1 Tarea 1: Cliente Eco UDP

Plazo de entrega: Viernes 1 de Octubre 2021

José M. Piquer

1.1 Motivación

Su misión, en esta tarea, es generar un cliente de eco sobre UDP que opere en forma equivalente al cliente 3 de eco TCP que está en los ejemplos del curso. Este cliente usa dos threads para que uno lea desde la entrada y envíe al servidor y otro para leer desde el servidor y enviar hacia la salida. Esto genera una gran carga sobre la red, ya que envía a toda la velocidad posible, sin nunca esperar las respuestas.

Siendo un socket UDP, habrá pérdida de paquetes, pero no importa mucho en este caso y se notará por que en la salida habrá menos información que en la entrada.

Un tema importante en los sockets UDP es qué tamaño de buffer usar cuando uno invoca send() y recv(). La documentación es un tanto vaga, hay un tamaño máximo absoluto de 65.535 bytes (un short integer si recuerdan su curso de Software de Sistemas), pero algunos sistemas limitan en el Sistema Operativo ese tamaño a números mucho más bajos.

En teoría es mejor usar paquetes grandes, envían más información por menos costo, y debieran ser más eficientes. Pero el tamaño puede afectar también la probabilidad de pérdida, y puede no ser tan bueno.

Esta tarea busca probar distintos valores; el cliente envía una propuesta de tamaño máximo y el servidor le responde con la suya. El valor a ser utilizado siempre debe ser la respuesta del servidor. Para poder probar eso, en vez de leer la entrada línea a línea, el cliente tiene que leerla en bloques de bytes, del tamaño del paquete máximo, para ser enviados. En Python, pueden usar sys.stdin.buffer.read1() para eso.

1.2 Ejercicio

Hemos inicializado un servidor eco UDP en `anakena.dcc.uchile.cl`, puerto `1818`. Usted deberá escribir un cliente UDP capaz de comunicarse con el servidor. Su ejecutable o script de Python deberá recibir dos argumentos, en el siguiente orden:

1. Un archivo de entrada (que puede generar usando una función que se propone más adelante).
2. Propuesta de tamaño máximo de paquetes para la comunicación, en número de bytes.

El archivo de entrada deberá ser leído por su cliente y enviado al servidor; las respuestas del servidor deberán ser impresas en la salida estándar.

Ejemplo de cómo su código deberá ser ejecutado:

```
$ python client.py archivo_de_entrada.txt 9000
```

1.2.1 Protocolo

Para poder comunicarse con el servidor, su cliente deberá implementar el siguiente protocolo:

```
% Inicio del "handshake" %

Cliente envía: "Hola"
Servidor responde: "OK"
Cliente envía: {TMaxClient}
Servidor responde: {TMaxServer}

% Fin del "handshake" %
% A partir de este punto, el cliente puede enviar paquetes arbitrarios. %

Cliente envía: ...
Servidor responde: ...
...
```

Es decir, su cliente deberá completar un "handshake" con el servidor, a través del cuál se establecerá el tamaño máximo de paquetes a usar en la comunicación. En este contexto, `{TMaxClient}` y `{TMaxServer}` corresponden a las propuestas de tamaño máximo en bytes del cliente y del servidor, respectivamente. Estos valores serán enviados como **strings** codificados como bytearrays en `UTF-8`; es decir, para – por ejemplo – enviar una propuesta de tamaño de `9 000` bytes, usted deberá construir un string con el texto "9000", convertirlo a un bytearray, y luego enviarlo.

Recuerde que luego del "handshake", el cliente debe enviar al servidor paquetes de tamaño igual o menor **al especificado por el servidor**. Se

recomienda ir almacenando todo lo recibido desde el servidor en un archivo para luego compararlo con el original y así detectar pérdidas.

1.2.2 Códigos útiles

En Python 3, usted puede utilizar una función como la siguiente para convertir un integer a un string como bytearray:

```
def int_to_string_bytearray(a: int) -> bytearray:
    return f'{a:d}'.encode('utf-8')
```

Para decodificar un string bytearray a un integer:

```
def string_bytearray_to_int(b: bytearray) -> int:
    return int(b.decode('utf-8'))
```

Para generar un archivo de un tamaño arbitrario en bytes, puede usar la siguiente función, que guardará un archivo de texto con el prefijo `paquetes_`:

```
def generate_packet(byte_size):
    s = '0'
    i = 0
    while len(s) < byte_size:
        i = (i + 1) % 10
        s += str(i)
    with open(f"paquetes_{byte_size}.txt", 'w') as f:
        f.write(s)
```

Para comparar dos archivos puede usar la función `cmp` del módulo `filecmp`, que retorna `True` si los dos archivos son iguales:

```
import filecmp

if filecmp.cmp(archivo_1, archivo_2):
    print("Los archivos son iguales.")
else:
    print("Los archivos son distintos.")
```

1.3 Entregables

Usted deberá entregar:

1. Su implementación del cliente eco UDP como script de Python o binario ejecutable (en dicho caso, deberá incluir también su código fuente).

2. Un documento PDF aparte, donde deberá responder las siguientes preguntas:
    1. ¿Puede ver si hay diferencia en las pérdidas (comparación) o en la eficiencia (tiempo) según el tamaño de paquete utilizado?
    2. ¿Qué tamaño de paquete recomienda? Argumente según sus experimentos.
    3. ¿Puede modificar el tamaño máximo del paquete UDP del socket en su sistema operativo? ¿Si es así, cómo?