**Maestría en Ciencias y Tecnologías de la Computación**

**Sistemas Distribuidos**

**Nombre:** Luis Darío Sinche Cueva

**Taller:** Programación en Sockets

**Objetivo:**

- Entender la diferencia entre sockets con protocolo TCP y protocolo UDP

- Asimilar la abstracción por capas en programación en Redes

**Tarea:**

**1. Definir un software cliente y un software servidor para que envié un saludo y sea retroalimentado por una respuesta del servidor. Utilice como lenguaje de programación java. Ejecute los dos programas en el mismo equipo.**

**CÓDIGO SERVER TCP**

package sockets;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.lang.ClassNotFoundException;

import java.net.ServerSocket;

import java.net.Socket;

public class SOCKETS {

//static ServerSocket variable

private static ServerSocket server;

//socket server port on which it will listen

private static int port = 9876;

public static void main(String args[]) throws IOException, ClassNotFoundException{

//create the socket server object

server = new ServerSocket(port);

//keep listens indefinitely until receives 'exit' call or program terminates

while(true){

System.out.println("Waiting for client request");

//creating socket and waiting for client connection

Socket socket = server.accept();

//read from socket to ObjectInputStream object

ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());

//convert ObjectInputStream object to String

String message = (String) ois.readObject();

System.out.println("Message Received: " + message);

//create ObjectOutputStream object

ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());

//write object to Socket

oos.writeObject("Hi Client "+message);

//close resources

ois.close();

oos.close();

socket.close();

//terminate the server if client sends exit request

if(message.equalsIgnoreCase("exit")) break;

}

System.out.println("Shutting down Socket server!!");

//close the ServerSocket object

server.close();

}

}

**CÓDIGO CLIENTE TCP**

package socketservertcp;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.InetAddress;

import java.net.Socket;

import java.net.UnknownHostException;

public class SOCKETSERVERTCP {

public static void main(String[] args) throws UnknownHostException, IOException, ClassNotFoundException, InterruptedException{

//get the localhost IP address, if server is running on some other IP, you need to use that

//InetAddress host = InetAddress.getLocalHost();

String host = "172.17.171.164";

Socket socket = null;

ObjectOutputStream oos = null;

ObjectInputStream ois = null;

for(int i=0; i<5;i++){

//establish socket connection to server

//socket = new Socket(host.getHostName(), 9876);

socket = new Socket(host, 9876);

//write to socket using ObjectOutputStream

oos = new ObjectOutputStream(socket.getOutputStream());

System.out.println("Sending request to Socket Server");

if(i==4)oos.writeObject("exit");

else oos.writeObject(""+i);

//read the server response message

ois = new ObjectInputStream(socket.getInputStream());

String message = (String) ois.readObject();

System.out.println("Message: " + message);

//close resources

ois.close();

oos.close();

Thread.sleep(100);

}

}

}

**SERVER UDP**

package udpserver;

import java.net.\*;

public class UDPServer {

public static void main(String args[]) throws Exception

{

DatagramSocket serverSocket = new DatagramSocket(9876);

byte[] receiveData = new byte[1024];

byte[] sendData = new byte[1024];

while(true)

{

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

serverSocket.receive(receivePacket);

String sentence = new String( receivePacket.getData());

System.out.println("RECEIVED: " + sentence);

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =

new DatagramPacket(sendData, sendData.length, IPAddress, port);

serverSocket.send(sendPacket);

}

}

}

**CLIENTE UDP**

package udpclient;

import java.io.\*;

import java.net.\*;

public class UDPClient {

public static void main(String args[]) throws Exception

{

BufferedReader inFromUser =

new BufferedReader(new InputStreamReader(System.in));

DatagramSocket clientSocket = new DatagramSocket();

InetAddress IPAddress = InetAddress.getByName("172.17.171.164");

byte[] sendData = new byte[1024];

byte[] receiveData = new byte[1024];

