Teoría de las Comunicaciones

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

Trabajo Práctico 2

Integrante	LU	Correo electrónico
Antonio, Pablo	290/08	pabloa@gmail.com
Ferrari, Gastón	775/07	gastonferrari5@hotmail.com

Índice

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 llamado PTC. Para hacerlo, implementaremos un cliente para interactuar con el servidor provisto por la catedra.

1.1. PTC

El protocolo que estudiaremos, 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.

1.2. Capa de transporte

El nivel de transporte o capa de transporte es el cuarto nivel del modelo OSI encargado de la transferencia libre de errores de los datos entre el emisor y el receptor, aunque no estén directamente conectados, así como de mantener el flujo de la red. Es la base de toda la jerarquía de protocolo. La tarea de esta capa es proporcionar un transporte de datos confiable y económico de la máquina de origen a la máquina destino, independientemente de la red de redes física en uno. Sin la capa transporte, el concepto total de los protocolos en capas tendría poco sentido.

2. Desarrollo

Para este TP, a diferencia de los anteriores, tuvimos que completar la implementación dada del protocolo PTC. Las tareas consistieron en:

- Completar el método *handle_incoming* de la clase *PTCCLientProtocol*, que maneja los paquetes recibidos.
- Completar el método *handle_timeout* de la clase *PTCCLientProtocol*, que se llama al agotarse el tiempo de espera del primer paquete en la cola de retransmisión.
- Completar la clase *ClientControlBlock*, que maneja la ventana deslizante del protocolo.

2.1. *PTCClientProtocol* :: *handle_incoming*

Este método es prácticamente una traducción de la especificación del protocolo a python. Lo que hace es:

- Verificar que el flag de ACK este seteado.
- Ver si el nro de ACK es aceptado (ver *ClientControlBlock* en 4.3).
- Llamar al método de la *retransmission_queue* que saca los paquetes reconocidos por el ACK.
- Ajustar la ventana deslizante (ver *ClientControlBlock* en 4.3).
- En el caso que se esté en los estados SYN_SENT o FIN_SENT setear los nuevos estados como corresponda y establecer o cerrar la conexión según el caso.

2.2. *PTCClientProtocol* :: *handle_timeout*

Este método se encarga de retransmitir los paquetes que están en la *retransmission_queue*. Se itera la cola verificando que no se haya excedido la cantidad máxima de intentos de retransmisión, si esto pasa se cierra la conexión, sino se retransmite el paquete y se actualiza la cantidad de retransmisiones. Los paquetes se vuelven a encolar para poder ser retransmitidos nuevamente.

2.3. *ClientControlBlock*

En esta clase se completó el método *send_allowed* que checkea que la ventana de emisión no esté saturada, comparando el número de secuencia del mensaje a enviar con la variable *window_hi* de la clase que mantiene el máximo número de secuencia que se puede enviar.

A su vez se agregaron métodos para incrementar el número de secuencia para el próximo paquete, otro para verificar que el ACK que llega es válido y por último uno para ajustar la ventana de emisión que setea los valores correspondientes en las variables *window_lo* y *window_hi*.

2.4. Otros

Tuvimos que comentar la linea 81 de client.py ya que sino sólo se incrementaba el número de secuencia si el payload del mensaje no era vacío, esto hacía que no se incremente el número de secuencia después del primer mensaje (SYN) y quedaba desfasada la ventana de emisión.
Además, para testear el protocolo con archivos, implementamos una función *sendFile* la cual recibe un archivo y lo manda

3. Resultados

Para probar el protocolo, armamos archivos de 1kb, 5kb, 10kb, 50kb, 100kb, 200kb y 500kb. El servidor y el cliente se encuentran en distintos host en la misma LAN sobre Wi-Fi la cual presenta un RTT de aproximadamente 2ms entre ambos hosts. Para tomar los tiempos y comprobar el correcto funcionamiento del protocolo, usamos el wireshark.

Para tener mediciones mas aproximadas a la realidad, tomamos el promedio de 10 mediciones.

3.1. Tiempo de transmisión

En este primer gráfico presentamos el tiempo de transmisión de los archivos, usando SEND_WINDOWs de 1, 10, 15 y 20.

