSNs* menores o iguales

que el *Cumulative TSN Ack Point* luego de un *evento de retransmisión*. Estos *TSNs* ya asentidos en forma acumulada no serán retransmitidos, con lo cual no influirán en la detección del último *evento de retransmisión* o de los posibles eventos posteriores.
En SCTP puede ocurrir un *fast retransmit* estando en *fast recovery*. En éste caso la única diferencia que existe es que no se modifican las variables de los algoritmos de control de congestión. Hasta tanto no se salga de *fast recovery*, todos los *fast retransmits* que ocurran serán interpretados como un único *evento de retransmisión*. Para un correcto funcionamiento del algoritmo de detección en asociaciones multihomed, se debe ejecutar el mismo por dirección destino. Para esto es necesario mantener una *trl* por cada una de ellas.

4.2 Recovery

Ante un *timeout,* el emisor SCTP dispara los mecanismos de control de congestión que produce reducción del throughput. Esto se logra disminuyendo los valores de *cwnd* y *ssthresh*. Si el *timeout* fue generado por cambios bruscos del *RTT*, se estará penalizando el throughput de la asociación en forma innecesaria. El costo producido por un *timeout* consta de tres componentes: Un período de inactividad, *slow start*, y un crecimiento lineal más allá del *ssthresh*, que sería la mitad del *cwnd* antes del *timeout*.
El mismo problema se presenta con *fast retransmit*, que si bien causa una reducción del throughput, no es tan costosa como la de un *timeout*. Ante una simple pérdida que dispara el algoritmo de *fast retransmit*, se reduce el *cwnd* a la mitad, y el *ssthresh* toma el valor del *cwnd* reducido, entrando en *congestion avoidance*. Una solución posible es restaurar los valores del *cwnd* y *ssthresh* a los valores previos al *timeout* o *fast retransmit* que causó su reducción. Sin esto, alcanzar el valor previo del *cwnd* y el throughput anterior podría llevar varios *RTTs*. Con el fin de implementar el *Recovery* será necesario, ante un *evento de retransmisión*, almacenar los valores de *cwnd* y *ssthresh* previos a la reducción sobre la dirección que ocurrió el evento. La idea del algoritmo de *Recovery* consiste en restaurar a los valores previos de *cwnd* y *ssthresh*, tan pronto se detecta que un *evento de retransmisión* fue espurio. Hemos tomado solución propuesta en [15] adaptada para el algoritmo. Consiste en actualizar el *ssthresh* al valor previo del *cwnd* para así entrar en *slow start* y comenzar a probar la red nuevamente, luego del período de inactividad. De esta manera, en un par de *RTTs*, se puede alcanzar el valor anterior del *cwnd*. Si el valor del *cwnd* actual supera el valor que poseía anteriormente, no se realiza ningún ajuste a las variables.
Sin el algoritmo de *Recovery*, el valor del *cwnd* crecerá en forma exponencial hasta el *ssthresh*, que es igual a la mitad del valor del *cwnd* previo. Luego crecerá en forma lineal. Con el algoritmo de *Recovery*, el valor del *cwnd* crecerá en forma exponencial hasta el *ssthresh*, pero en este caso el valor es igual al *cwnd* previo. El crecimiento del valor del *cwnd* en forma exponencial comenzará tan pronto se detecte que el *evento de retransmisión* fue espurio. Entonces, mientras más pronto esto se detecte, menor la cantidad de *RRTs* necesarios para alcanzar el throughput anterior.

4.3 Adaptación dinámica del dupthresh

La decisión de elegir el valor del *dupthresh* no es trivial y está sostenida en la experiencia acumulada en TCP. Este valor tiene influencia en la performance del protocolo, cuando la red presenta reordenamiento. En SCTP dispara el mecanismo de *fast retransmit* cuando la cantidad de reportes de duplicados es *dupthresh* = 4. Para que un emisor SCTP evite *fast retransmits* espurios luego de que un paquete es reordenado, el *dupthresh* debe ser mayor o igual que el número de reporte de duplicados que fueron generados debido a reordenamiento para un *TSN*. Definimos como *contador de* r*eporte de faltantes (crf)* de un *TSN* a la cantidad de veces que el mismo fue reportado

