DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP3: Capa de Transporte

Teoría de las Comunicaciones

Integrante	LU	Correo electrónico
Furman, Damián	936/11	damian.a.furman@gmail.com
Lambrisca, Santiago	274/10	santiagolambrisca@hotmail.com
Marottoli, Daniela	42/10	dani.marottoli@gmail.com
Vanecek, Juan	169-10	juann.vanecek@hotmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Contents

1 Introducción

La capa de transporte, es la capa de nivel 4 en el modelo OSI, es la capa encargada del transporte de datos de una maquina a otra. Para permitir este intercambio de información y tener la certeza de que la información a sido bien entregada, manteniendo su integridad, existen distintos protocolos. Entre ellos, el mas utilizado es TCP. TCP, nos asegura el envío sin perdida de información, el orden de los datos y la confiabilidad. Para poder cumplir con su objetivo el protocolo debe resolver ciertos problemas que se le presentan en la red, entre ellos nos encontramos con la demora o *delay* y la *perdidadepaquetes*. Estos pueden producirse por diversas causas en la red, y son disparadores de acciones en nuestro protocolo.

El objetivo del trabajo realizado, es poder comprender el comportamiento de la capa de transporte. Para ello, utilizaremos un protocolo de transporte implementado por la cátedra de la materia, sobre el cual efectuaremos ciertos cambios que nos permitan simular los problemas presentes en una red. Pudiendo asi, controlar estas variables, y sacar conclusiones sobre los efectos que estas tienen sobre el comportamiento del protocolo.

2 Desarrollo

Tomando como punto de partida el código suministrado por la cátedra, introdujimos las siguientes modificaciones para agregar dos funcionalidades: por un lado, la introducción de Delay y por el otro, la posibilidad de definir una probabilidad p de pérdida de paquetes con valores que se ubiquen entre el 0 y el 1 (donde 0 no pierde ningún paquete y 1 lo pierde todos). Estos mismos seran efectuados sobre el envio de paquetes de acknowledgment ACK. Provocando que quien envia el paquete, crea que este no fue entregado correctamente.

La implementación se hizo de tal manera que tanto el valor del tiempo de delay como la probabilidad de la pérdida de paquetes se setean como parámetros al ejecutar la función.

Con el fin de agregar estas funcionalidades, se realizaron los siguientes cambios al código original:

- Dentro del archivo **handler.py**:
  - Agregamos la función *se_perdio_paquete* que toma la probabilidad pasada por parámetro y decide sobre la base de esa probabilidad si el paquete que va a enviar efectivamente se va a perder o no.
  - Dentro de la función *send_ack* agregamos un condicional que llama a la función "se_perdio_paquete". Si esta decide que el paquete se perdió, el condicional nos lleva a una sentencia de return y el paquete no se envía. Luego, agregamos también dentro de esta función un Delay simulado mediante la instrucción *time.sleep*() y cuyo valor es un porcentaje, pasado por parámetro, del delay máximo permitido por el protocolo (1 segundo). Para enviar un paquete, la función espera la cantidad de tiempo indicada antes de ejecutar el *build_packet* y el *send*.

Luego de algunas pruebas con estas implementaciones, pudimos observar que aun cuando estabamos perdiendo todos los $ACK's$, el protocolo funcionaba correctamente, asi decidimos que era una buena idea agregar tambien, la pérdida a la actualización de la ventana para realizar nuevas pruebas.

- Dentro del archivo **protocol.py**:
  - Agregamos la función *se_perdio_paquete* que se comporta de manera similar a la mencionada en el item anterior dentro del archivo handler.py
  - Dentro de la función receive se agrega un condicional que llama a la función *se_perdio_paquete* mencionada en el item anterior. El condicional enviará la actualización de ventana unicamente si esta función decide que el paquete a enviar no se va a perder. Si la función retorna que el paquete debe perderse, no se envía la actualización de ventana y se imprime por consola que el paquete se perdió.
  - Además, en la función $retransmit_packets_if_needed()$ se agrego un contador para las retransmiciones, para poder evaluar la cantidad de retransmiciones en función del delay.

3 Resultados

A medida que fuimos desarrollando nuestros experimentos, y modificando nuestro código, nos hemos encontrado con distintas caracteristicas, algunas de las cuales queremos mostrar a traves de los siguientes gráficos.

A continuación presentamos un gráfico del Throughput en función del tamaño del mensaje, para distintos delays.

