Teoría de las Comunicaciones

Primer Cuatrimestre de 2013

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Tp3: conexiones

Grupo:

Integrante	LU	Correo electrónico
Matías Capello	006/02	matiascapello@gmail.com
Santiago Hernández	48/11	santi-hernandez@hotmail.com

Índice

Índice de figuras

Abstract

En este trabajo se implemento el cliente de un protocolo de sliding window con goback n simplex y se analizó la eficiencia del mismo en una red local.

Nuestro proposito es comprender como afectan al desempeño del protocolo las diversas variables del protocolo mediante la experimentación y el análisis de los resultados.

Los experimentos fueron llevados a cabo en una red local wifi variando el tamaño de la ventana de emisión y el tamaño de los archivos transmitidos.

En los resultados obtenidos observamos que en principio la eficiencia del protocolo mejora a medida que se incremente el tamaño de la ventana hasta alcanzar un punto de quiebre, a partir del cual la eficiencia comienza a decrecer hasta estancarse.

1. Introducción

En este trabajo práctico ejercitaremos las nociones del nivel de transporte estudiadas en la materia a través de la implementación y análisis de un protocolo sencillo desarrollado por la cátedra.

El trabajo consiste, esencialmente, en implementar el cliente de dicho protocolo de transporte simplex y luego analizar la efciencia y el desempeño del miemo en el contexto de una red local.

El protocolo desarrollado por la cátedra, PTC (Protocolo de Teoría de las Comunicaciones), puede ubicarse dentro de la capa de transporte del modelo OSI tradicional. Fue concebido como un protocolo de exclusivo uso didáctico que permita lidiar en forma directa con algunas de las problemáticas usuales de la capa de transporte: establecimiento y liberación de conexión, control de errores y control de flujo.

1.1. Características básicas del protocolo

Desde un punto de vista general, PTC presenta las siguientes características:

- Unidireccionalidad: se trata de un protocolo simplex en el que un cliente activo envía datos y un servidor pasivo los recibe, sin la posibilidad de enviar sus propios datos (a diferencia de TCP).
- Orientación a conexión: contempla procesos formales de establecimiento y liberación de conexión. Esta conexión permite que tanto el cliente como el servidor generen estado para lograr luego una comunicación efectiva.
- Confiabilidad: a través de un algoritmo de ventana deslizante, garantiza que los datos enviados por el cliente lleguen correctamente a destino.

1.2. Formato del segmento

El encabezado es de tamaño fijo: 12 bytes. Sólo exhibe campos para representar los puertos de origen y destino, para secuenciar y reconocer los paquetes y para indicar el propósito del paquete (i.e., flags). El detalle de estos campos puede verse en la figura 1.

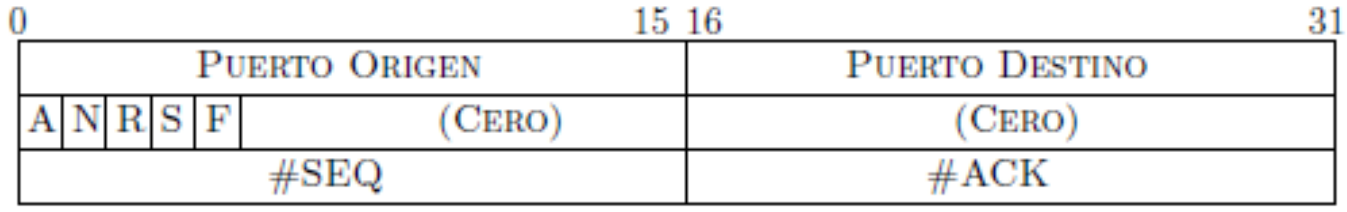

Figura 1: Formato del encabezado de PTC

Los campos A, N, R, S y F, todos de 1 bit, representan los flags:

- A es el flag de ACK; prendido sólo en los mensajes del servidor para reconocer la correcta recepción de un paquete.

- S es el flag de SYN; es enviado sólo por el cliente al requerir establecer una nueva conexión.
- F es el flag de FIN; es enviado únicamente por el cliente cuando desea finalizar una conexión existente.
- Los flags N y R actualmente no tienen uso.

