# Tarea 2: Cliente eco UDP para medir performance
# Redes
# Plazo de entrega: 21 de abril 2025

*José M. Piquer*

## 1. Descripción

Su misión, en esta tarea, es modificar el cliente TCP con 2 threads que hicieron en la T1, para que use una conexión UDP. El servidor que usaremos para la medición es: `server_udp3.py` (se provee como material docente, no está en el código estándar del curso).

Igual que en la T1, el cliente usa un archivo de entrada y otro de salida como archivos binarios (así pueden probar con cualquier tipo de archivo), y recibe como argumento el tamaño de las lecturas y escrituras que se harán (tanto de/desde el socket como de/desde los archivos).

De nuevo, deben modificar la detección de término ahora. Como se pueden perder datos, no es tan fácil como contar bytes enviados/recibidos. Lo que haremos es terminar el envío con un paquete UDP vacío (cero bytes) que hace de EOF. Cuando el receptor detecta este paquete, debe terminar la ejecución.

Como se pierden paquetes, el paquete final vacío también se puede perder. Entonces, muchas veces no detectarán el final. Para arreglar eso, deben definir un timeout máximo de espera de 3 segundos en el socket para el receptor, con `settimeout()`. Cuando ocurre este timeout deben terminar con un error y esa medición no es válida. Cuando no ocurre un timeout, pueden medir el tiempo de ejecución y usar el tamaño del archivo de salida como la cantidad de bytes transmitidos.

El cliente que deben escribir recibe el tamaño de lectura/escritura, el archivo de entrada, el de salida, el servidor y el puerto UDP.

```
./client_bw.py size IN OUT host port
```

Para medir el tiempo de ejecución pueden usar el comando `time`.

El servidor para las pruebas pueden correrlo localmente en local (usar localhost o 127.0.0.1 como "host"). Dejaremos uno corriendo en anakena también, en el puerto 1818 UDP.

Un ejemplo de medición sería (suponiendo que tengo un servidor corriendo en anakena en el puerto 1818):

```
% time ./client_bw.py 10 anakena.dcc.uchile.cl 1818 /etc/services  OUT
0.79 real         0.41 user         0.61 sys
```

Para enviar/leer paquetes binarios del socket usen send() y recv() directamente, sin pasar por encode()/decode(). El arreglo de bytes que usan es un bytearray en el concepto de Python (al ser binarios, no son strings).

## 2. Mediciones

El cliente sirve para medir eficiencia. Lo usaremos para ver cuánto ancho de banda logramos obtener y también probar cuánto afecta el largo de las lecturas/escrituras en la eficiencia.

Para probar en localhost necesitan archivos realmente grandes, tipo 0.5 o 1 Gbytes (un valor que demore tipo 5 segundos). Para probar con anakena, basta un archivo tipo 500 Kbytes.

Midan y prueben en las mismas condiciones que probaron la T1 y reporten sus resultados y las diferencias que obtengan.

Responda las siguientes preguntas (se pueden basar en los valores medidos en la T1 como referencia de los que son valores "correctos"):

1. Si el archivo de salida es más pequeño que el de entrada, ¿es correcto medir el ancho de banda disponible como tamaño recibido/tiempo?

2. Si el archivo de salida es más pequeño que el de entrada, ¿es correcto medir el ancho de banda disponible como tamaño enviado/tiempo?

3. Una medición que termina por timeout, ¿podría usarse de alguna forma para medir el ancho de banda disponible?

## 3. Entregables

Básicamente entregar el archivo con el cliente que implementa el protocolo, los scripts shell que utilizó para los experimentos y un archivo con los resultados medidos, tanto para localhost como para anakena, una comparación con la T1 y sus respuestas a las preguntas.

## 4. Strings y bytearrays en sockets

Una confusión clásica en los sockets en Python es la diferencia entre enviar un string y/o un bytearray. Partamos por los strings, que Uds conocen mejor: los strings no son simplemente arreglos de bytes (alguna vez lo fueron, pero hoy pueden contener hasta alfabetos asiáticos y árabes), son codificaciones en un *encoding* particular. Los sockets no soportan strings, es decir, Uds no pueden llegar y enviar/recibir un string por el socket, deben convertirlo a un bytearray, que es una colección de bytes binarios primitivos, no se interpretan. La forma de convertir un string a un bytearray es aplicando la función *encode()* y un bytearray a un string, con la función *decode()*. Ojo que si se aplican a cualquier cosa, pueden fallar, particularmente *decode()* falla si uno le pasa cualquier cosa en el bytearray. Entonces, si quiero enviar/recibir el string 'niño' hago:

```
enviador:
        s = 'niño'
        sock.send(s.encode('UTF-8')) # UTF-8 es el encoding clásico hoy
receptor:
        s = sock.recv().decode()     # recibe s == 'niño'
        print(s)
```

En cambio, si recibo bytes y quiero escribirlos en un archivo cualquiera, no sé si hay strings o no dentro, entonces mejor es no transformarlo y siempre usar bytes puros:

```
enviador:
    data = fdin.read(MAXDATA)
    sock-send(data)

receptor:
    data = sock.recv()
    fdout.write(data)
```

En esta tarea sólo usaremos bytearrays.