

Redes de Computadores

Tarea 3

No somos nada, ¡Hola Internet! (Capa de Red)

Integrantes:

Guillermo Fernández	Álvaro Rojas
guillermo.fernand.12@alumnos.usm.cl	alvaro.rojasv@alumnos.usm.cl
201073523-0	201073555-9

Profesor:

Oscar Encina

Ayudantes:

Alex Arenas
Carlos Marchant

02 de Junio, 2014

1. Introducción

En este laboratorio, se analizará qué es lo que ocurre con los paquetes a medida que viajan por la gran red de redes que supone Internet, y además, se conocerá cómo funciona el algoritmo vector-distancia con el que los routers completan sus tablas de costos.

1.1. Objetivos

- *Evidenciar la real dimensión de la internet y analizar su interconexión.*
- *Conocer el funcionamiento del algoritmo de enrutamiento llamado Algoritmo de Vector-Distancia.*
- *Recordar la programación en Python.*

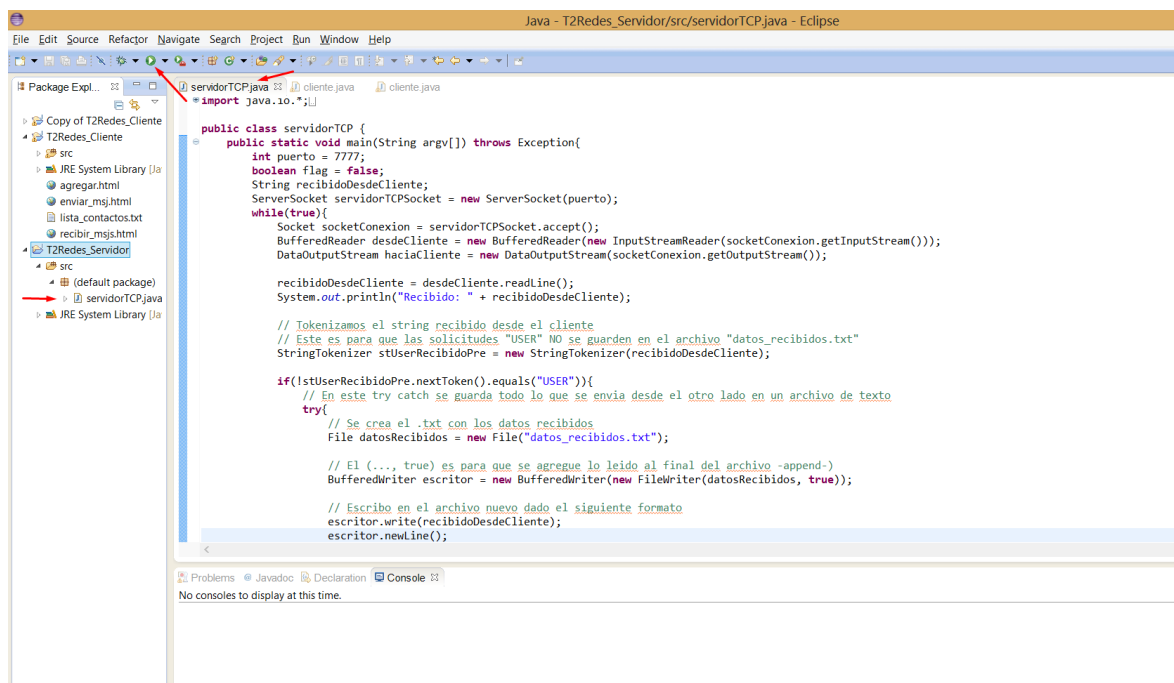
2. Desarrollo

2.1. Las dimensiones de Internet

En esta sección, se han analizado una serie de sitios web que, mediante el ingreso de sus URL al programa *Open Visual Traceroute*, se puede conocer qué camino toman los paquetes a través del planeta, evidenciándose así que existen saltos tan extraordinarios como lo que significa ir de un continente a otro. Los sitios web analizados son los siguientes:

1. <http://moodle.inf.utfsm.cl/>
2. <http://google.cl/>
3. <http://cime.cl/>
4. <http://wikipedia.com/>
5. <http://www.chile.embassy.gov.au/>