Inmediatamente después de este encabezado siguen los datos, de longitud arbitraria aunque entre 1 y 1000 bytes. El número de secuencia no trabaja a nivel de byte como ocurre en TCP sino que identifica a toda la porción de los datos contenida en el paquete. Así como un segmento TCP se encapsula dentro de un datagrama IP, los paquetes de PTC también viajarán dentro de IP. Para que los hosts puedan reconocer este hecho, el campo proto del header IP debe definirise con valor 202, tradicionalmente sin uso (el protocolo TCP se identifica con el valor 6).

1.3. Establecimiento y liberación de conexión

Antes de poder enviar sus datos, el cliente debe establecer activamente una conexión con el servidor deseado. Para ello, debe enviar un segmento con el flag SYN prendido y con un valor arbitrario en el campo #SEQ que indique el número de secuencia inicial que utilizará el cliente para identificar sus paquetes. El resto de los campos pueden tener un valor arbitrario; no serán tenidos en cuenta por el servidor. Por su parte, el servidor responderá con un segmento con el flag ACK prendido y el campo #ACK reconociendo el valor de #SEQ previamente recibido. Una vez hecho esto, el servidor considerará que la conexión está establecida. Lo mismo hará el cliente al recibir este reconocimiento. En caso de no llegar, el cliente eventualmente retransmitirá el paquete inicial. Se observa que no se requiere un SYN por parte del sevidor: al no enviar datos, no es necesario que éste sincronice su número de secuencia con el cliente. Para cerrar la conexión, la secuencia a seguir es similar, aunque cambiando el flag de SYN por el flag de FIN. El segmento FIN enviado por el cliente también debe ir secuenciado como cualquier otro segmento de datos previamente enviado. En caso de no recibir el ACK respectivo, el cliente también retransmitirá el FIN.

1.4. Ventana deslizante y retransmisiones

Para garantizar la confiabilidad, PTC implementa ventana deslizante, particularmente GoBackN[1]. De esta manera, el servidor tiene una ventana de recepción de tamaño 1, mientras que el cliente tiene una ventana de emisión de tamaño que por simplicidad se fija en un valor constante arbitrario (potencialmente mayor que 1). Llamaremos `SEND_WINDOW` a este valor. El cliente, entonces, podrá enviar hasta `SEND_WINDOW` paquetes en simultáneo antes de bloquearse esperando por los sucesivos reconocimientos. Cada ACK recibido permitirá desplazar la ventana y así hacer lugar para el envío de nuevos segmentos. Además, los ACKs del servidor pueden ser acumulativos (i.e., un paquete ACK puede reconocer más de un paquete con números de secuencia contiguos enviados por el cliente).

Al enviar un segmento con datos, el cliente también encolará este segmento en la cola de retransmisión. Éste permanecerá allí hasta ser eventualmente reconocido. Por otro lado, el cliente también define un tiempo máximo de espera `RETRANSMISSION_TIMEOUT` para esperar por estos reconocimientos. De superarse este tiempo, se asumirá que el paquete se extravió en los rincones de la red y por ende será retransmitido junto con todo segmento posterior aguardando en la cola de retransmisión.

Se define también un número máximo admisible de retranmisiones, `MAX_RETRANSMISSION_ATTEMPTS`. Si algún segmento del cliente debiera ser retransmitido más veces que esta cantidad, se debe asumir que la conexión se perdió, y se pasará a cerrarla sin enviar FIN.

1.5. Estados

En lo que sigue veremos los posibles estados que pueden atrevesar tanto el cliente como el servidor:
Estados del cliente:

- CLOSED: Representa la ausencia de conexión (Estado anterior: FIN_SENT o ninguno)
- SYN_SENT: El cliente desea iniciar una conexión y ha enviado un SYN (Estado anterior: CLOSED)
- ESTABLISHED: El cliente recibió el ACK del SYN y ya puede enviar datos. (Estado anterior: SYN_SENT)
- FIN_SENT: El cliente desea terminar la conexión y ha enviado un FIN al servidor (Estado anterior: ESTABLISHED)

Estados del servidor:

