Tarea 1: proxies multi-clientes UDP
Redes
Plazo de entrega: 30 de abril 2021

José M. Piquer

1 DESCRIPCIÓN

Estamos enfrentados a una red que nos conecta a Internet, pero donde nos han bloqueado todo el acceso a conexiones TCP hacia el exterior. Pero, el administrador olvidó bloquear UDP, pensando en que no sirve para acceso serio a Internet.

Su misión, en esta tarea, es lograr que el cliente del servidor de eco visto en los ejemplos (*client_echo2.py*) logre conectarse a su servidor (*server_echo2.py*) usando una red que sólo acepta UDP. Para esto, crearemos dos proxies, uno en la red interna (a quien se conecta el cliente), y otro en el servidor externo (quien se conecta al servidor). Como UDP puede perder datos, no es grave que el eco que recibimos no sea exactamente igual a lo que enviamos. Pero, en conexiones locales, normalmente no se pierde nada.

Para esto, se usa un proxy2 que actúa como servidor en la red externa y un proxy1 que actúa como cliente de proxy2 y como servidor para client_echo2.

El profe implementó una solución rasca, que soporta un sólo cliente a la vez. Es decir, si hay un cliente conectado al proxy1, el resto queda bloqueado esperando ser aceptado. Se les entrega el código de proxy1 y de proxy2 para ese caso. Ver la primera figura, al final del texto.

En esta tarea, se les pide modificar proxy1 para que soporte múltiples clientes, lo que lleva a modificar un poco el protocolo completo (y proxy2, pero esa versión la implementamos nosotros).

Para el esquema a montar, ver figura al final del documento.

2 ENTREGABLES

Básicamente entregar el archivo proxy1.py que implementa el nuevo protocolo.

3 SOLUCIÓN RASCA DEL PROFE

La solución que se les entrega incluye el proxy1 y el proxy2. El proxy2 es para que puedan probarlo, pero no es necesario que entiendan su implementación. Es interesante eso sí, por que usa unos trucos para permitir tener varios proxy1 como clientes y no mezclarlos. El importante para Uds es proxy1, por que es lo que tendrán que modificar o implementar de cero si prefieren.

El proxy1 entregado establece un socket TCP con proxy2, luego espera una conexión TCP. Cuando llega una, invoca la función proxy(), que crea un thread paralelo que ejecuta UDP_rdr() y él invoca TCP_rdr(). Entre ellos dos, se encargan de pasar el tráfico de paqueten hacia/desde proxy2. Si se fijan, no hay ningún protocolo ni información extra en los paquetes UDP, ya que la comunicación es uno a uno (desde un client2 a un server_echo2).

4 SOLUCIÓN BUENA SOLICITADA

Se les pide implementar un protocolo entre proxy1 y proxy2 que permita soportar múltiples conexiones entre client_echo2 y server_echo2 en paralelo. Para esto, le agregaremos un header a los paquetes UDP, con dos bytes: un tipo de paquete (un byte) y un identificador de la conexión (0-9, soportamos un máximo de 10 conexiones simultáneas).

Los paquetes entre proxy1 y proxy2 pueden ser de tres tipos diferentes:

- 'D': datos, son paquetes "normales" entre los programas de eco, el identificador de conexión indica a cual conexión corresponde.
- 'C': conexión: es un paquete solicitando conexión nueva o la respuesta a una conexión
- 'X': cierre de conexión: es un paquete que solicita cerrar esta conexión

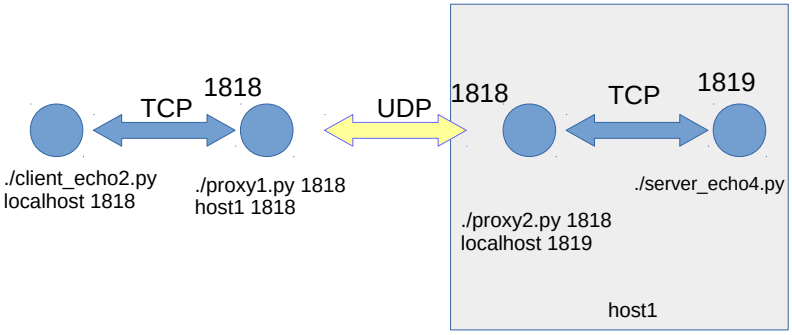
El flujo normal de una conexión es:

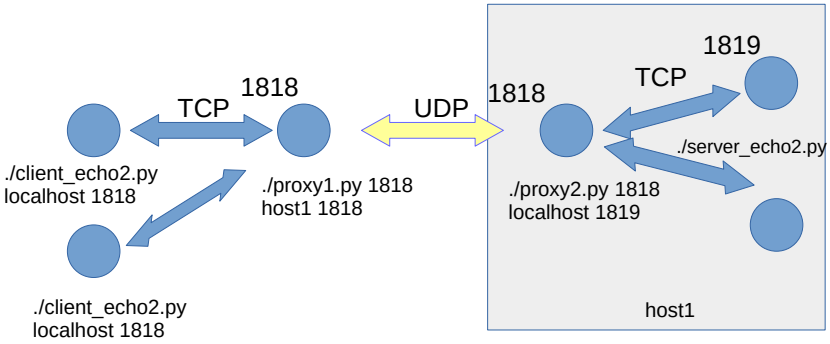
1. llega cliente nuevo de echo a proxy1
2. proxy1 busca un identificador de conexión que no esté en uso, entre 0 y 9. Digamos que elige 1.
3. envía paquete 'C1' a proxy2 indicando que quiere nueva conexión de identificador 1.
4. proxy2 recibe paquete de conexión y crea conexión TCP nueva al servidor server_echo.

5. si falla la conexión nueva, responde con un paquete CoNOK indicando que falló.
6. si funciona bien, responde con un paquete CoOK. Y la conexión está establecida
7. llegan datos por el socket TCP asociado al nuevo cliente, digamos 'hola'.
8. proxy1 escribe en el socket UDP: 'D1hola'
9. proxy2 recibe 'D1hola' y escribe 'hola' en la conexión TCP con el servidor de eco
10. el servidor de eco responde 'hola' en el socket TCP
11. proxy2 ahora escribe 'D1hola' en el socket UDP, que es la respuesta del servidor.
12. proxy1 lee 'D1hola' en el socket UDP, y escribe 'hola' por el socket TCP al cliente
13. proxy1 recibe None por el socket TCP cuando el cliente cierra el socket.
14. proxy1 escribe 'X1' por UDP para indicar fin de la conexión 1.
15. proxy2 lee 'X1' y cierra el socket TCP correspondiente.

5 SUGERENCIA DE IMPLEMENTACIÓN

Siguiendo un poco la idea de proxy1 como se les entrega, se sugiere crear un único thread UDP_rdr() que se encargue de leer desde el socket UDP, mientras cread un thread TCP_rdr() por cada cliente TCP echo_client2 conextado para que lea desde ese socket en particular. UDP_rdr() escribe en el socket TCP correspondiente a la conexión que recibe. TCP_rdr() escribe siempre en el socket UDP común.


TCP
1818
UDP
1818
TCP
1819
./client_echo2.py
localhost 1818
./proxy1.py 1818
host1 1818
./server_echo4.py
./proxy2.py 1818
localhost 1819
host1


1818
TCP
UDP
1818
1819
TCP
./server_echo2.py
./client_echo2.py
localhost 1818
./proxy1.py 1818
host1 1818
./proxy2.py 1818
localhost 1819
./client_echo2.py
localhost 1818
host1