UNIVERSIDAD DE COSTA RICA

ESCUELA DE CIENCIAS DE LA COMPUTACIÓN E INFORMÁTICA
FACULTAD DE INGENIERÍA

CI-1320 Redes de computadoras

Tarea Programada I
Simulador de protocolo Selective Repeat

Profesor:

Adrián Lara

Estudiantes:

Matías Rodríguez Singer - B15647

Andrea Gómez Montero - B32896

San Pedro, Montes de Oca

Jueves 13 de octubre ,2016

II Semestre 2016

Resumen

El presente proyecto pretende implementar una simulación del protocolo de la capa de enlace Selective Repeat con el fin de visualizar y comprender cómo funciona dicho algoritmo en la vida real. En efecto, la adición de un intermediario que intercepta y a veces elimina los datos (frames) que el servidor y el cliente intercambian provee una aproximación a la verdadera interacción que ocurre entre un emisor y un receptor en una red.
Palabras clave: selective repeat, emisor, receptor, intermediario, frame.

Descripción de Selective Repeat

Selective Repeat es un protocolo ARQ (Automatic Repeat-Request) diseñado para evitar la pérdida de datos (frames) durante la comunicación entre un emisor y un receptor.

En dicho protocolo, el emisor y el receptor poseen ventanas de un tamaño definido las cuales determinan el número de frames que pueden ser enviados y recibidos en sucesión. Cada frame tiene una identificación para ser distinguido.

Primeramente, el emisor envía por el medio en rápida sucesión cada frame que se encuentra en su ventana, conservando su orden. Por cada frame enviado se inicia un contador de timeout. Al recibir los datos, el receptor envía un Acknowledge (ACK) de vuelta al emisor por cada frame recibido, el cual comparte la identificación de dicho frame. Este ACK sirve para notificar al emisor que el frame con la misma identificación ha sido recibido. Si el ACK para el primer frame de la ventana es recibido por el emisor, se mueve dicha ventana de una posición para enviar el siguiente frame en la lista de datos a enviar. Si el siguiente frame ubicado al principio de la ventana también fue recibido, se repite el paso anterior. En caso contrario, la ventana permanece en esa misma posición hasta que el primer frame reciba su ACK. La ventana del receptor sigue una heurística similar para los frames que recibe. La única diferencia con la ventana del receptor es que como recibe frames en lugar de ACKs, lo que se revisa es si el primer frame de la ventana ha sido recibido.

El timeout determina la cantidad de tiempo que espera cada sección de la ventana del emisor para volver a mandar su frame. Si el timeout se termina, se asume entonces que el frame o el ACK fueron perdidos en su viaje y los datos deben ser re-enviados para asegurar su consistencia. En el caso de que el receptor reciba un

frame que ya había recibido anteriormente (esto ocurre cuando un ACK es perdido), un nuevo ACK con la misma identificación es enviado para comunicarle al emisor que el frame fue recibido. Esto es necesario para evitar ciclos y asegurar que ambas ventanas se mantengan sincronizadas.

Implementación del Selective Repeat

En esta implementación hay tres componentes principales que son el cliente (que actúa como el emisor de los datos descrito en la sección anterior), el servidor (que actúa como el receptor de los datos) y el intermediario que para efectos de esta simulación, es el componente que se encarga de calcular qué paquetes o ACKs se pierden en el medio.

La simulación del protocolo fue implementado en el lenguaje de programación de alto nivel Java. En este, se crearon las clases llamadas FileHandler y Frame las cuales se utilizaron para leer archivos y para simular los frames de datos que se envían en el protocolo respectivamente.

También, fue necesario importar algunas librerías. Entre las principales se encuentran: BufferedReader, InputStreamReader, PrintWriter, Socket, concurrent.TimeUnit, Scanner, Random, LinkedList y Queue, de estas, las primeras cuatro son para el uso y envío de datos de los sockets y las otras 5 se utilizaron para calcular el tiempo transcurrido de los timeouts, leer datos que el usuario ingresa en la línea de comandos, calcular el número random que utiliza el intermediario para calcular cuales paquetes se pierden y para el implementar las listas de datos (entre estas se encuentran las de datos por enviar, recibidos y que se encuentran en la ventana) respectivamente.