- CLOSED: Representa la ausencia de conexión (Estado anterior: FIN_RECEIVED o ninguno)
- SYN_RECEIVED: El servidor recibió un SYN y debe enviar el ACK respectivo (Estado anterior: CLOSED)
- ESTABLISHED: El servidor ha enviado el ACK y está a la espera de recibir datos (Estado anterior: SYN_RECEIVED)
- FIN_RECEIVED: El servidor recibió un FIN y debe enviar el ACK respectivo (Estado anterior: ESTABLISHED)

2. Desarrollo

2.1. Implementación

Tomando como punto de partida el código suministrado por la cátedra se completó la implementación en Python del cliente del protocolo PTC. Puntualmente, se completó lo siguiente en el archivo `client.py`:

2.1.1. Método `handle_incoming` de PTCClientProtocol

Este método es invocado cada vez que llega un paquete desde el servidor. La idea de la implementación es, primero chequear si el paquete posee un `ACK`, y el mismo es válido.
Si lo es, quitamos el paquete de la cola de retransmisión, y actualizamos la ventana deslizante.
Luego eliminamos los paquetes que están en el rango del `ACK` del diccionario de intentos de retransmisión.
Finalmente, actualizamos el estado del cliente, en los casos que sea necesario: Si el estado era `SYN_SENT`, se pasa a `ESTABLISHED`; y si el estado era `FIN_SENT`, se pasa a `CLOSED`.

2.1.2. Método `handle_timeout` de PTCClientProtocol

Este método es invocado siempre que el tiempo de espera del primer paquete encolado en la cola de retransmisión se agota.

Si hay paquetes en la cola de retransmisión, primero chequeamos que no se haya excedido la cantidad máxima de reenvíos. Si eso ocurre, cerramos la conexión y dejamos el mensaje de error correspondiente. Caso contrario, hacemos una copia de la cola de retransmisión, borramos la cola original, y a cada paquete en la copia lo reenviamos y encolamos en la cola original, actualizando el número de retransmisión.

2.1.3. Lógica de la clase ClientControlBlock

Esta clase maneja las variables de la ventana deslizante del protocolo. Se crearon los métodos necesarios para poder verificar si es posible enviar (método `send_allowed`), para determinar si un ACK es aceptado (método `valid_ack`) y para reajustar la ventana dado un ACK aceptado (método `update_window`).

2.2. Experimentación

Una vez implementado el protocolo procedimos a probarlo enviando archivos, variando el tamaño de la ventana de emisión y el tamaño del archivo.

Las pruebas fueron realizadas en una red local con equipos conectados por wifi, realizando multiples experimentaciones para el mismo SEND_WINDOW y tamaño de archivo.

2.2.1. Ventana de emisión

La ventana de emisión es de suma importancia en el testeo y análisis del protocolo debido a la siguiente limitación: $throughput \leq SEND\_WINDOW/RTT$.

Por este motivo medimos el desempeño variando el tamaño de la ventana para observar como evoluciona la eficiencia a medida que crece, si bien la cota para el throughput va a crecer junto al SEND_WINDOW (suponiendo que el RTT se mantiene fijo en nuestra LAN), el throughput no necesariamente debe hacerlo.

Esperamos que una vez alcanzado cierto throughput, el aumento del SEND_WINDOW no este ligado a un incremento aún mayor del throughput debido a que la red en si misma cuenta con limitaciones en cuanto al ancho de banda, velocidad, buffers, etc.

2.2.2. Tamaño del archivo

Para probar la efectividad del protocolo variamos el tamaño del archivo de forma de mantener SEND_WINDOW paquetes en vuelo y ver la efectividad del protocolo cuando es utilizado en toda su capacidad por períodos largos.

Variando el tamaño del archivo esperamos, de estabilizarse o presentarse algún patrón en la taza de reenvios, connection timeouts o throughput, poder detectarlo.

2.2.3. Retransmiciones y connection timeouts

Un factor determinante en el desempeño del protocolo son los timeouts y la cantidad de retransmiciones realizadas debido a el impacto que tiene sobre el throughput y a que de alcanzarse los MAX RETRANSMISSION ATTEMPTS para algún paquete, la conexión es cerrada.

Por este motivo se registro la cantidad de retransmiciones realizada en cada experimento y la cantidad de veces que se alcanzaron los MAX RETRANSMISSION ATTEMPTS.