como faltante, siguiendo el algoritmo de *HTNA*. Este valor es utilizado para disparar *fast retransmit*, y es mantenido por cada *TSN* transmitido en una asociación.
La performance de SCTP será pobre en redes que presenten *crf* mayor o igual que cuatro. En este caso, SCTP entrará en *fast retransmit* reiteradamente.
SCTP retransmitirá todos los paquetes con *crf* mayor que tres. Ademas se producirá una reducción del *cwnd* y *ssthresh*. Estos factores tienen como consecuencia directa un período de inactividad de la asociación, donde no se transmiten datos nuevos, causando una gran reducción del throughput.
Por esta razón es conveniente que el *dupthresh* no sea un valor constante, sino que se adapte dinámicamente a las condiciones de reordenamiento presentes en la red.
La elección de un *dupthresh* pequeño en relación al *crf* medio de la red produce que una asociación entre en *fast retransmit* cuando no debieron hacerlo, ya que los paquetes solo fueron demorados o se reordenaron. Por el otro, la elección de un *dupthresh* grande, no permitirá la recuperación frente a pérdidas simples por medio de *fast retransmit*, lo que dará lugar a un *timeout*, con el costo que esto tiene asociado.
Se incluye en el algoritmo un histograma que contendrá el *crf* de cada uno de los paquetes transmitidos tales que su *crf* sea mayor o igual que cuatro. El histograma de *crfs* sumariza la distribución del reordenamiento experimentado para una asociación La inserción en el histograma del *crf* de un *TSN* que fue retransmitido, deberá demorarse hasta la recepción de un reporte de duplicado para el mismo, pues debemos asegurarnos que el *TSN* no se haya perdido, de lo contrario dicho valor estará dando información acerca de un reordenamiento que nunca pudo haber ocurrido. Dada la imposibilidad de saber a qué transmisión corresponde la recepción de un reporte de duplicado, sólo se insertan en el histograma los *crfs* de los *TSNs* correspondientes a la última retransmisión. En vez de variar el *dupthresh* directamente variamos el percentil utilizado para calcular el *dupthresh* [20]. Denominamos a este valor *pef* (Percentil para Evitar *fast retransmit*). Incrementar el *pef* incrementará el *dupthresh*, mientras que decrementar el *pef* decrementará el *dupthresh*. La variación del *pef* se realiza en función del costo relativo de un *fast retransmit* o de un *timeout*.

5 SIMULACION

Se presentan las mejoras que se obtuvieron al incorporar el mecanismo RR en SCTP. Las simulaciones se llevarán a cabo utilizando el simulador de red *ns-2* [4]. Se desarrollo una extensión del mismo para generar reordenamientos con distribución normal, uniforme, exponencial, etc.
En todas las simulaciones, el máximo de ventana MW anunciado por el receptor es de 50 paquetes, con un MTU fijo de 1500 bytes.

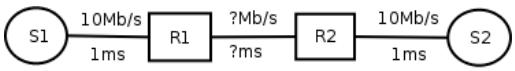

Figura 3

En la figura 3, S1 y S2 son nodos que no son multihomed, y R1 y R2 son routers. Los eventos de reordenamiento y pérdida de paquetes son introducidos en el link (R1,R2), cuyo delay y ancho de banda varía según la simulación. Con el fin de tener un control preciso de la tasa de pérdida del link (R1,R2), se configuró la capacidad del link a S= MW/*RTT* donde *RTT* es el *round-trip-time*.De esta forma, cuando no se introduce reordenamiento o pérdida de paquetes, SCTP alcanza el máximo throughput, con S como cota superior. Para las simulaciones con nodos multihomed se representa en la figura 4 la topología adoptada. En este caso el reordenamiento y la pérdida de paquetes están simulados por el enlace (S1_0, S2_0). La dirección primaria de las asociaciones es S2_0. El enlace (S1_1, S2_1) no posee tasas de pérdida ni reordenamiento. La retransmisión de paquetes se realiza por el enlace secundario, tal como se recomienda en la especificación de SCTP.

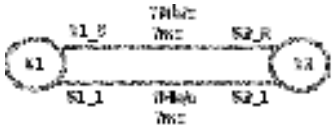

Figura 4 : nodos multihomed

Durante las simulaciones determinamos que las conclusiones no variaban frente a distintas distribuciones de reordenamiento.
En la Figura 5 se presentan dos simulaciones de 10 segundos en la cual se introduce el reordenamiento del paquete 100 en "cuatro posiciones" en una asociación SCTP y RR-SCTP, respectivamente. Este reordenamiento desemboca en un Fast Retransmit espurio, que lleva en ambos casos a la retransmisión del *TSN* 100 y la consecuente reducción del *ssthresh* y el *cwnd*.
Comparando ambas simulaciones se puede observar que la cantidad de paquetes trasmitidos por RR-SCTP es superior a la cantidad transmitida por SCTP. En el caso de SCTP, la asociación logró enviar hasta el *TSN* 320, mientras que la asociación RR-SCTP envió hasta el *TSN* 400. Esto se debe principalmente a la intervención del algoritmo de Recovery, que logra disminuir el impacto causado por este *Fast Retransmit* espurio. En el tiempo 6.5 aproximadamente, el algoritmo detecta que la transmisión fue espuria, permitiendo una rápida restauración de los valores de las variables de congestión.

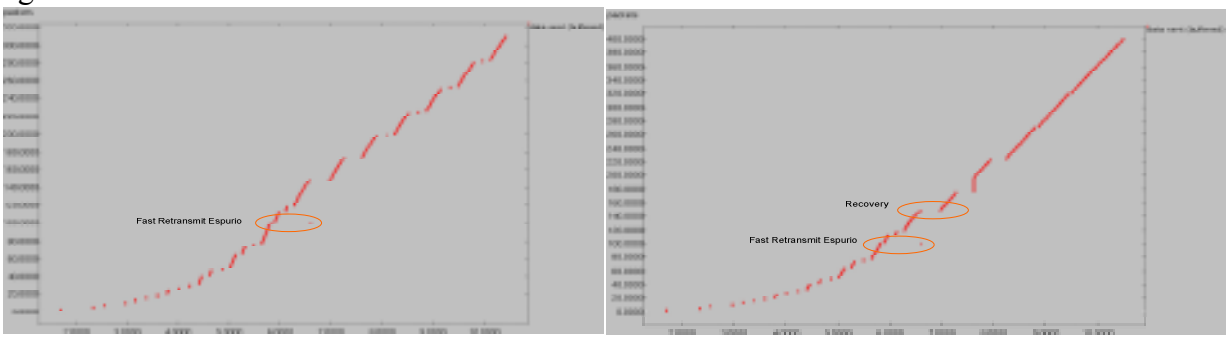

Figura 5: Paquetes transmitidos en función del tiempo en una asociación frente a la ocurrencia de un Fast Retransmit espurio a causa del reordenamiento del TSN 100 a) asociación SCTP b) asociación RR-SCTP.

En la Figura 6 se muestran dos simulaciones de 16 segundos en la cual se introduce un cambio del RTT al doble del valor actual, en el tiempo 5. Este cambio abrupto del RTT tiene como consecuencia la ocurrencia de un Timeout espurio. Nuevamente al igual que en el caso anterior, el desempeño de RR-SCTP es superior al de SCTP debido a la intervención del modulo de Recovery, en el tiempo 8 aproximadamente. En este caso, la asociación SCTP pudo enviar hasta el *TSN* 400, mientras que la asociación RR-SCTP envió hasta el *TSN* 450.

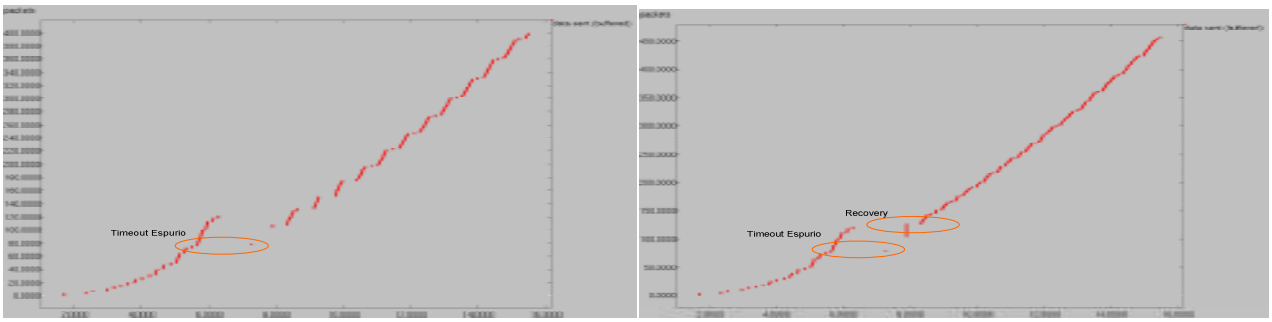

Figura 6: Paquetes transmitidos en función del tiempo en una asociación frente a la ocurrencia de un Timeout debido a un cambio abrupto del RTT a) SCTP b) RR-SCTP

5.1 Reducción de Fast Retransmit espurios

La adaptación dinámica del *dupthresh* sumado a la restauración de los valores de *cwnd* y *ssthresh* ante *fast retransmits* espurios, mejora la performance de una asociación SCTP.
Se han simulado demoras de paquetes que sigue una distribución normal con una media de 30ms y un desvío estándar de 15ms.
Como puede observarse en la Figura 7 cuanto es mayor tasa de reordenamiento peor el desempeño que presenta SCTP.

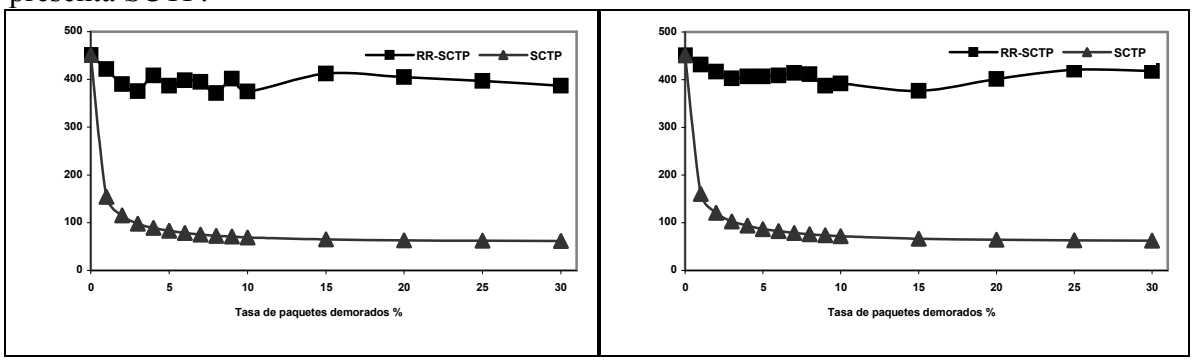

Figura 7: Throughput (pkts/seg) vs tasa de paquetes demorados a) sin multihome b)con multihome, 50 ms de delay de propagación.

Esto se debe al aumentar la tasa de paquetes reordenados, mayor es la tasa de *fast retransmits* que sufre SCTP con relación a la cantidad de paquetes transmitidos, con una consecuente pérdida de rendimiento innecesaria dado que todos los *fast retransmits* fueron espurios. Al contrario, RR-SCTP mantiene estable el throughput independientemente del reordenamiento. Esta mejora se debe principalmente a la adaptación dinámica del *dupthresh* comparando la tasa de *fast retransmits* sufridos entre ambos algoritmos. A medida que transcurre el tiempo, el histograma comienza a llenarse con los reordenamientos presentados en la red. Esto permite, por medio del percentil, adaptar el *dupthresh* a un valor con el que el protocolo presentará un mejor desempeño. Adicionalmente, la detección de *fast retransmits* espurios, con la consiguiente restauración de los valores de la ventana de congestión, mejora el throughput de la asociación.
Calculando la tasa de *fast retransmits* detectados, puede concluirse que el algoritmo de detección posee una efectividad superior al 65%.
Con tasas de reordenamiento menores 10 %, el algoritmo de detección presenta una efectividad superior al 90%.