A continuación se explica con más detalle cada la implementación de cada uno de los componentes principales del protocolo:

- Cliente:

Como se mencionó anteriormente, este es el emisor que manda datos al servidor. Este le solicita al cliente, a través de la consola, que ingrese el tamaño de la ventana, qué archivo debe ser enviado, el puerto con el que se va a comunicar con el intermediario, el modo en que se quiere correr el programa (si es normal o debug) y el tiempo de timeout con el que se quiere correr la simulación.

Luego, se lee el archivo solicitado y a partir de este se crean:

I. La lista de datos por enviar (que quedan fuera de la ventana)
II. La ventana.
III. Se crea el socket, lo que hace que este se establezca la conexión con el intermediario. Para que esto funcione adecuadamente, se requiere que el intermediario se encuentre corriendo.
IV. Y los flujos de salida y entrada de datos.

A continuación, como ya se ha creado la conexión con el intermediario, se crea un ciclo que repite las siguientes tareas en orden:

A. Revisa si el ACK del primer frame de la ventana ya fue recibido. Si es así, corre la ventana de tal manera de que el nuevo primer elemento de dicha ventana, sea el siguiente frame en la ventana que no haya recibido ningún ACK previamente.
B. Luego, se crea un ciclo anidado, en el cual se revisa uno por uno si el timeout de los frames que se encuentran en la ventana se encuentra vencido. De ser así, se reenvía dicho frame al intermediario.

En este, también se revisa si el socket ha recibido algún ACK del intermediario y de ser así se sale del presente ciclo anidado para que de este modo se vuelva a ejecutar el punto A.

- Intermediario:

El intermediario, es el encargado de calcular cuales frames y ACKs se pierden en la conexión del cliente con el servidor. Por esto, este componente recibe frames del cliente y luego de determinar (mediante el cálculo de un random) que el frame no se perdió, debe enviarlos servidor. Funciona de manera similar con los ACKs, la diferencia con estos es que son recibidos del servidor y deben ser enviados al cliente luego de determinar si dicho ACK se perdió o no.

A continuación se explica cómo se implementó el intermediario:
Primero, y mediante el uso de la consola se le pide al usuario que ingrese el modo y los valores bajo los cuales desea correr la simulación (número de puerto por el que desea conectarse con el servidor, el número de puerto en el que desea conectarse con el cliente, el porcentaje (en enteros) de paquetes que deben perderse y por último si desea correr en modo debug o no.

Una vez que se tiene esto, el intermediario crea un socket de tipo cliente para establecer la conexión con el servidor en el número de puerto previamente indicado

(por esto se requiere que para que la simulación se ejecute adecuadamente, una instancia del componente servidor ya debe estar corriendo esperando una conexión en dicho puerto). También se crean sockets de tipo servidor para establecer la conexión con el cliente en el puerto previamente indicado y se espera a que el cliente se conecte.

Cuando el cliente crea la conexión en dicho puerto, el intermediario la acepta y se crean dos hilos:

I. Escucha la comunicación con el cliente y determina mediante un calculo random si la entrada se debe ser reenviada o no al servidor.
II. Escucha la comunicación con el servidor y determina mediante un calculo random si la entrada se debe ser reenviada o no al cliente.

Por último, una vez que reciba como entrada un solo un punto (.) de parte del cliente, se cierra la conexión con el servidor, se liberan ambos hilos y se termina la ejecución del programa.

- Servidor:

El servidor, representa al receptor de los datos en esta simulación, tal y como se mencionó con anterioridad.

Al comenzar la ejecución de este componente, se le solicita al usuario por medio de la consola que ingrese el modo, tamaño de la ventana y el número de puerto al que debe conectarse con el intermediario.