2.2.4. Throughput

Con los valores obtenidos de los experimentos realizados medimos el throughput percibido y observamos la relación entre este y las retransmiciones con el fin de determinar de que forman afectan las variables testeadas en el resultado del protocolo.

3. Resultados

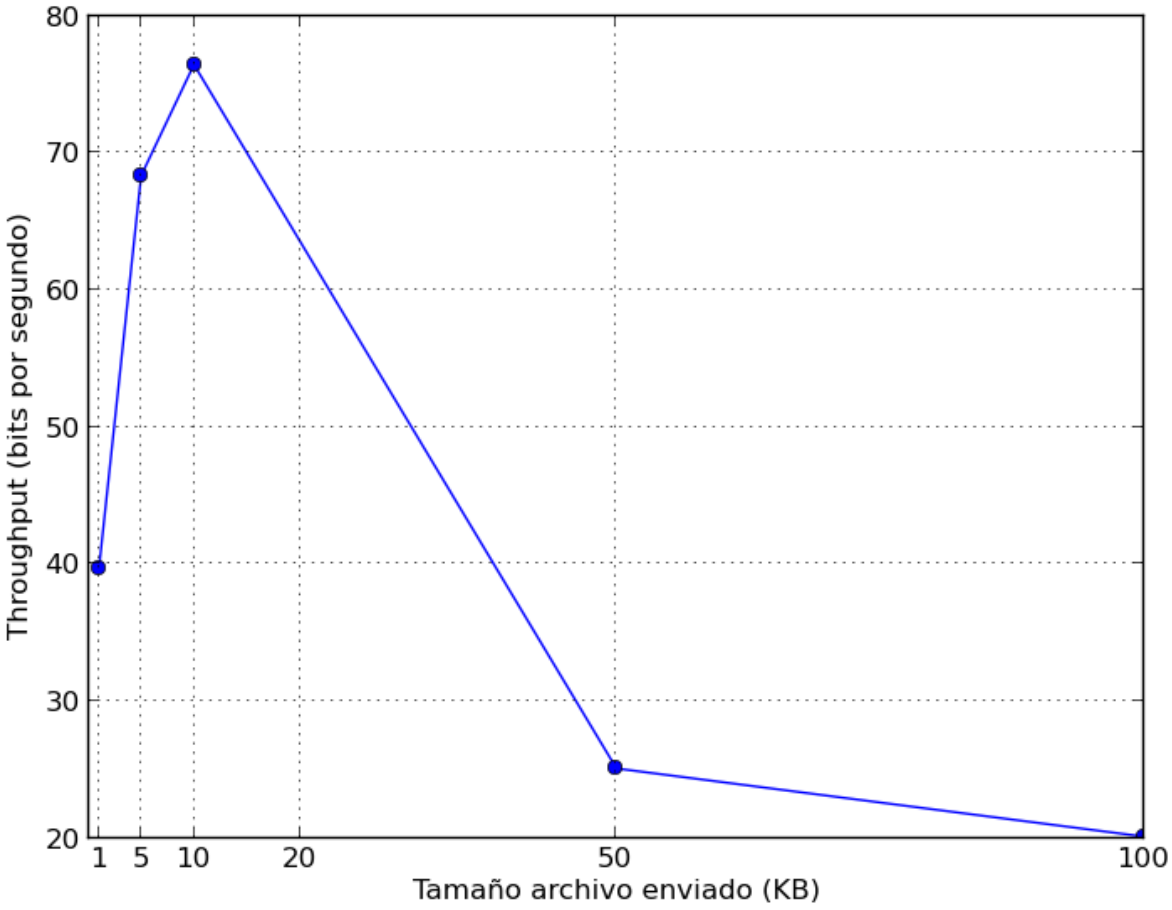

Figura 2: Throughput percibido (ventana de emisión fija en 10)