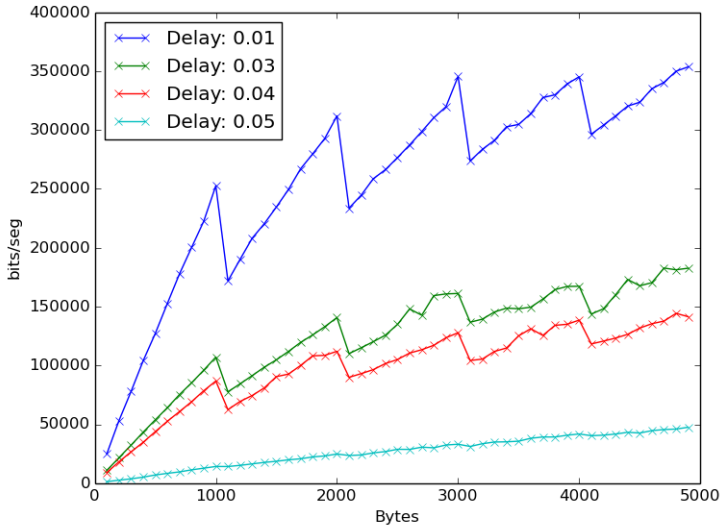

Figure 1: caption

Buffer 1024 bytes.

Aqui podemos observar, como disminuye el throughput a medida que aumenta el delay. Tambien podemos observar que el throughput aumenta a medida que aumenta el tamaño del mensaje. Esto se debe, a el aprovechamiento del buffer y queda evidenciado en los saltos que se hacen presentes cada 1024 Bytes, donde delimita el uso completo del buffer. Por ende, vemos como aumenta la eficiencia cuando los mensajes aprovechan el tamaño total de la ventana.

Ahora, presentaremos 4 gráficos correspondientes al tamaño del mensaje en función del tiempo, para distintos tamaños de buffer, comparando los resultados para distintos delays y porcentajes de perdida de paquetes.

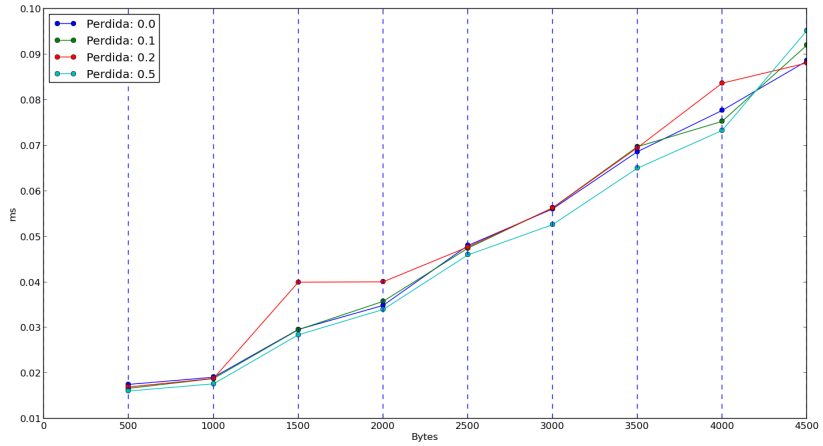

Figure 2: caption

Tiene el max buffer seteado en 500 Bytes. Se evaluó el tiempo que tarda el protocolo en función del tamaños, y distintos porcentajes de perdidas. El delay esta seteado en 0.

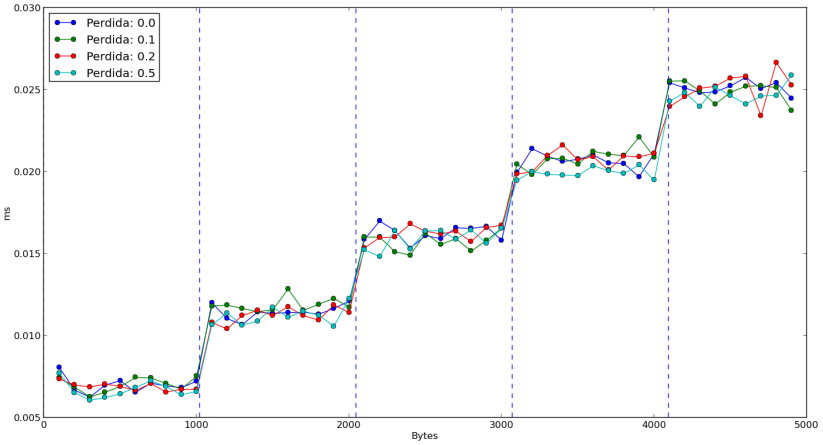

Figure 3: caption

El mismo que el anterior pero con un buffer de 1024, y una mayor granularidad en los tamaños.