Luego, se crea un socket de tipo servidor para establecer la conexión con el intermediario. Se corre un método que crea una cola de frames vacíos que representan a la ventana donde se van a ir almacenando los frames conforme van llegando.

Una vez que el cliente envía una solicitud de conexión, el servidor la acepta y crea un variables que controlan tanto el flujo de entrada como de salida de la comunicación del socket (instancias de las clases BufferedReader y PrintWriter, respectivamente). Y a continuación, se comienza un ciclo que hace las siguientes tareas en orden:
Lee la comunicación que tiene con el intermediario y si:

a. La línea que lee es igual a un punto (.), sale del ciclo pues indica que ya no se van a recibir más datos.
b. Si es diferente a punto, quiere decir que lo que recibió fue un frame por lo que:
    i. Se separa el header (que en este caso, solo posee el ID del frame) de los datos (que en este caso es el último carácter de la hilera leída).

ii. Se envía un ACK indicando el ID del frame recibido.
iii. Se almacena el dato en el frame de la ventana con el ID que corresponda el ID del frame al que pertenece el dato. (En caso de que el primer frame de la ventana sea mayor al ID recibido, una vez que se envía el ACK como respuesta que aparece en el punto anterior, se ignora y se continúa con el siguiente).

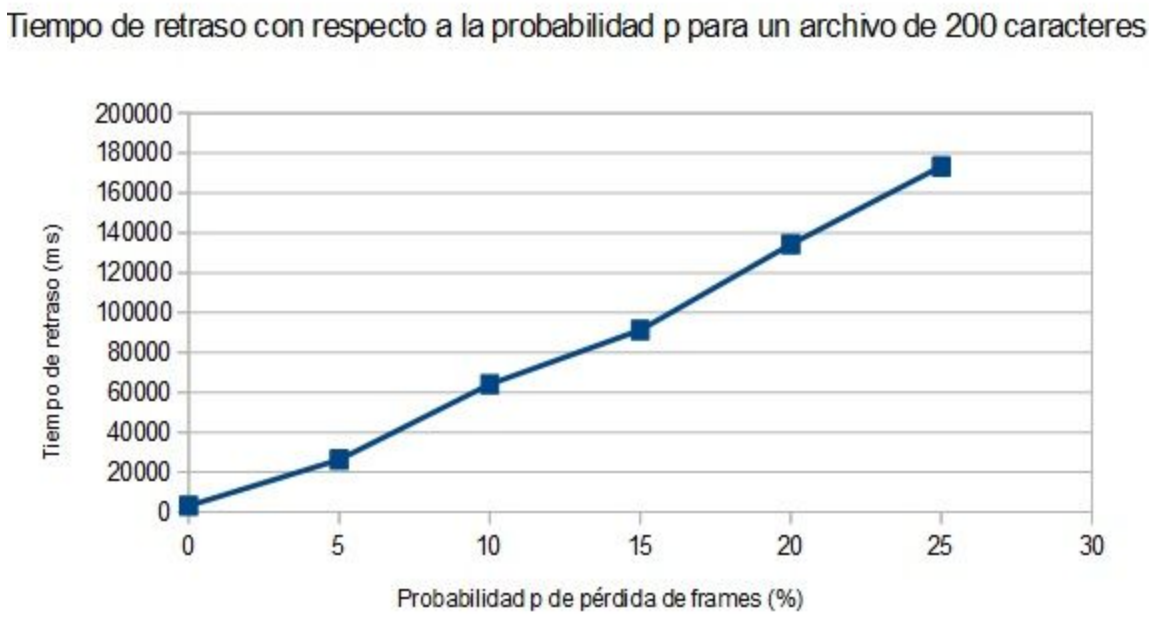
Una vez que se sale del ciclo, se cierra el socket la conexión, se procede a leer la cola de todos los frames recibidos, para meter sus datos en una hilera, guardarla en un archivo y se termina el programa.

Hay que tomar en cuenta que para el buen funcionamiento del algoritmo, la ventana del cliente y del servidor deben ser del mismo tamaño o podrían darse problemas.

