

# Tarea 2 Redes

Entrega: 20 de Octubre 2014

## 1. Objetivos

Esta tarea persigue construir una capa de transporte simple sobre UDP, que permita un protocolo confiable usando Go-Back-N, con timeouts adaptables.

Para esto, deben basarse en su tarea 1, e implementar:

1. Go-Back-N: como visto en clases, ventanas fijas y retransmisiones completas. Los números de secuencia van de 0 a 255.
2. timeouts adaptables: mejorar lo hecho en la tarea 1 si es necesario. Pueden implementar el algoritmo de TCP visto en clases.
3. timeouts límites: deben fijar como límite mínimo del timeout 50ms (0.05s) y como máximo 3 segundos.
4. Ventana enviador: tamaño fijo de 50 paquetes, Ventana receptor: fija de un paquete. La retransmisión de la ventana siempre implica retransmitirla completa.
5. ACK N: implementar que ACK para el paquete N implica para todos los menores que él. Como los números son cíclicos, deben revisar que esto implica ACK para los que están en la ventana de envío solamente (incluido el N que reciben).
6. Fast retransmit: si reciben más de 3 ACKs duplicados que no son el que esperan, deben generar una retransmisión de la ventana, equivalente a un timeout.
7. Retries: Se agranda el header en un byte más, que cuenta el número de retransmisiones de este paquete. En primer envío del paquete va con contador igual a uno. El ACK va con ese mismo número, el que corresponde al número de retransmisiones que venía en el paquete al que le hacemos ACK. Como enviamos ACKs para el último paquete recibido OK incluso al recibir paquetes fuera de rango, en ese caso el ACK va con un contador de retransmisiones en cero. Esto puede usarse como indicador de ACK duplicado para el Fast retransmit.
8. Karn mejorado: En vez de descartar los ACKs de paquetes retransmitidos para calcular el RTT, ahora sólo descartamos los ACKs que recibimos con un número de retransmisión distinto a la última retransmisión del paquete (esto incluye el contador cero, que siempre debe ignorarse).

## 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 1, salvo el contador de retransmisiones, que es un byte sin signo que va al final del header, el que queda ahora de tamaño 4 bytes (identificador de conexión, tipo, secuencia, retransmisión). El contador de secuencia ahora va entre 0 y 255.

### 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 adaptable `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.

Se les recomienda ir implementando las funcionalidades pedidas de a una, de modo de tener versiones preliminares funcionales que puedan entregar si les llega el plazo. El orden que recomiendo de implementación (por número de la funcionalidad) es: 1,3,4,5,7 (son básicos, no pueden probar la tarea sin eso) y luego, en orden: 8, 6, 2.