Para realizarlo, se ejecuta el código de la siguiente forma. En primer lugar, se ejecuta *servidorTCP.java*:



```
Java - T2Redes_Servidor/src/servidorTCP.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

T2Redes_Servidor
  src
    servidorTCP.java
  T2Redes_Cliente
  T2Redes_Servidor

public class servidorTCP {
    public static void main(String argv[]) throws Exception{
        int puerto = 7777;
        boolean flag = false;
        String recibidoDesdeCliente;
        ServerSocket servidorTCPSocket = new ServerSocket(puerto);
        while(true){
            Socket socketConexion = servidorTCPSocket.accept();
            BufferedReader desdeCliente = new BufferedReader(new InputStreamReader(socketConexion.getInputStream()));
            DataOutputStream haciaCliente = new DataOutputStream(socketConexion.getOutputStream());

            recibidoDesdeCliente = desdeCliente.readLine();
            System.out.println("Recibido: " + recibidoDesdeCliente);

            // Tokenizamos el string recibido desde el cliente
            // Este es para que las solicitudes "USER" NO se guarden en el archivo "datos_recibidos.txt"
            StringTokenizer stUserRecibidoPre = new StringTokenizer(recibidoDesdeCliente);

            if(!stUserRecibidoPre.nextToken().equals("USER")){
                // En este try catch se guarda todo lo que se envia desde el otro lado en un archivo de texto
                try{
                    // Se crea el .txt con los datos recibidos
                    File datosRecibidos = new File("datos_recibidos.txt");
                    // El (... , true) es para que se agregue lo leído al final del archivo -append-
                    BufferedWriter escritor = new BufferedWriter(new FileWriter(datosRecibidos, true));
                    // Escribo en el archivo nuevo dado el siguiente formato
                    escritor.write(recibidoDesdeCliente);
                    escritor.newLine();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Figura 1: Iniciando servidor

Se escogió un servidor con protocolos TCP en vez de UDP debido a que se busca la integridad de los mensajes y archivos enviados.

Luego, se inician los dos clientes con diferentes puertos entre ellos y con el servidor:

De esta forma, posteriormente se puede acceder a mandar solicitudes de conexión y aceptarlas para poder mandar mensajes entre clientes a través del servidor TCP.

2.2. Conectar ambos clientes

Mediante la agregación de contactos realizado en la Tarea 1, los clientes quedan "conectados" además de registrados en la lista de contactos.

2.3. Captura de paquetes con Wireshark

Mediante la herramienta RawCap, que permite monitorear interfaces loopback (dado que se utilizó solo un computador para la ejecución, lo que quiere decir que el envío de mensajes es entre localhost y localhost) y permite generar un archivo *.pcap* que Wireshark puede leer, se puede generar un registro de los paquetes transferidos, y luego ser visualizados con Wireshark:

2.4. Enviar un mensaje entre clientes

Para enviar un mensaje entre clientes, se identifica quien es el usuario emisor, y a quién va dirigido el mensaje, además del mensaje propiamente tal.

El otro cliente tiene la posibilidad de ver los mensajes recibidos mediante la opción habilitada en la página.

2.5. Enviar una respuesta entre clientes

Para enviar una respuesta entre clientes, se identifica quien es el usuario, y a quién va dirigida la respuesta, además del mensaje propiamente tal.

El otro cliente tiene la posibilidad de ver las respuestas recibidos mediante la opción habilitada en la página.

2.6. Mostrar en Wireshark los mensajes enviados

En la figura 11, se observan los números de puertos de origen y de destino de los paquetes enviados. En el caso del puerto de origen, se ve que corresponde al 65172. Este puerto correspondería a un socket perteneciente al puerto 6666 que se ha definido en el código Java. El número 7777 corresponde al puerto en el que está escuchando el

servidor TCP.

El socket asociado al puerto de cliente no tiene por qué ser el mismo, ya que a medida que se hace la transferencia de paquetes, el socket asociado a los puertos de cliente va cambiando, tal como se verá en la siguiente captura.

En la figura 12 se puede observar un mensaje desde el servidor al cliente; en este caso, el puerto de origen es el del servidor que es el 7777, y el de destino es 65181, que vendría a ser un socket asentado en el puerto 6666 que se definió en el código Java para el cliente.

Además, en esta imagen se pueden observar las IP de origen y destino de los paquetes, que en este caso son ambas iguales a 127.0.0.1, ya que el envío de mensajes se hizo en el mismo computador, o sea, desde *localhost* a *localhost*.

En cuando a los números de secuencia y ACK's, en TCP se relacionan con la transferencia de datos fiable. Los números de secuencia, corresponden al primer byte numerado de cada segmento, y el ACK es el número de secuencia del siguiente byte que el otro host está esperando por recibir. En este sentido, al comienzo de la transmisión de paquetes, la primera señal que se envía es un SYN (sincronización de números de secuencia), y luego se puede observar que en el caso de envío desde cliente a servidor, el número de secuencia va tomando el mismo valor del ACK anterior, mientras que el ACK toma el valor 1. Cuando el envío es desde servidor a cliente, ahora el número de secuencia es 1 y el ACK es uno más el mismo número de secuencia del envío servidor-cliente. Cabe destacar que la señal PSH quiere decir que no se ha esperado que el buffer se llene para enviar, sino que el paquete se envió de inmediato.

2.7. Diagramas de conexiones con los puertos, sockets e IP's

El resultado del análisis del punto 2.6 usando *Wireshark* es: