Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación
CC4303-1 Redes

Tarea 2

"Un protocolo de transporte apurete"

Integrantes: Sebastián Arriola
Carolina Contreras
Profesora: Catalina Alvarez
Auxiliar: Ivana Bachmann
Ayudantes: Rodrigo Delgado
Daniela Ruz
Fecha : 27 de Noviembre de 2018
Santiago, Chile

1. Introducción

El presente informe corresponde a la presentación de los resultados obtenidos al ejecutar diversos casos de prueba para la tarea 2 del curso CC4303-1 Redes, semestre primavera 2018. En dicha tarea se implementan un cliente y un servidor que envían y reciben un archivo respectivamente, usando *go-back-N*. Antes de enviar el archivo, ambos deben establecer una conexión, la cual se valida utilizando los mecanismos *two/three way handshake*, ambos por separado.

Para cada uno de ellos, se simulan con el comando *netem* porcentajes de pérdida y *delay* en la transmisión, con tal de estudiar como varía el tiempo promedio para enviar el archivo, y si ocurren o no fallos en la conexión o exceso de re-transmisiones.

Para esta implementación en particular, se usó un valor máximo aceptado de 10 re-transmisiones, tanto a la hora de establecer la conexión como de enviar una misma ventana de datos por parte del enviador, o de esperar recibir un mismo *ack* por parte del receptor. El tamaño de ventana utilizada también fue 10, y se utilizan números de secuencias tal que la cantidad es igual a dos veces el tamaño de la ventana,partiendo desde el 0.

Finalmente, para terminar la conexión, tanto cliente como servidor deben comunicar deben enviar un mensaje de fin y recibir su respectivo *ack* de la contraparte. Este mecanismo es siempre el mismo, tanto para *two* como para *three way handshake*.

2. Resultados

El archivo enviado en cada una de las pruebas corresponde al provisto por la profesora auxiliar: *"The Project Gutenberg EBook of The Adventures of Sherlock Holmes, by Sir Arthur Conan Doyle"*, cuyo tamaño es de 122.539 bytes.

Para cada una de las formas de establecer la conexión (*two/three way handshake*) se realizan pruebas usando pérdidas de 0 %, 10 % y 30 %, y *delays* de 0ms, 100ms y 500ms, todas las combinaciones posibles, y para cada combinación se realiza el envío 10 veces, con tal de poder estudiar el tiempo promedio, la desviación estándar, y en cuántos casos ocurren errores, tales como una conexión fallida (no se logra establecer la conexión) o exceso de re-intentos para enviar una misma ventana o recibir los datos correctamente por parte del receptor.

Dado que la implementación tiene un contador para el número de re-intentos tanto en caso del enviador como del receptor, cuando uno de los dos sufre una caída inesperada o se pierde la conexión, el otro termina también la conexión por exceso de re-intentos.

Cabe destacar también que en cada ensayo, se arranca primero el enviador y luego el receptor: en caso contrario, es decir cuando el receptor está previamente listo para recibir la información, en proceso se envío tarde del orden de 2 segundos menos.

Los resultados obtenidos se presentan en las tablas a continuación:

2.1. *two way handshake*

% perdida	delay [ms]	con. fallidas	exceso re-intentos	promedio [s]	dev. std. [s]
0	0	0	0	2.0128	0.0033
0	100	0	0	7.4197	0.0035
0	500	0	0	27.0189	0.0026
10	0	0	2	61.3525	10.5681
10	100	1	1	66.6328	9.5667
10	500	2	2	88.9156	8.4396
30	0	4	2	237.7824	29.0780
30	100	4	2	241.7836	28.4556
30	500	3	3	265.5913	29.0424

2.2. *three way handshake*

% perdida	delay [ms]	con. fallida	exceso re-intentos	promedio [s]	dev. std. [s]
0	0	0	0	2.0125	0.0024
0	100	0	0	7.4194	0.0018
0	500	0	0	27.0539	0.0385
10	0	1	2	56.9378	12.0710
10	100	3	1	66.2213	8.6069
10	500	3	0	91.3669	10.8517
30	0	5	1	206.2861	33.9374
30	100	7	0	251.5590	29.6479
30	500	5	2	311.9642	10.7021

3. Discusión y Conclusiones

Se puede ver que cuando no existe porcentaje de pérdidas, los archivos se envían sin problemas y el tiempo total del proceso aumenta a medida que aumenta el *delay*. Además, las desviaciones estándar son extremadamente bajas, mucho menores a 1 segundo.

Cuando existe pérdida, no solo aumentan considerablemente los tiempos promedio de envío, sino que la desviación estándar también, evidenciando que el tiempo de envío puede variar mucho de un caso a otro. Además, existen situaciones en las que no es posible establecer la conexión o el proceso termina por exceder el número máximo de re-intentos. Esto ocurre en menor medida cuando la pérdida es del 10 %, y se vuelve mucho más considerable cuando la pérdida asciende al 30 %.

Con respecto a los métodos de establecer la conexión, los tiempos y casos totales con fallos son relativamente similares para ambos. Esto se atribuye a que para la implementación de esta tarea, cuando se quiere establecer la conexión, si falla algún paso intermedio en los mecanismos de *two/three way handshake*, el proceso se repite mediante un ciclo *while*, y es solamente el *ack* final de cada uno el que se envía una sola vez. Si los porcentajes de pérdida derivan en que este *ack* se pierde, entonces la conexión no se logra establecer. Deibdo a esta situación es que finalmente se obtienen resultados similares

para ambos mecanismos.

Si no fuese así, y cada uno de los *syn* y *ack* se enviara una sola vez, fallaría mucho más el *two/three way handshake* cuando los porcentajes de pérdidas son altos, pero a su vez en caso de lograrse la conexión, de cierta forma da una mayor garantía con respecto a que debiese existir menor probabilidad de que el envío falle antes de completarse. Cuando no hay pérdidas, cualquiera de los dos que se use no es relevante, pues ni siquiera son necesarios.

Una forma de mejorar los resultados obtenidos es aumentar el número permitido de re-transmisiones, pero esto en un caso real se traduce en que existe mayor tráfico de datos en la red. Cambiar el protocolo a *selective repeat* también podría ser una opción, pues así los *ack* fuera de orden no necesitan ser re-transmitidos, y esto solo se haría para aquellos que se pierden. Empero, se debe tener en cuenta que con *selective repeat* es necesario que el receptor también tenga un *buffer* de datos y además el enviador debe manejar *timeouts* para cada uno de los elementos de la ventana.