String sentence = inFromUser.readLine();

sendData = sentence.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

clientSocket.send(sendPacket);

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

String modifiedSentence = new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);

clientSocket.close();

}

}

Una vez finalizado responda las siguientes preguntas:

**a) Cuál es la diferencia entre el Socket TCP y el Socket UDP**

|  |  |
| --- | --- |
| **TCP: Transmission Control Protocol** | **UDP: User Datagram Protocol** |
| El protocolo TCP está orientado a la conexión. Flujo bidireccional. La transferencia se realiza previniendo al receptor que el Emisor desea comunicarse. Y el receptor envía respuesta al emisor. | No orientado a conexión. El flujo es unidireccional. La transferencia de datos se realiza sin prevenir al destinatario, y el destinatario recibe los datos sin enviar una confirmación al emisor |
| La velocidad de trasferencia de datos, debido a la recuperación de errores y organización de paquetes, es menos que UDP. | La velocidad de transferencia de datos es mejor que TCP. |
| Re ordena los paquetes de datos en el orden especificado. | No tiene un orden inherente ya que todos los paquetes son independientes entre sí. Si se requiere un pedido, debe ser administrado por la capa de aplicación. |
| Realiza la comprobación y recuperación de errores. Los paquetes erróneos se retransmiten desde el origen al destino. | Realiza la comprobación de errores, pero simplemente descarta los paquetes erróneos. No se intenta recuperar el error. |
| Admite bloques de datos (cuyo tamaño puede ir desde 1 bytes hasta muchos Kbytes, dependiendo de la implementación) que serán empaquetados de forma transparente antes de ser transmitidos. | Necesita que le entreguemos paquetes de datos que el usuario debe construir |
| La función se da a medida que un mensaje se abre paso a través de internet de una computadora a otra | Es un protocolo de transporte de mensajes |
| Protocolos: HTTP, HTTPs, FTP, SMTP, Telnet | Protocolos: DNS, DHCP, TFTP, SNMP, RIP, VOIP. |
| Requiere tres paquetes para configurar una conexión de socket. | No hay solicitud de mensajes ni conexiones de seguimiento. |

**b) Cuál es el número de puerto en el servidor, y como se genera el número de puerto en el cliente.**

El puerto en el servidor elegido es 9876 de igual manera para el puerto en el cliente.

**c) Seleccione las líneas de código que se utilizan para la comunicación tanto con TCP como con UDP.**

|  |  |
| --- | --- |
| **TCP** | **UDP** |
| Socket socket = server.accept();  //read from socket to ObjectInputStream object  ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());  //convert ObjectInputStream object to String  String message = (String) ois.readObject();  System.out.println("Message Received: " + message);  //create ObjectOutputStream object  ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());  //write object to Socket  oos.writeObject("Hi Client "+message);  //close resources  ois.close();  oos.close();  socket.close(); | DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);  serverSocket.receive(receivePacket);  String sentence = new String( receivePacket.getData());  System.out.println("RECEIVED: " + sentence);  InetAddress IPAddress = receivePacket.getAddress();  int port = receivePacket.getPort();  String capitalizedSentence = sentence.toUpperCase();  sendData = capitalizedSentence.getBytes();  DatagramPacket sendPacket =  new DatagramPacket(sendData, sendData.length, IPAddress, port);  serverSocket.send(sendPacket); |

**2. Con el apoyo de un colega, ejecute el código cliente y el servidor en diferentes equipos. Haga los cambios necesarios para que el software funcione. Una vez que tenga comunicación, utlice wireshark para contestar las siguientes preguntas:**

**a) Cuál es la dirección IP y el número de puerto del cliente y cuales son del servidor.**

Servidor 172.17.171.164

Puerto: 9876

Cliente 172.17.171.255

Puerto: 9876

**b) Identifique la diferencia en los campos de la cabecera tanto del protocolo TCP como del protocolo UDP.**

El tamaño de cabecera de UDP es de 8 bytes, y el de TCP es el doble. El tamaño de la cabecera TCP es de 20 bytes y contiene algunas opciones como relleno, suma de comprobación, banderas, desplazamiento de datos, número de confirmación, número de secuencia, puertos de origen y destino, etc, como se puede mostrar en las figuras anteriores donde se indican los campos de la cabecera tanto para TCP como para UDP.