Descripción de experimentos.

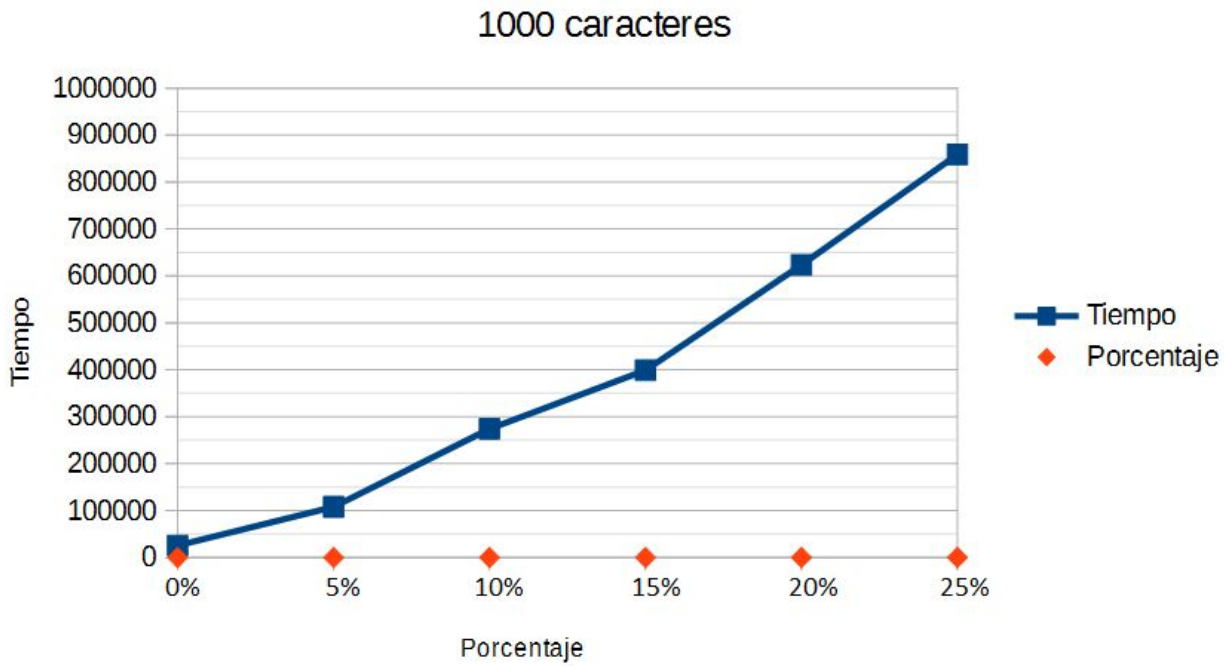
Con el objetivo de probar la funcionalidad del programa y observar los efectos de la pérdida de frames en nuestra simulación, se realizaron cuatro experimentos con tres tamaños de archivos distintos. El primer archivo contaba con 200 caracteres, el segundo con 500 y el último con 1000. La variedad de tamaños de archivos nos permitiría concluir con mayor veracidad sobre los efectos de la pérdida de frames en el protocolo selective repeat.

Para el primer experimento se ejecutó el programa seis veces en modo normal introduciendo una probabilidad p de pérdida de frames distinta para cada corrida. El file path proveído fue el del archivo de 200 caracteres. El tiempo de timeout introducido fue de 1000000 milisegundos y el tamaño de las ventanas fue de 4. Para cada tamaño de archivo, se corrió la simulación seis veces para probar la duración del mismo con los porcentajes de pérdida de paquetes de 0%, 5%, 10%, 15%, 20% y 20%. A continuación se muestra un gráfico que representa la progresión de la curva del tiempo de retraso con respecto a la métrica de la probabilidad p.