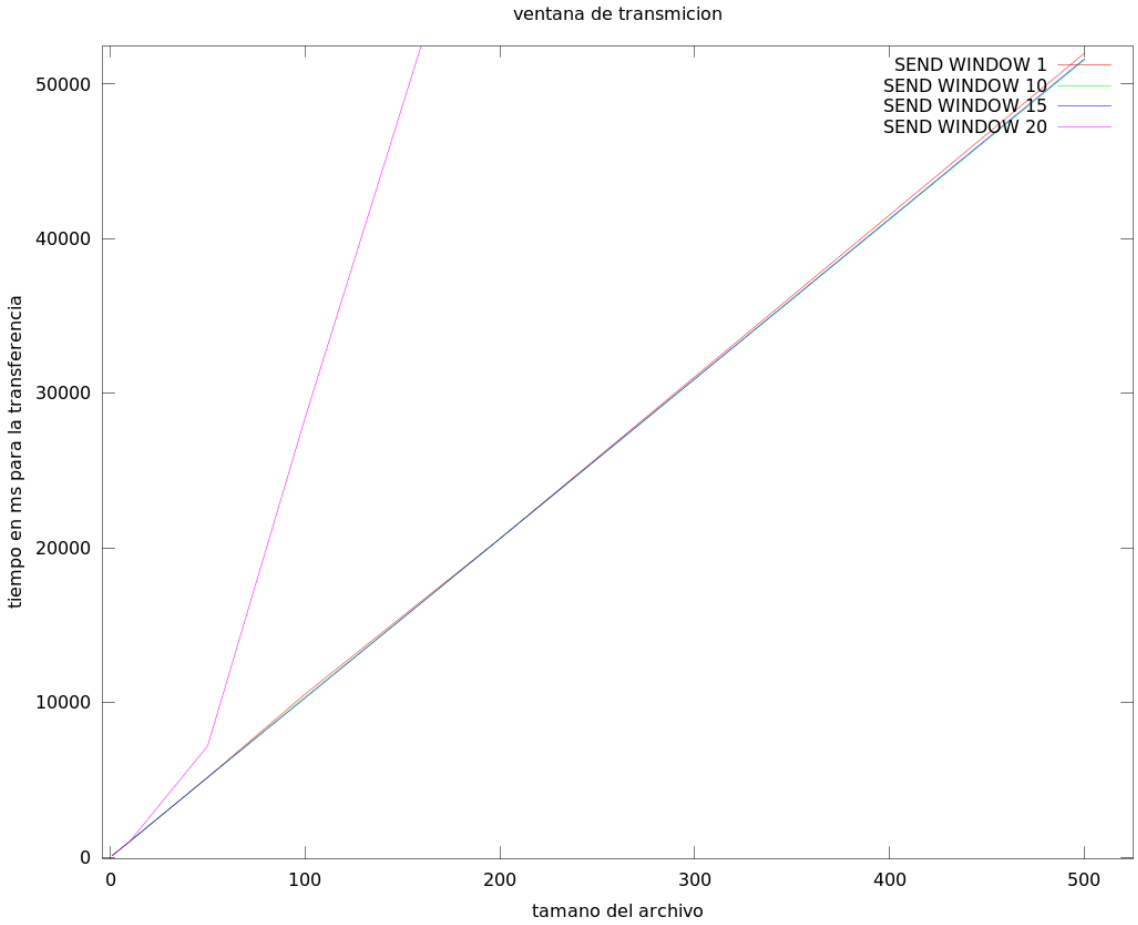

Figura 1: Tiempo de transmisión

En el gráfico podemos observar como el tiempo de transmisión usando ventanas de 1, 10 y 15, es muy similar. En cambio, cuando usamos una ventana de 20, es notable la diferencia en el tiempo que tarda en enviarse el archivo. Esto se debe a la cantidad de timeouts que ocurren al utilizar este tamaño de ventana.

3.2. Velocidad de Transferencia

En este segundo gráfico presentamos la velocidad de transmisión (en kbs) de la transmisión de los archivos, usando SEND_WINDOWs de 1, 10, 15 y 20.