Los campos para UDP son puerto de origen, puerto de destino, longitud de datagrama UDP y el checksum o suma de verificación (opcional), mientras que para TCP son puerto de origen y destino, secuencia del paquete, número de acuse de recibo entre otros como se indica en la siguiente figura.

**c) Identifique las capas del modelo TCP/IP y determine cómo intervienen en la comunicación.**

Las capas del modelo TCP/IP tienen tareas mucho más diversas que las del modelo OSI, considerando que ciertas capas del modelo TCP/IP corresponden a varios niveles del modelo OSI.

Las funciones de las capas son las siguientes:

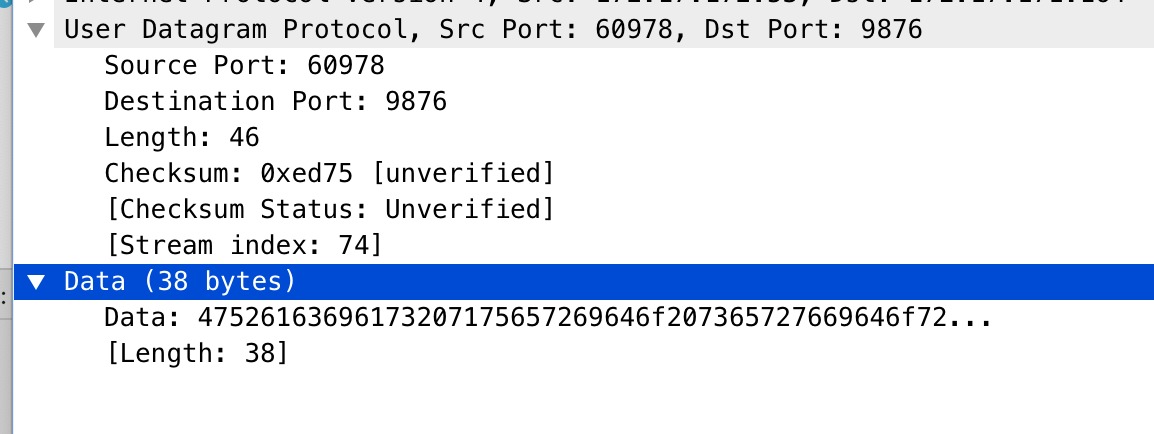
**Capa de Acceso a la Red:** Especifica cómo se enrutan los datos, sea cual sea el tipo de red utilizado.

**Capa de Internet:** Es responsable de proporcionar el paquete de datos (datagrama). (ARP)

**Capa de Transporte:** Brinda los datos de enrutamiento, junto con los mecanismos que permiten conocer el estado de la transmisión. Comprende a los protocolos TCP y UDP.

**Capa de aplicación:** Incorpora aplicaciones de red estándar (Telnet, SMTP, FTP, etc.)

En este caso la aplicación utilizada en la práctica interviene en la capa de transporte hasta la capa de aplicación, en la capa de transporte a través de los sockets se establece la utilización del protocolo TCP o UDP, también se establece un canal de comunicación para el envío y recepción de información y en la capa de Aplicación que corresponde a la interfaz con el usuario en nuestro caso la consola en la que recibimos y enviamos la información a través de mensajes

.