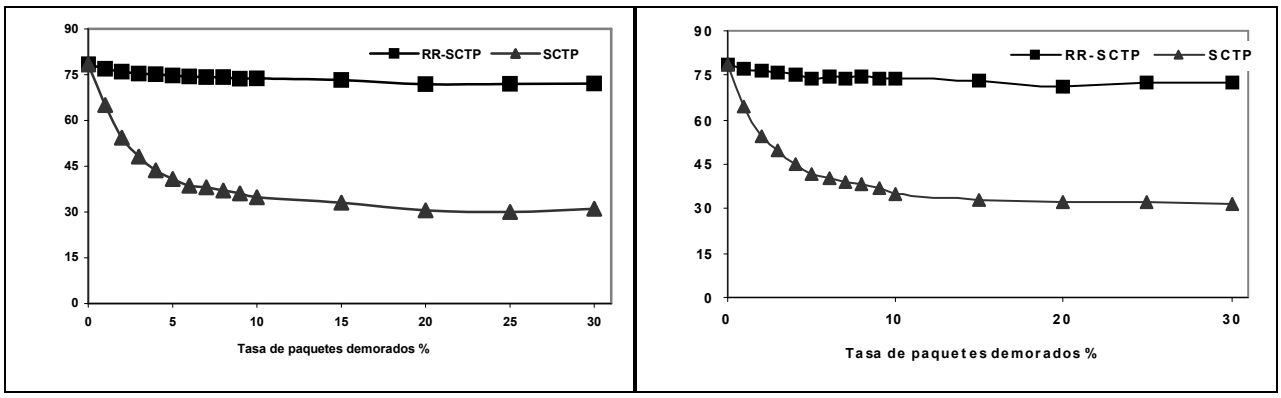

Figura 8: 300 ms de delay de propagación, ,a) sin multihome b) con multihome

Las Figuras 8 corresponde a simulaciones con los mismos parámetros que en el caso anterior pero variando el delay del enlace de 50 ms a 300 ms. La característica común es que al aumentar la tasa

de reordenamiento se produce una gradual degradación de la performance en SCTP, mientras que en el caso de RR-SCTP el desempeño se mantiene constante.

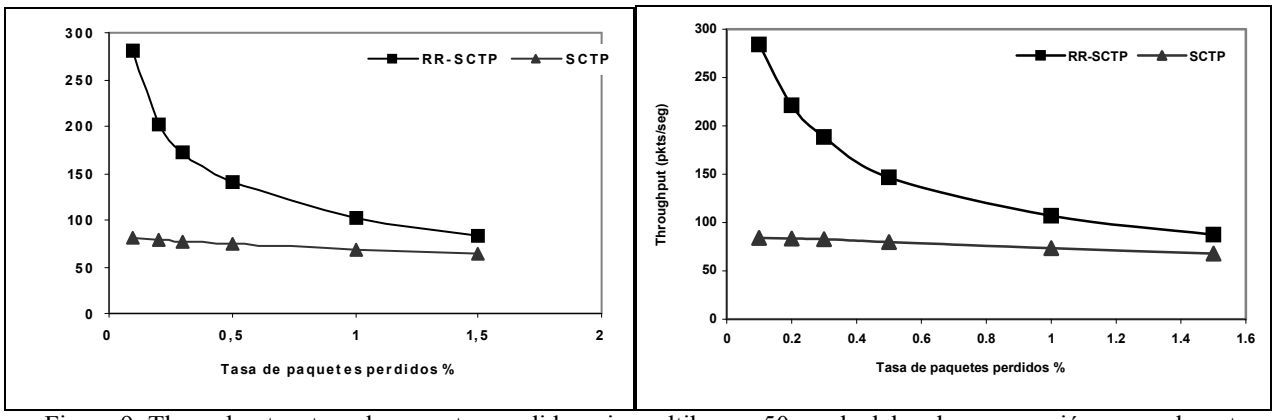

Figura 9 :Throughput vs tasa de paquetes perdidos, sin multihome, 50 ms de delay de propagación, normalmente distribuido, media 30 ms y desvío estándar 15 ms, 5% de reordenamiento.

Analizando la Figura 9 puede observarse nuevamente, que RR-SCTP tiene un mejor desempeño que SCTP frente a una tasa de paquetes perdidos de 0.1% a 1.5%.

En el caso de RR-SCTP, un *fast retransmit* puede ser identificado como espurio sólo cuando no ocurren pérdidas en esa ventana de transmisión. A medida que la tasa de pérdida aumenta, es altamente probable que ocurra al menos una pérdida en la ventana. Como consecuencia de esto, la cantidad de *fast retransmits* espurios decrece rápidamente, y la diferencia en el throughput entre RR-SCTP y SCTP es menor. Considerando que las tasas de errores en enlaces de Internet rondan el 0.02% [15], con tasas de errores inferiores al 0.1 %, se obtiene una mejora en la performance de más del 200%. Como es de esperarse, al aumentar la tasa de pérdida de paquetes, también aumenta la cantidad de *fast retransmits* y *timeouts*. Sin embargo, el dinamismo del *dupthresh* causa que la cantidad de *fast retransmits* en RR-SCTP sea inferior a la de SCTP, debido a que este último no es capaz de detectar que algunos de esos *fast retransmits* fueron espurios, para luego poder evitarlos.