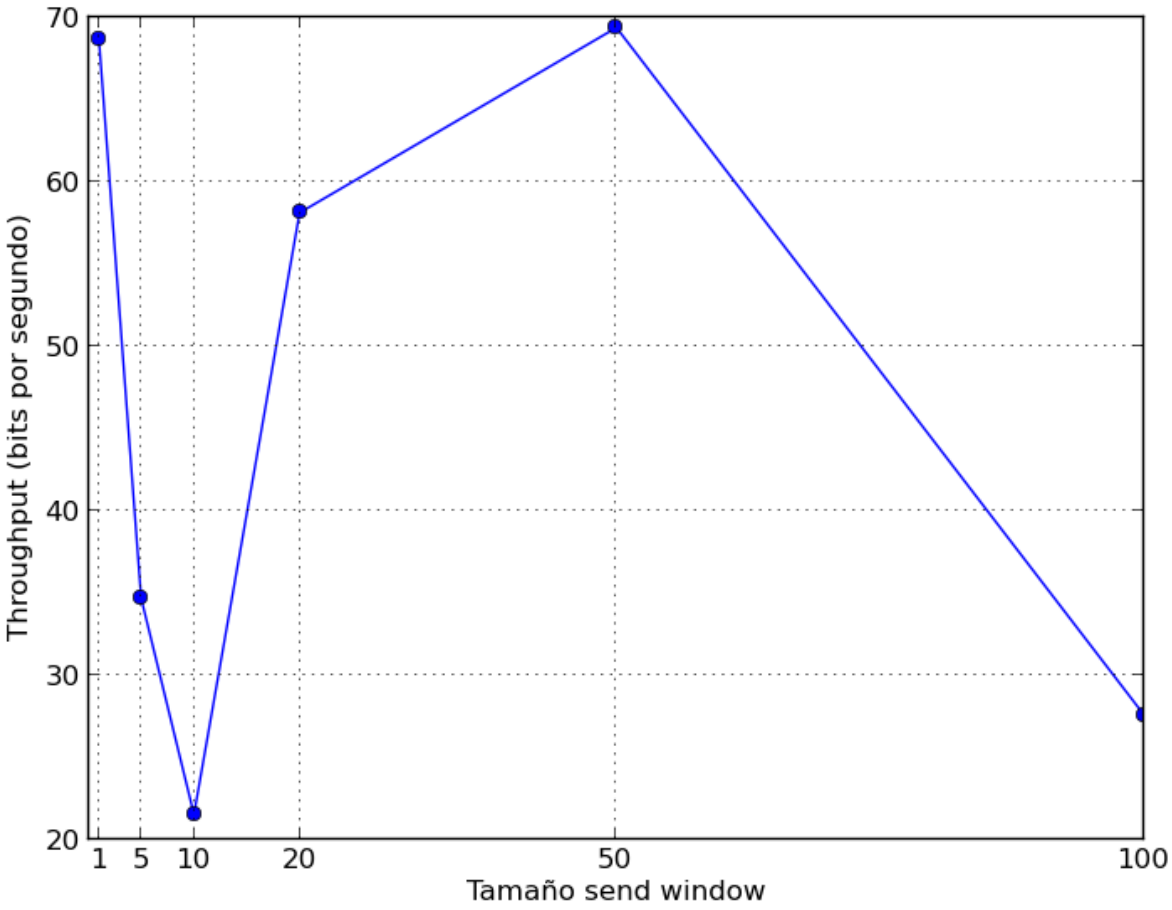

Figura 3: Throughput percibido (tamaño de archivo fijo de 50KB)