**ANEXOS**

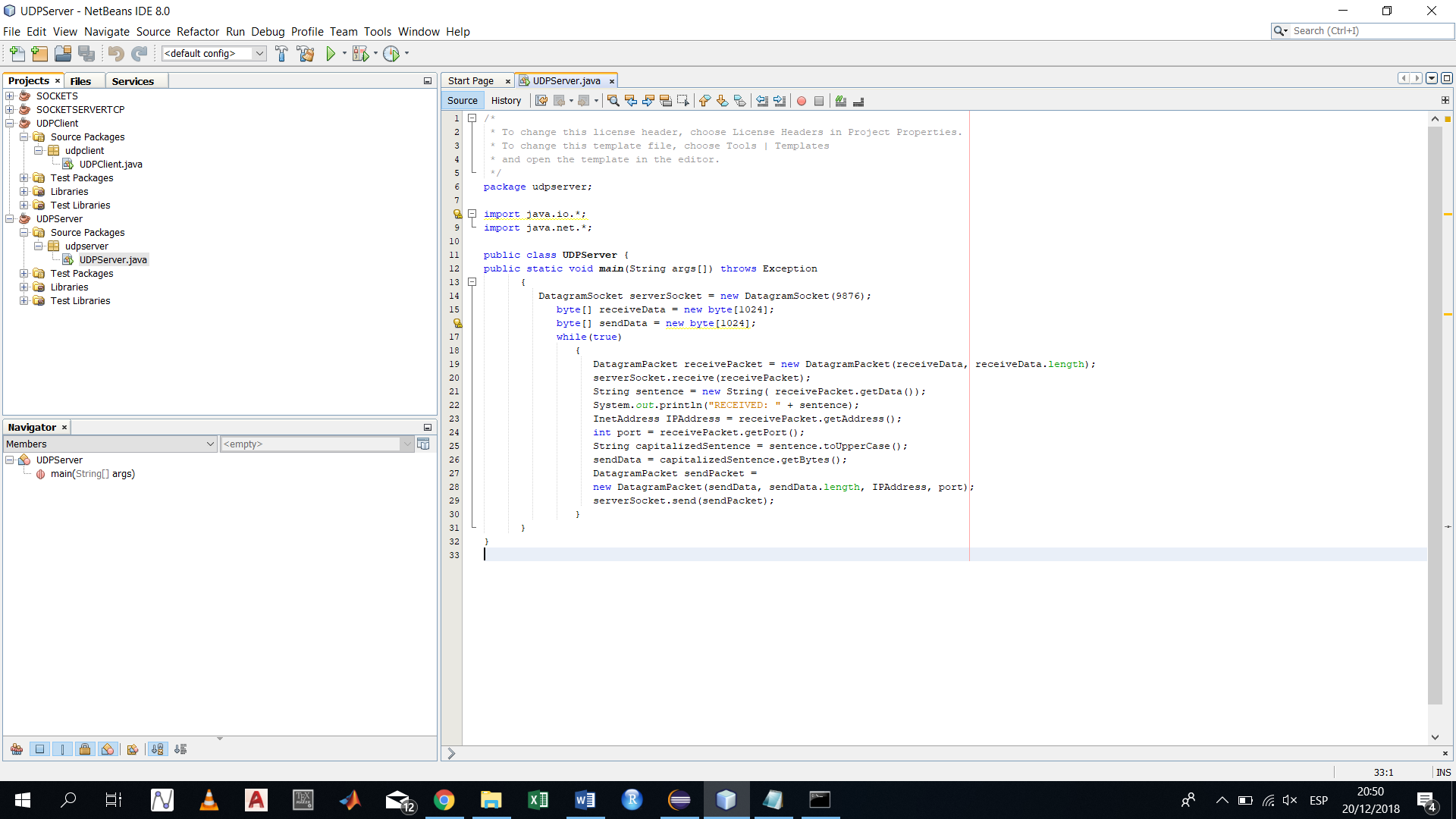


Fig.1. Código Sever UDP