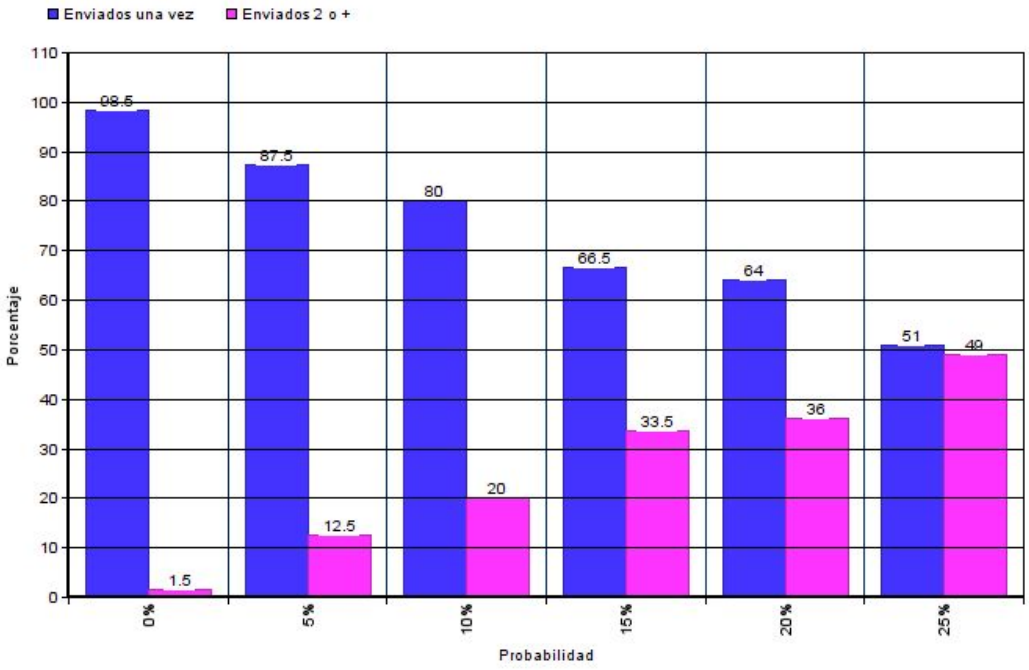


El segundo experimento se realizó de manera análoga al primero, con la única diferencia de que el archivo introducido tenía una longitud de 500 caracteres.

Igualmente, el tercer experimento se realizó de manera análoga al primero, esta vez con un archivo de 1000 caracteres.



En el cuarto experimento se ejecutó el programa seis veces para determinar con un contador la cantidad de frames que fueron enviados una sola vez y la cantidad de frames que tuvieron que ser enviados múltiples veces por el timeout durante cada corrida. El gráfico siguiente muestra la evolución de la proporción entre frames enviados una vez y frames enviados al menos dos veces para un archivo de 500 caracteres.



Resultados de experimentos.

Analizando los distintos gráficos presentados podemos concluir que una mayor probabilidad p de pérdida de frames incrementa consistentemente el tiempo de retraso en la simulación. En efecto, sin importar el tamaño del archivo introducido, la curva de los gráficos ve una evolución relativamente similar a la de la función identidad. El gráfico de la proporción entre frames enviados una y más de dos veces confirma esta suposición. Como puede observarse, a medida que la probabilidad de perder frames crece, la cantidad de frames que deben ser re-enviados incrementa progresivamente.
Este fenómeno es de esperarse puesto que el reenvío de frames en una red real incrementa considerablemente el tiempo de retraso de cada mensaje.

Conclusiones.

El protocolo selective repeat es un protocolo bastante sencillo de implementar y al enviar los ACK para confirmar que frames han sido recibidos es muy conveniente usarlo para saber con certeza que los datos enviados fueron recibidos. Sin embargo, como se puede apreciar en los gráficos conforme crece el porcentaje de frames perdidos en el canal, también crece proporcionalmente el número de frames enviados y el tiempo de retraso por lo que no es muy conveniente usarlo en caso de que el canal por el cual se transmiten los datos sea un canal con altas probabilidades de pérdida de paquetes y no es necesario tener tanta precisión con respecto a los paquetes que han sido enviados.