Viendo los resultados obtenidos, para distintos porcentajes de perdida en dos buffers distintos. Vemos que no difieren demasiado los resultados obtenidos. Debido a la aleatoreidad de la perdida de paquetes nos encontramos con algunos picos que parecen indicarnos que se han producido una cantidad de perdidas considerables en algunas pruebas.

Sin embargo, sabemos que el echo de la actualización de ventana, seguramente este afectando nuestro resultados. Para evitar estos, nos dispusimos a agregar perdida de paquetes no solo a los $ACK's$, sino, también las actualizaciones de ventana. Era nuestra intención presentar los gráficos correspondientes a los mismos experimentos con dichos cambios al código, pero nos encontramos con problemas a la hora de ejecutarlo.

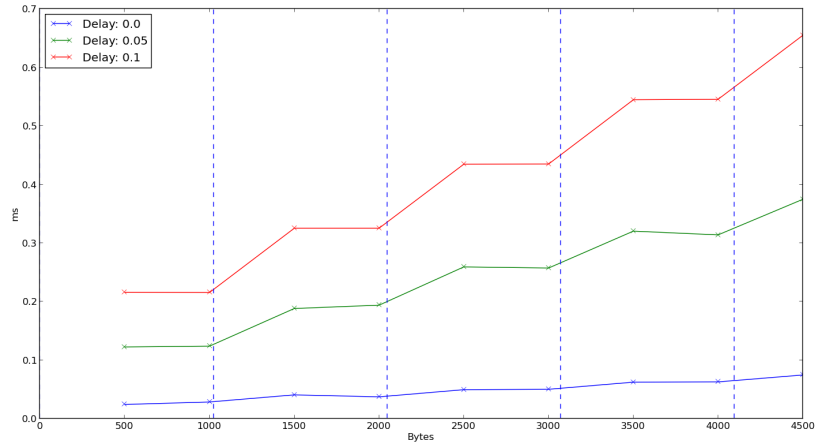

Figure 4: caption

Tiene el max buffer seteado en 1024 Bytes. Se evaluaó el tiempo que tarda el protocolo en función del tamaños, y distintos porcentajes de delay. La perdida esta fijada en 0.

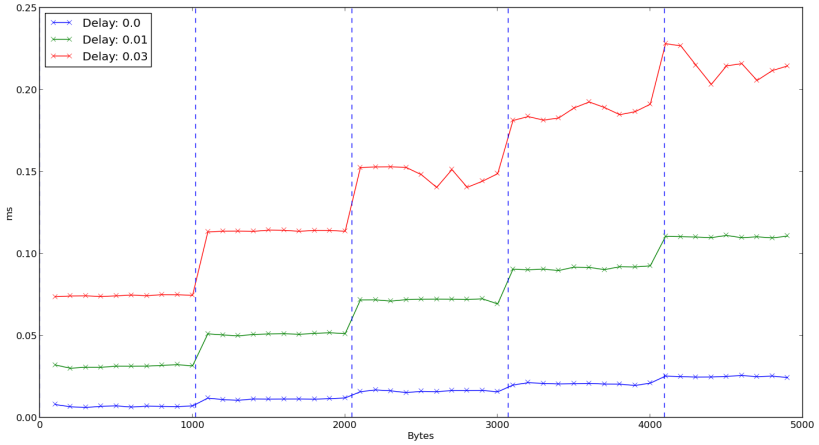

Figure 5: caption

El mismo que el anterior pero más granularidad en los tamaños evaluados.

En este caso, podemos ver como aumenta el tiempo de transmision a medida que aumenta el delay, también se pueden observar esos saltos correspondientes al tamaño del buffer. Los gráficos parecen bastante explicitos, y no dejan mucho para decir. A medida que aumenta el tamaño del mensaje aumenta el tiempo de transmisión, este tiempo presenta un aumento marcado cuando se llena el buffer y debe utilizarse la primitiva de recepcion de datos mas de una vez.