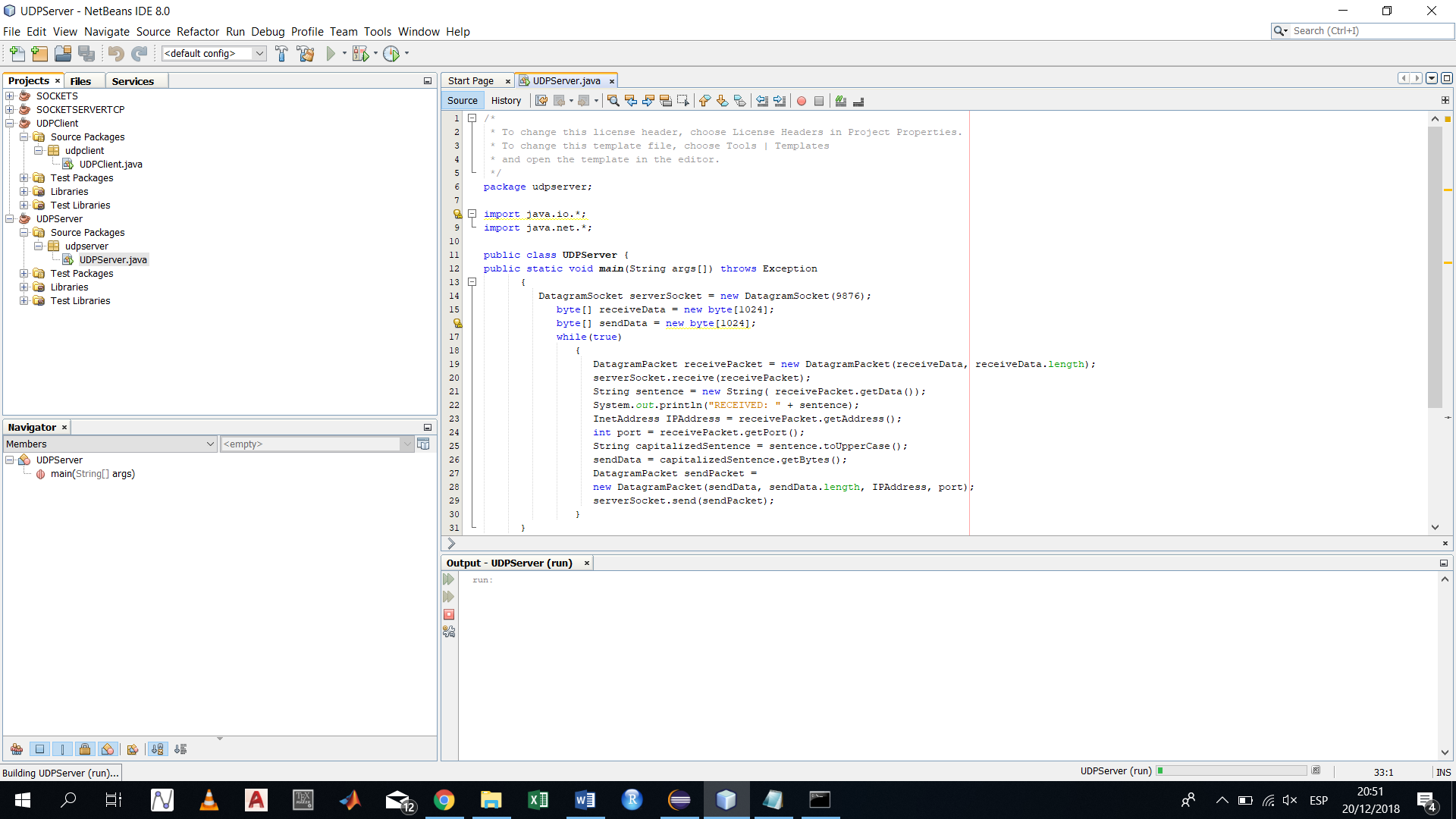


Fig.2. Ejecución código Sever UDP