Las simulaciones de la Figura 10 no muestran que frente a un alto valor de delay del enlace, la mejora en el desempeño de RR-SCTP sobre SCTP es menor. Esto es consecuencia directa de la pérdida de al menos un paquete en la ventana de transmisión, que afecta considerablemente el throughput de una asociación sobre un enlace con alto delay

Finalmente puede observase en la Figura 11, bajo todas las distribuciones simuladas, RR-SCTP mantiene estable el throughput.

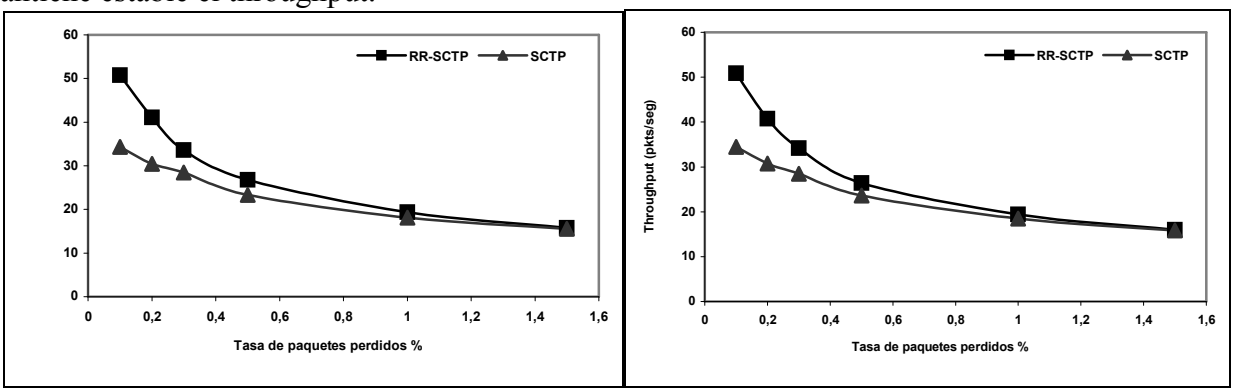

Figura10: Throughput vs tasa de paquetes perdidos, sin multihome, 300 ms de delay de propagación, normalmente distribuido, media 30 ms y desvio estándar 15 ms, 5% de reordenamiento.

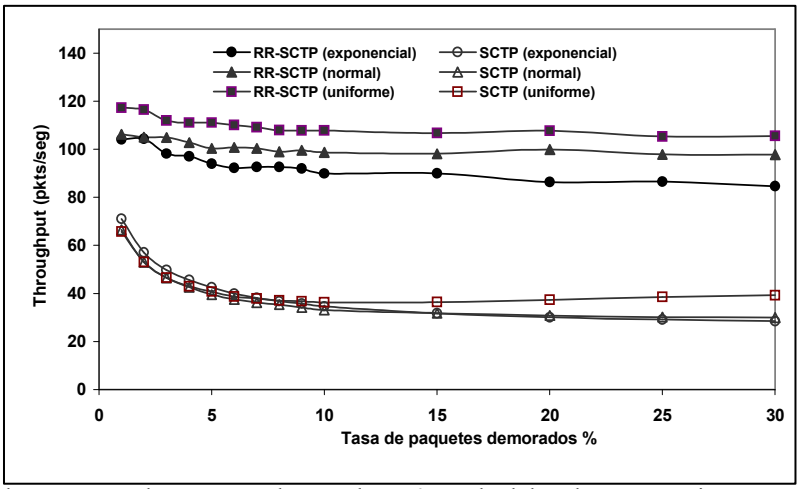

Figura 11: Throughput vs tasa de paquetes demorados, 50 ms de delay de propagación, tasa de pérdida de 0.2 %.

