Tomás Maturana
19.186.973-7

T2 Redes

1) Si conectamos el cliente directo al servidor, se hace una conexión mediante TCP, por lo que, asumiendo que queremos un eco correcto, la eficiencia de la comunicación es mucho mayor que haciéndolo mediante proxy1. Proxy1, al conectarse por UDP, debe reenviar paquetes perdidos para poder mantener el eco correcto y hacer de la comunicación, una comunicación *fiable.* Además, el protocolo que sigue proxy1 no está optimizado, lo que lo hace menos eficiente aún.
   Cabe recalcar que TCP es un protocolo orientado a la conexión y está optimizado para tener una comunicación fiable **y** eficiente. En cambio UDP es eficiente, pero no fiable, por lo que parchar lo no-fiable, requiere un algoritmo mejor que el que se plantea en el enunciado.

2) Cuando existe una alta probabilidad de pérdida de paquetes, la implementación actual del protocolo es muy ineficiente, debido al reenvío del segundo ACK por parte de proxy1. Estos son los pasos:
   - Recibir mensaje de client.
   - Transmitir el mensaje más el header hasta que llegue el ACK del proxy2.
   - Recibir el eco del proxy2 y enviarlo a client.
   - *Retransmitir el "ACK de ACK´s" al proxy2 hasta que deje de retransmitir el mensaje.
   - Repetir para siguientes mensajes de client.

   El último paso es necesario para poder mantener una comunicación con el proxy2, ya que éste reenvía el mensaje (eco) hasta que proxy1 le envíe el "ACK de ACK´s". En caso de no enviar el ACK de ACK´s, el proxy2 sigue retransmitiendo el mensaje infinitamente. Sin probabilidades de pérdida de paquetes, esto no genera mayores problemas, pero si se tiene una probabilidad de pérdida grande, se debe retransmitir el ACK de ACK´s hasta estar seguros de que el proxy-server dejó de retransmitir el mensaje… Algo que puede ser confundido fácilmente con la pérdida de paquetes!!

   La implementación actual de proxy1 hace un timeout proporcional a la probabilidad de pérdidas al momento de estar reenviando el ACK de ACK´s. El timeout propuesto es la probabilidad de pérdidas multiplicada por 10 segundos, es decir, si la probabilidad de pérdida es de 0.2, se esperan 2 segundos para estar *seguros* (casi, casi seguros…) de que proxy2 dejó de retransmitir el mensaje.

   Sin duda esto no es para nada eficiente a la hora de tener una pérdida de 0.9, ya que se deben esperar 9 segundos **al menos** para poder recibir nuestro eco. Por el lado contrario, si se espera menos, a la hora de enviar un nuevo mensaje, éste será descartado por proxy2.

   Para optimizar el protocolo, basta con modificar el proxy2 para que retransmita el mensaje hasta que le llegue un mensaje con el número de serie siguiente. Esto funciona ya que previamente se le pide al proxy-server que se desea un protocolo *stop&wait*, entonces, luego de que proxy2 envía el ACK, supongamos "A0", más el eco "D0xxxxx", y luego le llega

un paquete con header “D1”, implica que proxy1 recibió el eco. De este modo se elimina el último paso del algoritmo y se mantiene la comunicación fiable, mejorando la eficiencia notablemente.

3) El ancho de banda ocupado, ya que el tiempo de espera de cada ACK, más la probabilidad de pérdidas hará que se envíen mensajes con muy poca frecuencia. Idealmente se debería optar por cambiar el protocolo a uno que aproveche el ancho de banda como go-back-n. Aún así, si se quiere seguir utilizando este protocolo a pesar de la ineficiencia intrínseca, antes que todo habría que hacer la optimización planteada en la pregunta 2 y mejorar la implementación con un programador mejor :c…
   Otra opción es aumentar la frecuencia de retransmisión, de este modo se disminuye la probabilidad de pérdida de un paquete por unidad de tiempo.
   Una última opción que se me ocurre para seguir usando este protocolo y mejorar la eficiencia, sería conectar varios proxy1 al proxy2, de tal modo que juntos aprovechen el ancho de banda. Cada proxy1 enviaría mensajes distintos, por lo que sólo serviría si cada mensaje no es parte de un mensaje secuencial más grande… Es decir, si quiero enviar una lista de compras de tamaño N, donde no me importa el orden de llegada, puedo conectar N proxy1 al proxy2 y que cada uno envíe un elemento de la lista. En efecto, no serviría si quiero enviar un poema, ya que podrían llegar en cualquier orden al proxy2.