LABORATORIO 2

TRANSFERENCIA DE DATOS CONFIABLE RDT1.1

Propósito

Aprender a usar la API de sockets provista por Python. Usar clases para haces uso del objeto Paquete

Generalidades

Escriba y testee un programa `emisor_rdt1.1` y un programa `receptor_rdt1.1` en una misma computadora. Luego ejecute ambos programas en computadoras separadas, y verifique que se pueden comunicar. Para ello, consideramos el protocolo analizado en clases `rdt1.0`.

Para este proyecto, deberá modificar el Laboratorio 1, de manera que ahora permita crear, enviar y recibir **paquetes**, una nueva clase de objetos provista por la cátedra, en el archivo `paquete.py`. Estudie el programa, `paquetesTest.py`, para entender su correcto uso.

A su vez, deberá hacer uso de constantes de configuración que provee el archivo `constantes.py`. En la misma se presentan valores que vamos a usar tanto en el emisor como en el receptor. La idea de este archivo, es que nos permita definir dichos valores en un sólo lugar y luego poder usarlos en otros lugares mediante sus variables.

Procedimientos y Detalles

1. Investigue la API de sockets provista por el lenguaje de programación Python. Para ello deberá poner especial énfasis en los sockets que hacen uso del protocolo UDP. El capítulo 2.7 provisto por la cátedra es la lectura sugerida al respecto.

2. Desarrolle un programa `emisor_rdt1.1.py` que deberá aguardar infinitamente por mensajes que la capa de aplicacion leerá desde consola para luego enviarlos a traves de un puerto UDP al receptor. Para ello, debería implementar las siguientes funcionalidades:

   a. `socket = create_socket(RECEIVER_IP, RECEICER_PORT)`: Crea un socket con las direcciones provistas como parámetros.

   b. `rdt_send()`: Envía los datos leídos desde la capa de aplicación a la capa de transporte

   c. `packet = make_pkt(data)`: crea el paquete en la capa de transporte. Para este segundo laboratorio, el paquete va a contener las direcciones de puertos, tanto de emisor como receptor, la longitud de los datos así como el checksum, además de los datos. Observe que crear el paquete no crea el checksum sino que lo setea como valor por defecto a 0.

   d. `udp_send(socket, packet)`: Envía el paquete a traves de UDP, al servidor. Esta función deberá implementarse haciendo uso de la librería `pickle`. La función `datos = dumps(packet)` genera los datos que se pueden enviar a través del socket, a partir de un paquete.

   e. `close_socket(socket)`: Cierra el socket pasado como parámetro.

3. Desarrolle un programa `receptor_rdt1.1.py` que deberá aguardar infinitamente por mensajes de la capa de aplicación. El mismo deberá implementar las funcionies:

   a. `socket = create_socket(RECEIVER_IP, RECEICER_PORT)`: Crea un socket con las direcciones provistas como parámetros.

   b. `packet = udt_rcv(socket)`: permite recibir un paquete a traves de un socket UDP. Para ello también deberá hacer uso de la libreria `pickle`, y la función `packet = loads(data)`, que toma como argumento los datos recibidos por medio del socket, y los vuelve a convertir en un paquete.

c. `checksum = calculate_checksum(packet)`: Esta función calcula el checksum de un paquete recibido. Como se analiza en el teórico, este valor debe de ser igual a 0 (cero). (**Esta función ya se encuentra implementada, sólo debe hacer uso de la misma.**)

d. `data = extract(packet)`, tal que extrae del paquete la información contenida en el mismo;

e. `deliver_data(data)`, que recibe datos y, por el momento, simplemente los imprime por pantalla.

f. `close_socket(socket)`: Se encarga de cerrar el socket creado.

5. Realizar un test de loopback con el `servidor_rdt1.0.py` y el `cliente_rdt1.0.py` ejecutándose en una misma computadora como se explicó en clases.

6. Testear el cliente y el servidor ejecutando el primero en una computadora y el segundo en otra.

7. Modificar el servidor para que no sólo imprima el mensaje en pantalla, sino que además guarde un registro de todas las palabras recibidas y que haya impreso.

8. Como los puertos de conexión son de carácter efímeros, vamos a requerir pasar por parámetro dicho valor. Como también el servidor se va a ejecutar en diferentes máquinas, el cliente debe de poder recibir no sólo el puerto como parámetro, sino también la dirección ip del servidor con quien intenta establecer comunicación.