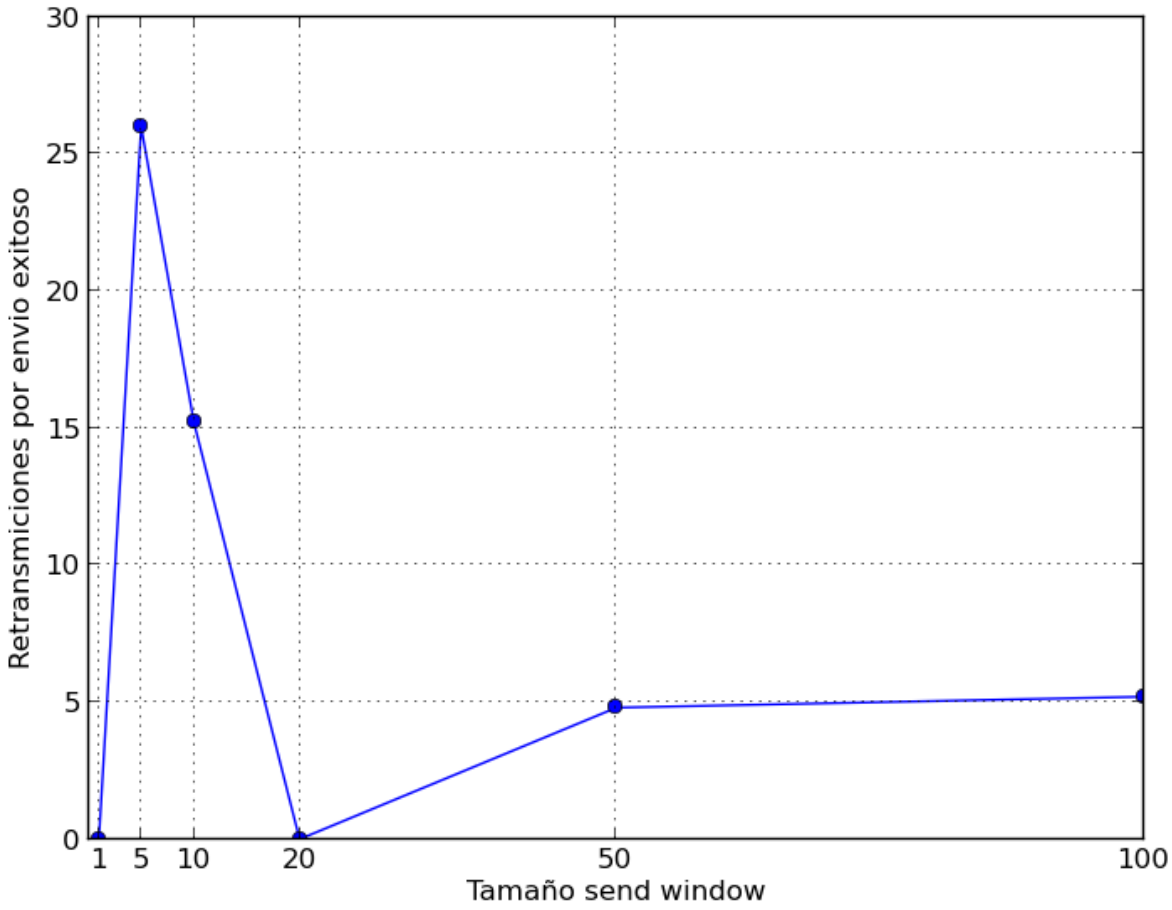

Figura 4: Cantidad de retransmiciones (tamaño de archivo fijo de 50KB)