6 CONCLUSIONES y TRABAJO FUTURO

En este articulo presentamos un mecanismo para SCTP compuesto por tres algoritmos que hacen a SCTP más robusto frente a retransmisiones espúreas. El algoritmo de detección le permite a un emisor distinguir si un *timeout* o un *fast retransmit* fue espurio, basándose en los reportes de duplicados. Los paquetes retransmitidos son almacenados de forma tal de poder verificar, frente a la recepción de duplicados, si los mismos fueron retransmitidos innecesariamente.
El algoritmo de Recovery restaura los valores de las variables de control de congestión cuando un *evento de retransmisión* fue detectado como espurio. Para esto, dichas variables deben ser almacenadas cuando ocurre un *timeout* o un *fast retransmit*. Esto permite a un emisor que redujo sus ventanas en forma innecesaria, recuperar la tasa de transmisión más rápidamente.
El tercer algoritmo se encarga de ajustar el *dupthresh* en base a información sobre el reordenamiento existente en la asociación, con el fin de evitar futuros *fast retransmits* espurios. La elección del *dupthresh* se basa en la utilización de una función de costos, de forma tal de maximizar el throughput de la asociación. El protocolo SCTP con el mecanismo propuesto lo denominados RR-SCTP. Ninguno de los cambios en SCTP que se proponen en este artículo genera overhead adicional sobre la asociación cuando la misma no experimenta ninguna de las patologías que enfrenta el mecanismo. La comparación de rendimiento en el simulador ns-2 nos confirma que RR-SCTP tiene un mejor desempeño que SCTP en todos los casos. Esta diferencia es aún superior en enlaces con alta tasa de reordenamiento y baja tasa de pérdida, tanto en asociaciones multihomed como en asociaciones no multihomed. Otra alternativa a la propuesta a nuestro algoritmo de detección para ambientes multihoming, es poseer una *trl* única por asociación y almacenar cual era la dirección primaria en el momento que se produce el *evento de retransmisión*.
En función de los resultados obtenidos actualmente estamos desarrollando la implementación de RR-SCTP, con foco en la disminución del uso de memoria que este insume.

REFERENCIAS

[1] Allman, M., Paxson, V., Stevens, W.: TCP Congestion Control. *RFC 2581,* April 1999.

[2] Allman, M., Floyd, S.: Enhancing TCP's loss recovery using Limited Transmit. *RFC 3042,* January 2001.

[3] Bennet, J., Partridge, C., Shectman, N.: Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networking* , December 1999.

[4] http://www.isi.edu/nsnam/ns/

[5] Blanton, E., Allman, M.: On making TCP more robust to packet reordering. *ACM Computer Communicaction Review,* 32 (1), January 2002.

[6] Blanton, E., Allman, M.: Using TCP Duplicate Selective Acknowledgement (DSAcks) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions. *RFC 3708*, February 2004.

[7] Caro Jr., A., Shah, K., Iyengar., J., Amer, P., Ladha, S., Heinz, G.: SCTP: A Proposed Standard for Robust Internet Data Transport. IEEE Computer, 36(11):56-63, November 2003

[8] Floyd, S., Mahdavi, J., Mathis, M., Podolsky, M. An Extension to the Selective Acknowledgement (SACK) Option for TCP. *RFC 2883*, July 2000.

[9] Handley, M., Padhye, J., Floyd, S.: TCP Congestion Window validation. *RFC 2861*, June 2000.

[10] Jacobson, V., Karels, M.: Congestion Avoidance and Control. *ACM SIGCOMM,* Nov. 1988.

[11] Jacobson, Braden, Borman, D.: TCP Extensions for High Performance. *RFC 1323,* May 1992.

[12] Ladha, S.: Presenting the case of Eifel Algorithm for SCTP. http://www.eecis.udel.edu/~ladha/EifelSCTP-2003

[13] Ludwig, R., Meyer, M.: *The Eifel Detection Algorithm for TCP*. *RFC 3522,* April 2003.

[14] Ludwig, R., Gurtov, A.: The Eifel Response Algorithm for TCP. March 2004. http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-tcp-eifel-response-05.txt

[15] Paxson, V.: End-to-End Internet Packet Dynamics. *ACM SIGCOMM*, December 1997.

[16] Robison, H., Stewart, R., Morneault, K.: SIGNALING: Next-Generation Transport, http://www.cisco.com/en/US/about/ac123/ac114/ac173/Q2-04/nextgeneration.html

[17] Sarolahti, P.: F-RTO: An algorithm for detecting Spurious Retransmission Timeouts with TCP and SCTP. Febrero 2004.http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-tcp-frto-01.txt

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

[19] Stewart, R., Xie, Q., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., Paxson, V.: Stream Control Transmission Protocol. *RFC 2960,* October 2000.

[20] Zhang, M., Karp, B., Floyd, S., and Peterson, L.: RR-TCP: A Reordering-Robust TCP with DSACK, *in Proceedings of the Eleventh IEEE International Conference on Networking Protocols (ICNP 2003), Atlanta, GA,* November 2003.