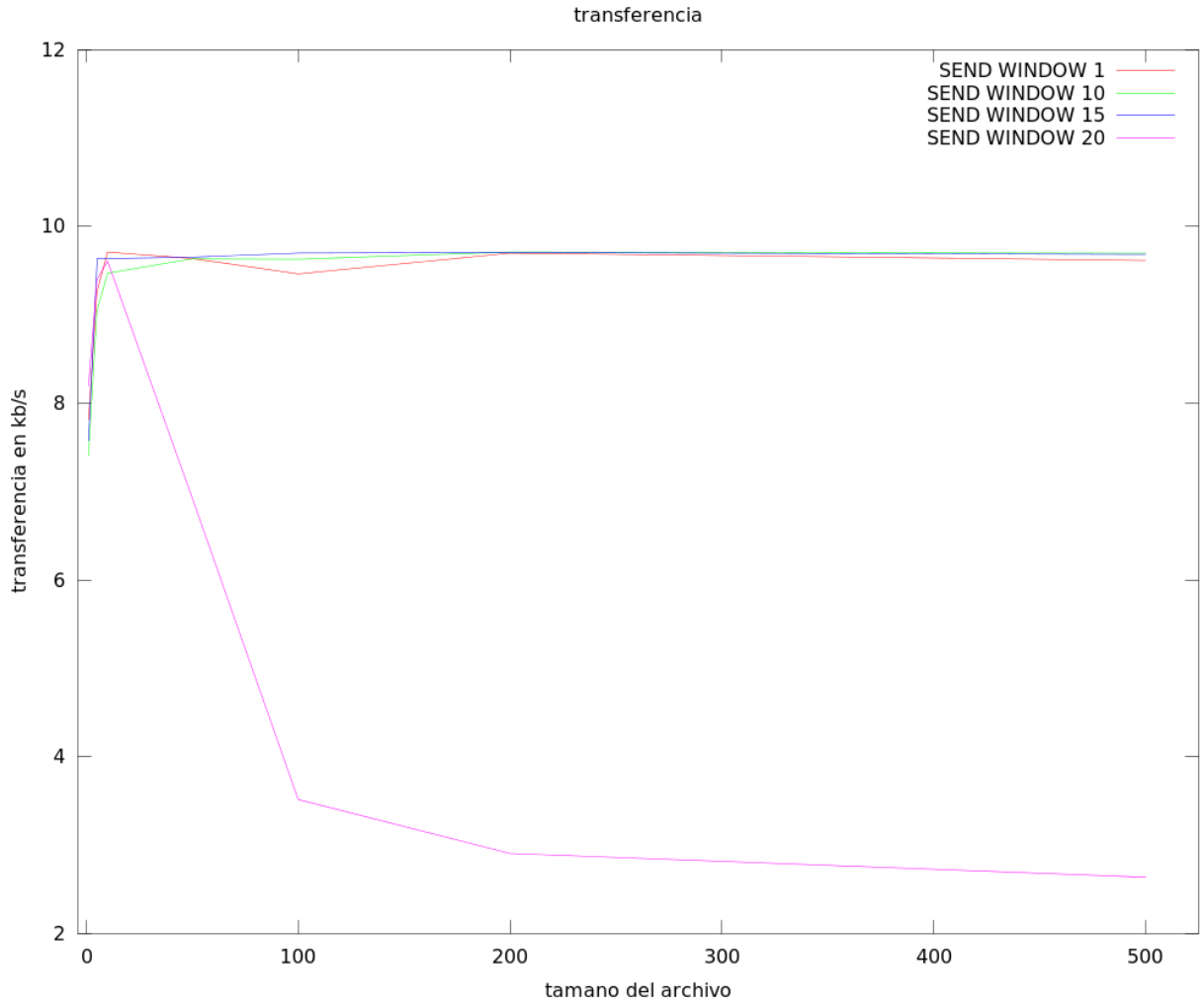

Figura 2: Tiempo de transmisión

Al igual que en el gráfico anterior, podemos observar la gran diferencia que existe en la velocidad de transmisión cuando la ventana de emisión crece.

La velocidad se mantiene estable con ventanas de 1, 10 y 15, pero cuando usamos una ventana de 20, la velocidad tiende a caer mucho a medida de que aumenta el tamaño del archivo.

Si comparamos este gráfico con el anterior, se ve claramente como al ser menor el throughput, el tiempo de transmisión es mucho mayor.

3.3. Cantidad de Timeouts

En esta sección presentamos la cantidad de timeouts que tuvieron en promedio el envío de los archivos.

Tamaño del archivo/SEND_WINDOW	1	10	15	20
1kb	0	0	0	0
5kb	0	0	0	0
10kb	0	0	0	0
50kb	0	0	0	5
100kb	0	0	0	14
200kb	0	0	0	34
500kb	0	0	0	94

De igual manera que en las 2 secciones anteriores, podemos observar la gran diferencia que hay entre las ventanas, en este caso, con respecto a los timeout.

Vemos que con ventanas de 1, 10 y 15, no tuvimos timeout en ninguno de los archivos que mandamos. En cambio, vemos que con la ventana en 20, los archivos mas chicos tampoco tuvieron timeout, cuando los archivos mas grandes empiezan a presentar cada vez mas timeouts. Los timeouts se dan ya que los paquetes encolados en el buffer del emisor se les acaba el ttl dado que los acks que se reciben del receptor no llegan lo suficientemente rápido para evitar que se acabe dicho tiempo para el paquete. Si seguimos aumentando el send window a tamaños mayores a 20 solo empeoraríamos la perfomance ya que al producirse un timeout se aplica una política de retransmisión de GoBackN lo que implica que una mayor cantidad de paquetes se deben retransmitir.

4. Conclusiones

Concluimos que, al tener una ventana de recepción de 1, el emisor esta muy restringido en cuanto al valor de su ventana de emisión. Esto genera que para valores de la ventana de emisión que superan 20 el protocolo deje de funcionar de manera eficiente y aceptable ya que se genera un cuello de botella en el receptor. También provoca que los tamaños de ventana de emisión con valores menores a 20 se comporten y presenten una performance casi idéntica ya que la ventana de recepción de tamaño 1 no permite aprovechar este aumento.

5. Bibliografia

- `http://es.wikipedia.org/wiki/Capa_de_transporte`
- Tanenbaum, A. Computer Networks, 3ra Ed. Capítulo 3: páginas 207-213.