# Tarea 1 Redes: Cliente eco TCP para medir performance

**Nombre**: Patricio Espinoza A. **Sección**: 1

## Preparación

**Comando de uso general**

- $ ./client_bw.py size IN OUT host port

**Generar un archivo de 1 GB, uno de 500 KB y 100 archivos de 10.000 KB y 5KB (.bin)**

- $ python3 files_generator.py

**Ejecución del servidor local**

- $ python3 server_echo4.py 1818

**Limpiar cache**

- $ sudo sync && echo 3 | sudo tee /proc/sys/vm/drop_caches

## Ejecuciones y Resultados

### Localhost

➢ **Un archivo de 1000000 KB (1GB) → 1000000000 bytes**

$ time python3 client_bw.py ***size*** files_localhost/big_file.bin outputs/OUT_1 localhost 1818

- Manteniendo cache

| Size (bytes) | Tiempo real (s) | Ancho de banda (B/s) |
|---|---|---|
| 10 | 383,93 | 0,02 |
| 100 | 15,30 | 6,53 |
| 500 | 3,39 | 147,49 |
| 1.000 | 2,5314 | 395,03 |
| 10.000 | 2,5346 | 3.945 |
| 100.000 | 3,28 | 30.487 |
| 1.000.000 | 3,54 | 282.485 |
| 10.000.000 | 3,84 | 2.604.166 |

- Limpiando cache

| Size (bytes) | Tiempo real (s) | Ancho de banda |
|---|---|---|
| 10 | 326s | 0,03 |
| 100 | 16,81 | 5,94 |
| 500 | 4,13 | 121,06 |
| 1.000 | 2,70 | 370,37 |
| 10.000 | 2,57 | 3.891 |
| 100.000 | 2,61 | 38.314 |
| 1.000.000 | 3,15 | 317.460 |
| 10.000.000 | 3,07 | 3.257.328 |

- **100 archivos de 10.000 KB**

```
$ for i in {0..99}; do
  time python3 client_bw.py size files_localhost/file_$i.bin outputs/OUT_$i localhost 1818 &
done
wait
```

- Limpiando el cache

| Size (bytes) | Tiempo real (s) del último archivo | Ancho de banda |
|---|---|---|
| 10 | 65,5 s | 0,15 |
| 100 | 56,81 | 1,76 |
| 500 | 42,58 | 11,74 |
| 1.000 | 28,90 | 34,60 |
| 10.000 | 63,37 | 157,80 |
| 100.000 | 46,84 | 2.134,93 |
| 1.000.000 | 49,90 | 20.040,08 |
| 10.000.000 | 35,97 | 278.009,45 |

## Anakena

- **Un archivo de 500 KB**

$ time python3 client_bw.py ***size*** files_anakena/big_file.bin outputs/OUT_1 anakena.dcc.uchile.cl 1818

- Manteniendo el cache

| Size (bytes) | Tiempo real (s) | Ancho de banda |
|---|---|---|
| 10 | 0,760 | 13,16 |
| 100 | 0,731 | 136,80 |
| 500 | 0,759 | 658,76 |
| 1.000 | 0,647 | 1.545,60 |
| 10.000 | 0,628 | 15.923,57 |
| 100.000 | 0,824 | 121.359,22 |
| 1.000.000 | 1,239 | 807.102,50 |
| 10.000.000 | 1,538 | 6.501.950,59 |

- Limpiando cache

| Size (bytes) | Tiempo real (s) | Ancho de banda |
|---|---|---|
| 10 | 0,667 | 14,99 |
| 100 | 0,850 | 117,65 |
| 500 | 0,927 | 539,37 |
| 1.000 | 1,245 | 803,21 |
| 10.000 | 2,020 | 4.950,50 |
| 100.000 | 0,940 | 106.382,98 |
| 1.000.000 | 0,782 | 1.278.772,38 |
| 10.000.000 | 2,058 | 4.859.086,49 |

➢ **100 archivos de 5 KB**

```
$ for i in {0..99}; do
  time python3 client_bw.py size files_anakena/file_$i.bin outputs/OUT_$i
anakena.dcc.uchile.cl 1818 &
done
wait
```

- Limpiando el cache

| Size (bytes) | Tiempo real (s) del último archivo | Ancho de banda |
|---|---|---|
| 10 | 16,95 | 0,59 |
| 100 | 14,80 | 6,76 |
| 500 | 13,87 | 36,05 |
| 1.000 | 21,48 | 46,55 |
| 10.000 | 15,94 | 627,35 |
| 100.000 | 14,11 | 7.087,17 |
| 1.000.000 | 14,34 | 69.735,01 |
| 10.000.000 | 36,49 | 274.047,68 |

# Conclusiones

De acuerdo a los resultados, el ancho de banda tendió a incrementar en todos los experimentos realizados a medida que el ***size*** se aumentaba, lo que tiene sentido pues se envió una mayor cantidad de datos y los tiempos reales generalmente disminuyeron o se mantuvieron en un rango cercano. También se experimentó comparando el envío unitario de un archivo con y sin limpieza del cache de forma previa, sin embargo, no mostró una diferencia significativa.

Cuando el ***size*** era igual a 10 los tiempos fueron más altos ya que el archivo debía descomponerse en una mayor cantidad de paquetes y como cada uno tiene un costo de envío por su encabezado, esto disparo el tiempo real. En consecuencia, al incrementar el ***size*** el tiempo real tendió a la baja.

Un fenómeno interesante observado es que en ocasiones el tiempo real tendía a la baja mientras se aumentaba el ***size***, para que en la siguiente iteración con otro ***size*** mayor el tiempo real aumentará en vez de disminuir. Esto se destacó con amarillo en las filas correspondientes. Dentro de las posibles razones encontradas de este fenómeno se tiene que puede ser debido a:

- Fragmentación: Si un paquete supera el limite de transmisión de datos se fragmenta nuevamente en paquetes.

  [IP Fragmentation delay]

- Cuando se tienen múltiples paquetes en donde el tamaño aumento, se necesita más memoria para el envio y recepción de la información, y si esta está saturada, el cliente tendrá que esperar, y por ende el tiempo real aumenta.

  [Wikipedia Bufferbloat]
  [Bufferfloat Article]