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

Tarea 1

Profesor: José M. Piquer

Auxiliar: Catalina Álvarez, Tomás Wolf

Objetivos

Esta tarea persigue construir una capa de transporte simple, basado en el cliente y servidor cuyo código fue visto en clases. La aplicación conecta un cliente y un servidor a través de dos servidores proxy; se les provee ejecutables de tres cuartas partes de la aplicación, más el código base de un cliente TCP como ejemplo. Ustedes deben programar el proxy del cliente.

La aplicación en sí misma es muy simple. Es un echo de archivos: el cliente debe leer un archivo, enviarlo, y el servidor se lo devuelve.

Características de la aplicación

A continuación se listan las características más relevantes para ustedes de la aplicación:

- La aplicación funciona en dos etapas:
  1. El cliente lee el archivo de entrada, y lo envía por bloques al servidor. Espera confirmación de que un bloque (o paquete) llegó antes de continuar con el siguiente.
  2. El servidor envía el archivo, un bloque a la vez; el cliente va escribiendo dichos bloques en un archivo de salda. El servidor espera confirmación de cada paquete antes de enviar el siguiente. Cuando el servidor ha terminado, el cliente debe desconectarse e informar el tiempo que tardó el envío ida y vuelta.
- Ustedes deben programar una capa intermedia entre dos aplicaciones cuyo código no conocen. La estructura base se puede ver en la siguiente figura:

Ustedes deben implementar un servidor proxy (llamado **bwcs** en el dibujo), que se conecte **bwc** a través de un socket TCP por el puerto 2001, y con **bwss** por un socket UDP

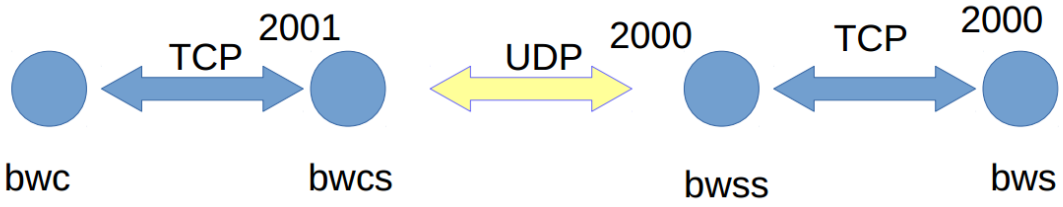

Figura 1: Estructura de la aplicación

por el puerto 2001. Ustedes no se deben preocupar para nada de la lectura del archivo a enviar (eso lo hace **bwc**) ni de la escritura (trabajo de **bws**), sino que solo deben leer de un socket y escribir en el otro el número de bytes que hayan recibido.

El archivo de prueba (cuyo tipo puede ser cualquiera, desde archivos de texto hasta videos) es leído por **bwc** y envíado como un paquete UDP; ustedes deben recibir estos paquetes y mandarlos tal como llegan: si reciben **X** bytes por el socket TCP, deben enviar los mismos **X** bytes por el socket UDP. Un detalle muy relevante es que, al comunicarse por UDP, los servidores proxies (no así cliente y servidor normales) **PUEDEN PERDER INFORMACIÓN ENTRE ELLOS**; esto significa que si el canal tiene pérdidas, el archivo de salida será más pequeño que el archivo original.

Finalmente, es importante recalcar que cliente y servidor **NO ESTAN SINCRONIZADOS**, por lo que nuestro servidor proxy puede recibir información por TCP y por UDP al mismo tiempo; eso significa que deben trabajar con threads independientes que lean y escriban de los sockets respectivos.

Características de la implementación

Ustedes deben implementar lo pedido en uno de los siguientes lenguajes de programación: **Java**, **C**, **C++** o **Python**. La posibilidad de utilizar otros lenguajes no está cerrada, pero debe conversarse 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 delay 0, 0.1 y 0.2, con pérdida de 0 y 10 %.

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 20.0% delay 100ms
```

Y tienen 20 % de pérdida y 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.