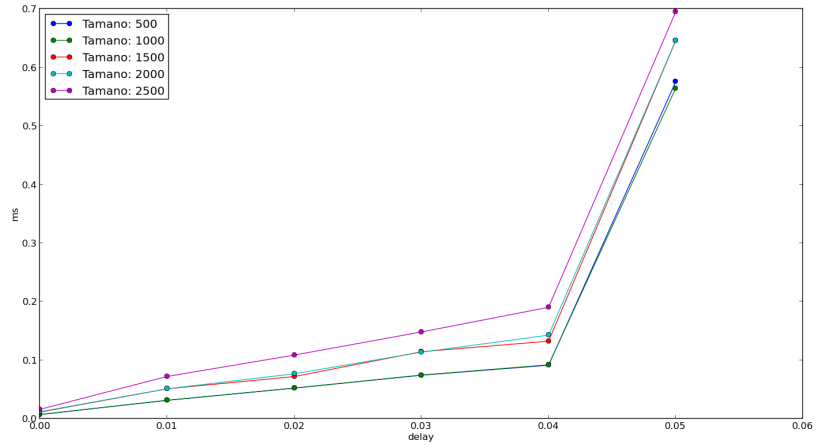

Figure 6: caption

Buffer 1024 bytes.

Como era de esperar, mientras mas delay el tiempo aumenta, en este gráfico vemos como para los mismos tamaños de mensaje el tiempo aumenta con el delay.

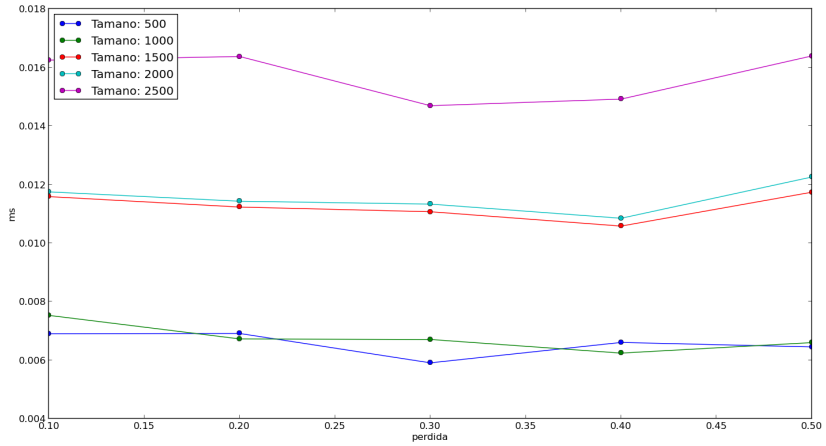

Figure 7: caption

Buffer 1024 bytes.

En cierto punto, estos resultados son los esperados. Los mensajes de mayor tamaño son los que mas tiempo tardan. Sin embargo, esperariamos que a medida que aumenta el porcentaje de perdida también aumente el tiempo. Como mencionamos anteriormente, esto seguramente se deba a la actualización de ventana, la cual permite al protocolo continuar transmitiendo los siguientes paquetes, aun sin haber recibido los $ACK's$ correspondientes.

4 Conclusiones

Luego de las pruebas realizadas, de los datos obtenidos y de los resultados analizados. Además de los problemas enfrentados durante las horas transcurridas durante el desarrollo. Podemos notar la complejidad que requiere tener un protoclo de transporte robusto, que nos permita mantener la integridad de los datos y la confiabilidad en el envío. Pudimos ver la cantidad de problemas que debemos enfrentarnos si pretendemos cumplir con estas caracteristicas a la hora de implementar un protocolo. También vimos el efecto que pueden tener los fenomenos que pueden presentarse en la red. También nos encontramos con que todos los mecanismos que nos llevan a asegurar la entrega correcta del mensaje, podrian insistir eternamente ante ciertas cirscunstancias, es por eso que el protocolo debe tener ciertos limites, y es una cuestión de decisión cuanta tolerancia a errores va a tener el protocolo. Asi, dependiendo de las decisiones que tomemos, se vera afectado, no solo el funcionamiento, sino, también el rendimiento del protocolo en cuestión.

Para finalizar, la transferencia de mensajes muchas veces puede darse en una red sobre la cual tenemos control, pero lo mas interesante lo vamos a encontrar cuando nos conectamos a redes ajenas, sobre las cuales sabemos poco y debemos abstraernos de ella. Es por eso que un protocolo de transporte que pretenda ser confiable, debe lidiar con muchisimos problemas ajenos para cumplir con su objetivo, y debe estar preparado para poder sobreponerse a ellos.