Tarea 2 Redes

Entrega: 17 de Noviembre 2014

1. Objetivos

Esta tarea persigue construir una capa de transporte simple sobre UDP, que permita un protocolo confiable usando Selective Repeat, con timeouts adaptables.

Para esto, deben basarse en su tarea 2, e implementar:

1. Selective Repeat: como visto en clases, ventanas fijas para enviador y para transmisor. Ahora los timeouts y los ACKs son para paquetes específicos, ya no puedo suponer que $ACKN \Longrightarrow ACKN-1$, esto quiere decir que siempre debo enviar el ACK del paquete que recibí correcto.

2. timeouts adaptables y límites: mantener el sistema de la tarea 2.

3. Ventana enviador: tamaño fijo de 50 paquetes, ventana receptor: fija de 50 paquetes. La retransmisión de la ventana ahora sólo implica retransmitir los paquetes que no han recibido ACK y a los que les toca un timeout.

4. ACK N: si N es el paquete más antiguo de la ventana de envío, deben avanzar la ventana. Si N no está en la ventana de envío, lo ignoran. Si está, deben anotarlo para no re-transmitirlo más tarde.

5. Fast retransmit: ahora ya no recibirán ACKs duplicados frente a fallas. Deben cambiarlo por más de 3 ACKs distintos al que Uds esperan (el del paquete más viejo de la ventana de envío). En ese caso, deben generar una retransmisión del paquete más viejo de la ventana (el primero), tal como si hubiese ocurrido un timeout.

6. Retries: igual a tarea2, pero ahora no habrán ACKs con un contador en cero, y todos pueden ser usados para calcular el RTT si el número corresponde al último enviado. En el caso de recepción de un paquete que no entra en la ventana de recepción (no está entre el que espero y 50 secuencias más), envíen un ACK con su número de secuencia y su número de retries, y el paquete mismo lo descartan.

7. Karn mejorado: Igual a la tarea 2.

2. Aplicación

Es un cliente y un servidor que intercambian un archivo para medir ancho de banda en ambas direcciones. Se les provee el fuente del cliente.

3. Capa Datos

Es todo igual que en la tarea 2, salvo la ventana de recepción, donde deben almacenar los paquetes recibidos fuera de rango, mientras estén menos de 50 números de secuencia más adelante del que esperan.

4. Ejecución

Para compilar la tarea, recomendamos hacer un Makefile. Mantengan la separación de las funciones de comunicación de la aplicación para que puedan reusar la aplicación para las próximas tareas.

Les proveo el ejecutable de un servidor `bws` y de un cliente `bwc`. Para probarlo deben correr el servidor primero (con opción de debug):

```
% ./bws -d
```

y luego el cliente:

```
% ./bwc -d archivo-in archivo-out ::1
```

Para probar con pérdidas, fuercen a localhost para que genere pérdidas aleatorias. En linux, usen "netem", usualmente basta hacer como superusuario (instalar paquete kernel-modules-extra):

```
% tc qdisc add dev lo root netem loss 10.0%
```

Y tienen 10 % de pérdida. Para modificar el valor, deben usar replace en vez de add en ese mismo comando.

Los valores recomendados para probar son con pérdidas entre 0 % y 20 %, y RTT entre 2ms y 200ms, con un ancho de banda total disponible de 1Mbps.

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

5. Entrega

A través de U-cursos, no se aceptan atrasos. La evaluación será en función de que logren mejorar las tasas de transferencia con parámetros de pérdida y delay en los rangos definidos, sin hacer demasiadas retransmisiones inútiles. Se hará una lista relativa de velocidad lograda contra retransmisiones inútiles y la mejor tarea tendrá un 7.0. La velocidad la pueden ver en el número que da el cliente, las retransmisiones inútiles en el DUP que va dando el servidor en modo debug. El código entregado DEBE compilar y ejecutar algo.

Al código, acompáñenlo de un archivo LEEME.txt donde explican los algoritmos utilizados y las mediciones que hicieron para validar que funcionan bien.