Universidad de Chile
Facultad de Cs. Físicas y Matemáticas
Departamento de Ciencias de la Computación
CC4303 Redes

Tarea 3

Profesor: José M. Piquer

Auxiliar: Catalina Álvarez, Tomás Wolf

Objetivos

Esta tarea persigue construir un protocolo bastante más eficiente para la transmisión de datos, basándose en su implementación de la tarea 2.

Usando como base Stop-And-Wait, deben implementar Go-Back-N, un protocolo de ventana de tamaño N en el lado del emisor, pero de tamaño 1 en el receptor; en esta tarea, tendrán la alternativa de optar por la implementación más simple, sin ningún agregado (con nota 5.0) o una versión con una optimización conocida como Fast Retransmit, para un 7.0. Eso sí, tanto los timeouts como ventanas serán de tamaño fijo en ambos caso y, nuevamente, no se consideran múltiples clientes (todos los paquetes corresponden, necesariamente, a la misma conexión), y se ignorará la etapa de conexión.

Características de la aplicación

La aplicación tiene exactamente la misma estructura que las tareas 1 y 2, ustedes simplemente deben preocuparse del control de envío de información; eso sí, hay varios puntos importantes que se deben aclarar:

- Tanto el header como el sistema de timeouts que implementaron para la tarea 2 queda igual. La única diferencia visible en el archivo *Data.h* es que ahora se define una variable extra, llamada **WIN_SZ**; este parámetro corresponde al tamaño, en número de paquetes, de la ventana del emisor.
- El funcionamiento de Go-Back-N es muy similar a Stop-And-Wait en el sentido en que los paquetes tienen que llegar ordenados al receptor; sin embargo, ahora no esperamos el ACK del paquete enviado para seguir avanzando, sino que enviamos tantos paquetes como espacio nos quede en la ventana (dado que un paquete no se puede dar por recibido hasta recibir su ACK).

- Los ACKs en Go-Back-N tienen un significado adicional. Dado que los paquetes se reciben orden, un ACK para el paquete N indica que todos los paquetes hasta el N llegaron correctamente. Es decir, incluso si se nos pierden muchos ACKS, basta uno para poder mover la ventana.
- Fast Retransmit es una optimización a Go-Back-N, que permite recuperarse más rápido frente a las pérdidas y evitar muchas retransmisiones inútiles. Su implementación es simple: cuando recibimos 3 ACKS duplicados, podemos concluir que se nos perdió el primer paquete de la ventana, por lo que la retransmitimos completa. Eso sí, deben tener cuidado de hacer este proceso una sola vez, para no terminar retransmitiendo varias veces por el mismo paquete perdido.

Para facilidad de desarrollo, `bwss` tiene incluido tres flags que se pueden poner antes de los parámetros (una flag adicional con respecto a la tarea 2): **-d** indica modo debug, que les permite obtener más información de lo que está haciendo el servidor proxy; **-L** seguido por un número entre 0 y 1 les permite incluir pérdida en la conexión (0.3 indica un 30 % de probabilidad de pérdida); y **-D** seguido de un número indica el delay **en segundos** que se le agrega al servidor (por ejemplo, un delay de 0.2 es equivalente a 200ms).

Un detalle importante extremadamente importante que necesitan conocer para la tarea es que hay que esperar tres cosas al mismo tiempo en el reader de TCP:

1. Leer desde el socket,
2. esperar timeouts para retransmitir la ventana y
3. esperar notificaciones de ACKS.

Para solucionar este problema, se les aconseja tomar en cuenta la siguiente observación: uno en general tiene solamente dos escenarios posibles, tengo o no tengo espacio en la ventana. Cuando uno tiene espacio en la ventana, los ACKS entrantes no son prioridad, y pueden procesarse luego de que se envíe todo lo posible; en el caso de que no tengo espacio en la ventana, no espero cosas del socket, sino que me preocupan los ACKS y los timeouts.

Características de la implementación

Ustedes deben implementar lo pedido basándose en su propia implementación de la tarea 2. Casos especiales deben conversarse directamente con los auxiliares.

La tarea puede ser hecha entre dos personas, siguiendo las reglas comentadas en clases: **una vez han trabajado con alguien, sólo tienen la posibilidad de seguir trabajando juntos o separarse**.

Cualquier duda o pregunta o reporte de bugs, dirigirse al foro de U-cursos.

Pruebas de eficiencia

Una vez terminada la tarea, se les pide probar su implementación con pérdida de 0, 10 % y 30 %, y con delays de 0, 100ms y 500ms. No es necesario que prueben todas las combinaciones posibles, pero sí que sean exigentes con su tarea (se recomienda al menos probar el valor máximo de pérdida con 100ms).

Al igual que en la tarea 2, se les recomienda cambiar los parámetros de dos formas distintas:

1. Utilicen las flags que provee el servidor, en particular en Anakena, donde no tienen permisos de superusuario.
2. **Muy recomendado, dado que así se realizará la evaluación**. Para asegurarse de que todo funcione bien utilicen "netem":

   ```
   % tc qdisc add dev lo root netem loss 20.0% delay 100ms
   ```

   Y tienen 20 % de pérdida con 100 milisegundos de delay. Para modificar el valor, deben usar **change** en vez de add en ese mismo comando.

Entrega

A través de U-cursos, **no se aceptan atrasos**. La evaluación de esta tarea es muy simple: el cliente debe funcionar bien con el servidor provisto.

Entreguen los fuentes, las instrucciones para compilar todo, y un LEEME.txt donde muestren los resultados que obtuvieron.