LABORATORIO 3

TRANSFERENCIA DE DATOS CONFIABLES RDT3.1

Propósito

Este proyecto es, más allá de la complejidad que puede tener su compresión, bastante sencillo. Se basa en el protocolo RDT3.1 analizado en clase. La cátedra le provee una red (*network.py*) inestable que presenta los problemas analizados (pérdida, duplicación y corrupción de paquetes).

Queda como tarea del alumno poder dar soporte a esta red resolviendo dichos problemas para que pase de ser una red de transferencia no confiable a una de transferencia confiable.

Las clases son:

Contantes.py

En primer lugar, se va a hacer empleo del archivo *constantes.py*. El mismo tiene apenas un par de definiciones para las direcciones de cada parte involucrada (*network*, *emisor* y *receptor*), pero a su vez, agrupa las clases propias de python que vamos a emplear en cada uno de los mencionados evitando así repetir dichas definiciones. Entonces, en resumidas cuentas, debe de incluir dicho archivo tanto en el *emisor* como en el *receptor*.

Paquete.py

En segundo lugar, se provee la clase *paquete.py*. La misma hace referecia a la unidad mínima de transferencia que permite la red. Cuando usted desee enviar un archivo a través de la red, debe de asegurarse que el tamaño de los datos a enviar en el paquete no exceda su capacidad. Este valor es modificable en el archivo *constantes.py*. Como parte de la correctitud del laboratorio, el mismo debe de poder funcionar modificando la longitud del paquete a diferentes valores.

Observe además, que se provee la funcionalidad necesaria para el cálculo de un checksum básico. Este cálculo va a ser modificado en las pruebas, por lo que debe de trabajar con la api en lugar de la implementación propiamente dicha.

Un paquete se conforma de los siguentes datos:

a. *Puerto Origen*: Es el puerto donde está escuchando el proceso de origen. Es necesario para que nos puedan enviar una respuesta.

b. *Puerto Destino*: Es el puerto donde está escuchando el proceso de destino a quien queremos enviarle un mensaje.

c. *Datos*: Parte del mensaje que estamos enviando. Recordar que los paquetes tienen un tamaño definido y posiblemente el mensaje sea mucho más grande que dicho tamaño. Por lo cual debemos de particinar el mensaje para que quepa dentro paquetes y los mismos sean enviados uno a uno mediante un *protocolo de parada y espera*.

d. *Número de Secuencia*: A fin de poder hacer un correcto seguimiento de los mensajes que estamos enviando, cada uno debe de llevar impreso un número de secuencia que nos va a permitir ubicar los mensajes entre todos los que enviemos.

Network.py

La parte más complicada de entender pero más fácil de usar, es la clase *Network.py*. Al igual que toda red, debe de estar operativa antes de poder enviar mensajes.

Esta clase opera con sockets UDP, no con TCP. Ya se han estudiando los sockets UDP a comienzo de la capa de transporte. En caso de dudas al respecto, se recomienda una lectura.

Los mensajes con los cuales nos comunicaremos tienen la siguiente estructura

```
( (NETWORK_IP, NETWORK_PORT), Paquete )
```

Observe que es un tupla formada por un par `(NETWORK_IP, NETWORK_PORT)`, es decir la *dirección IP* y el *puerto* donde la red está escuchando. Recuerde que al usar un protocolo de transferencia no orientado a la conexión, el mensaje debe de incluir dichos valores específicamente. Y por el `Paquete` propiamente dicho.

Como ayuda, son necesarias dos funciones para el envío/recepción de mensajes a través de un puerto UDP:

a. *dumps*: que toma un par `(destinatario, mensaje)` y lo comprime para ser enviado.

b. *loads*: que toma una respuesta desde un socket UDP y lo descomprime.

Tenga en cuenta, que la red es una que presenta problemas. Tanto la duplicación, como la pérdida y la corrupción son posibles a través de esta red.

Procedimientos y Detalles

1. `Emisor.py`: La primer clase a elaborar. El mismo simplemente abre un archivo de texto y haciendo uso de la red provista, envía el archivo al *emisor*. Debe de agregar una manera de poder avisarle al receptor el fin del archivo transmitido. El archivo a enviar debe de ser ingresado como parámetro.

2. `Receptor.py`: La segunda clase a elaborar. Deberá de estar corriendo como un servidor (es decir, desde antes de que el emisor.py desee enviarle un mensaje) y escuchando en un puerto UDP recibir el mensaje que le llegue desde la clase *emisor.py*. Además, deberá de guardar una copia de lo recibido. El nombre del archivo debe de ser ingresado como parámetro.

3. Recuerde que la red presenta problemas propios de un protocolo no orientado a la comunicación. Pero sólo los presenta en los mensajes enviados desde el emisor hacia el receptor (los mensajes enviados desde el receptor al emisor, no presentan problema alguno). Para ello, deberá de implementar los elementos introducidos en el protocolo *rdt3.1* que ayudan a una comunicación sin problemas. Deberá estudiar los casos y proveer los recursos para un correcto funcionamiento.