Tarea 2: proxies UDP + Stop and Wait
Redes
Plazo de entrega: 4 de diciembre 2020

José M. Piquer

1 DESCRIPCIÓN

La tarea 1 logró que enviáramos paquetes UDP entre proxies y simular una conexión TCP. Pero, si se pierden datos en la conexión UDP, tenemos pérdida en la comunicación.

Su misión, en esta tarea, es lograr que el nunca se pierdan datos entre los proxies, implementando un protocolo Stop and Wait para garantizar la entrega ordenada y completa de los datos.

Para probar esto, les dejaremos el proxy "servidor" en anakena (puerto UDP/1818), junto con un servidor de eco (puerto TCP/1819). Uds deben implementar el proxy "cliente" y probarlo junto con el cliente de eco.

2 PROTOCOLO SOBRE UDP

Para poder implementar Stop and Wait sobre UDP, debemos generar un *header* que identifique los paquetes, con números de secuencia 0 y 1 y un ACK de vuelta que indique qué paquete recibí OK. También necesito un timeout para la retransmisión (lo definimos en 0.3 segundos) y un pequeño protocolo de conexión.

Para probar pérdida, podemos esperar a que la red pierda algo, pero eso es muy difícil de predecir, por lo que hemos implementado pérdida "simulada" en el servidor provisto, y Uds piden qué pérdida quieren usar dentro del protocolo de conexión.

2.1 *Protocolo Conexión*

Todo el protocolo consiste en headers (algunos van sólos, como los ACKs) que contienen bytes con sólo caracteres ASCII.

Como UDP implementa paquetes, no es necesario incluir el largo en el header, la función `recv` siempre me retorna el largo del paquete, preservando lo que el enviador envió con `send`.

El primer paquete UDP que Uds envían a nuestro servidor UDP debe ser siempre dos bytes:

CS

Ese paquete indica que queremos una conexión usando Stop and Wait. El servidor les responderá siempre con dos bytes:

A0

Que es un ACK con número de secuencia 0.

El segundo paquete UDP que Uds deben enviar es el nivel de pérdida que quieren tener:

Lx

En que x es un dígito (0-9), que se traduce a una probabilidad de pérdida multiplicándolo por 0.1:

- 0 es 0 probabilidad, sin pérdida
- 1 es 0.1 probabilidad (10 %)
- 2 es 0.2
- 3 es 0.3
- ...

El servidor les responderá con dos bytes:

A1

Que es un ACK con número de secuencia 1.

Después de eso, estamos listos para los paquetes de datos que se enviarán al servidor de eco.

2.2 *Protocolo Datos*

Todos los datos que lleguen desde el cliente de eco al proxy1, deben ser enviados al proxy2 precedidos por un header de 2 bytes que indica que es un paquete de datos y su número de secuencia (0 o 1):

D0adasdasdasd

Es el paquete secuencia 0, que contiene: adasdasdasd y que debe ser retransmitido (cada 0.3s) hasta recibir el Ack correspondiente:

A0

Dejamos un video con un terminal usando 'nc' como comando para probarlo a mano y entender mejor cómo funciona.

2.3 *cliente de eco de prueba*

Les dejamos un nuevo cliente de eco que es más agresivo y permite mejor probar pérdidas, llamado ./client_echo3.py. Básicamente genera un thread para enviar los datos y otro thread para recibir. Para detectar el fin de la transmisión, envía una línea especial al final y, cuando la recibimos de vuelta, decidimos que terminó el protocolo. Si lo ejecutan con una entrada desde un archivo, pueden guardar su salida y luego comparar ambos archivos, que debieran ser exactamente iguales. De hecho, ¡esta es una forma muy ineficiente de copiar un archivo!

3 RECOMENDACIONES DE IMPLEMENTACIÓN

El proxy1 debe manejar un flujo desde proxy1 a proxy2 y el flujo inverso desde proxy2 a proxy1. La recomendación para hacer esto bien es usar un thread para una dirección y otro para la otra. Eso permite que el thread se bloquee leyendo desde un socket y se concentre en los envíos en una dirección solamente. Pero hay que comunicar los ACKs recibidos hacia el otro thread, con una condición por ejemplo. Ver figura.

4 ENTREGABLES

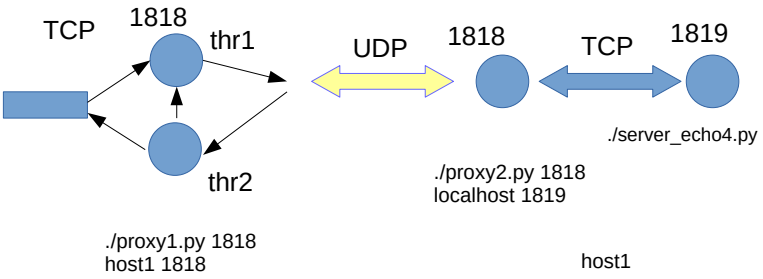
Básicamente entregar el archivo proxy1.py que implementa el protocolo explicado acá.

En un archivo aparte responder las preguntas siguientes:

1. Compare la eficiencia de hacer eco de un archivo usando el proxy y sin él (conectando el cliente directo al servidor). Puede probar en local y luego en la red. Comente las diferencias de eficiencia y por qué pueden ser.

2. Si queremos mejorar la eficiencia del protocolo y hacer más rápida la transferencia, qué sería lo primero que Ud modificaría en el protocolo? ¿Por qué?

3. Si queremos que el proxy2 corra en un servidor en Japón en vez de anakena, ¿qué es lo que más importa en nuestro protocolo para que no sea tan ineficiente?



Thr1: thread que lee desde socket TCP y escribe en el UDP
(y retransmite hasta que le avisen que se recibió ACK)

Thr2: thread que lee desde socket UDP y escribe en el TCP
(cuando recibe un ACK, debe informarlo a thr1)