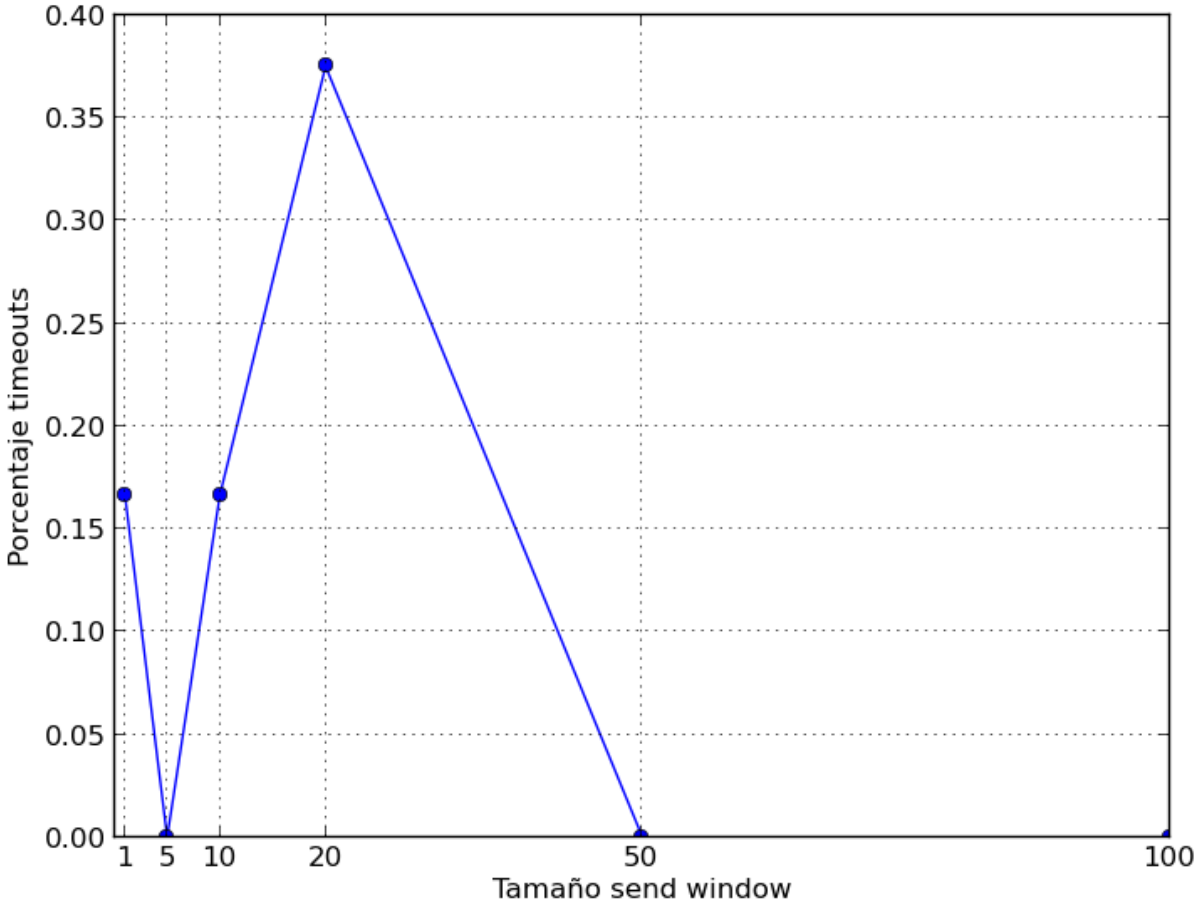

Figura 5: Porcentaje de timeouts por muchas retransmiciones (tamaño de archivo fijo de 50KB)

4. Análisis y conclusiones

Cuando medimos el throughput para una ventana fija en 10, variando el tamaño del archivo, observamos que el máximo está en un tamaño de 10kb.

En el siguiente experimento, manteniendo fijo el tamaño del archivo en 50 kb, vemos que el mayor throughput se observa con tamaño de ventana de 50 y 1, mientras que para un tamaño de 10 se obtiene el menor. Para dicha configuración, analizando la cantidad de retransmisiones vemos que el pico está en un tamaño de ventana de 5, aunque 10 también posee un valor considerable. Esto parecería tener correlación con el bajo throughput percibido para estos tamaños de ventana en el gráfico anterior.

En cuanto a la ventana de emisión, si bien una mayor ventana nos puede permitir llegar a utilizar la capacidad completa del canal y aumentar el throughput, si es demasiado grande, esto puede conllevar desventajas:

- Al vernos obligados a mantener los segmentos no reconocidos, en caso de ser necesarios para una posible retransmición, necesitamos un RETRANSMISSION_BUFFER más grande.
- El receptor se puede ver obligado a descartar mensajes si su buffer se encuentra lleno y el emisor sigue enviando mensajes.
- La red o buffers intermedios pueden ser congestionados de forma que se pierdan paquetes.

Esta relación entre una mayor ventana de emisión y un decremento en el throughput se puede observar una vez superado el punto de quiebre de la SEND_WINDOW = 50, obteniendo un throughput de aproximadamente un 40 % del máximo, duplicando la SEND_WINDOW a 100.

En el último gráfico se observa que el porcentaje de timeouts para un tamaño de archivo fijo de 50kb, se da con un tamaño de ventana de 20. Para los valores de 1 y 10 se obtiene un porcentaje igual, mientras que para el resto de los tamaños de ventana el porcentaje es 0. No logramos encontrar una explicación para este comportamiento aunque sospechamos que se debe a la red en la que se realizaron las pruebas.