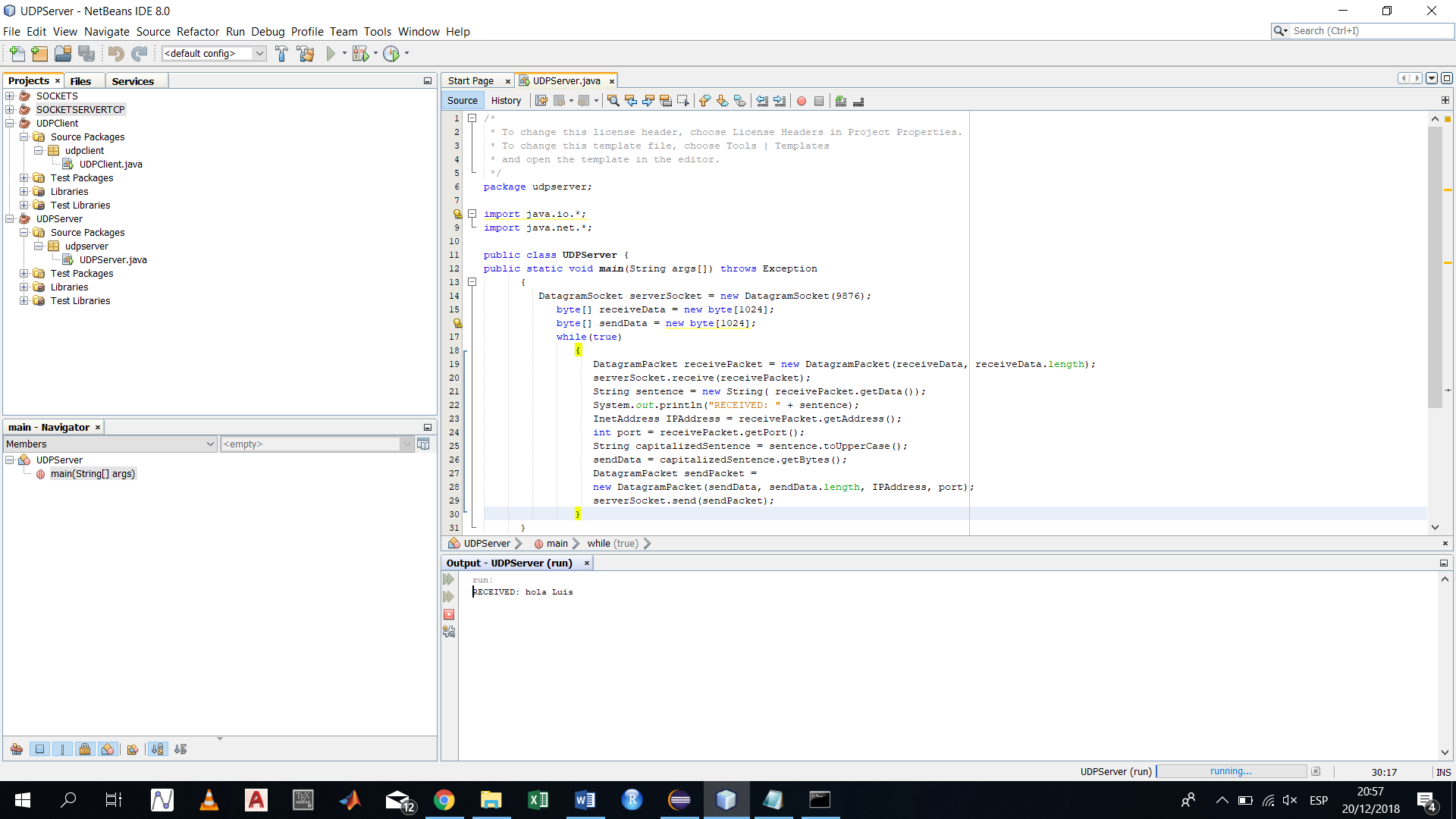


Fig.3. Recepción de información del Cliente

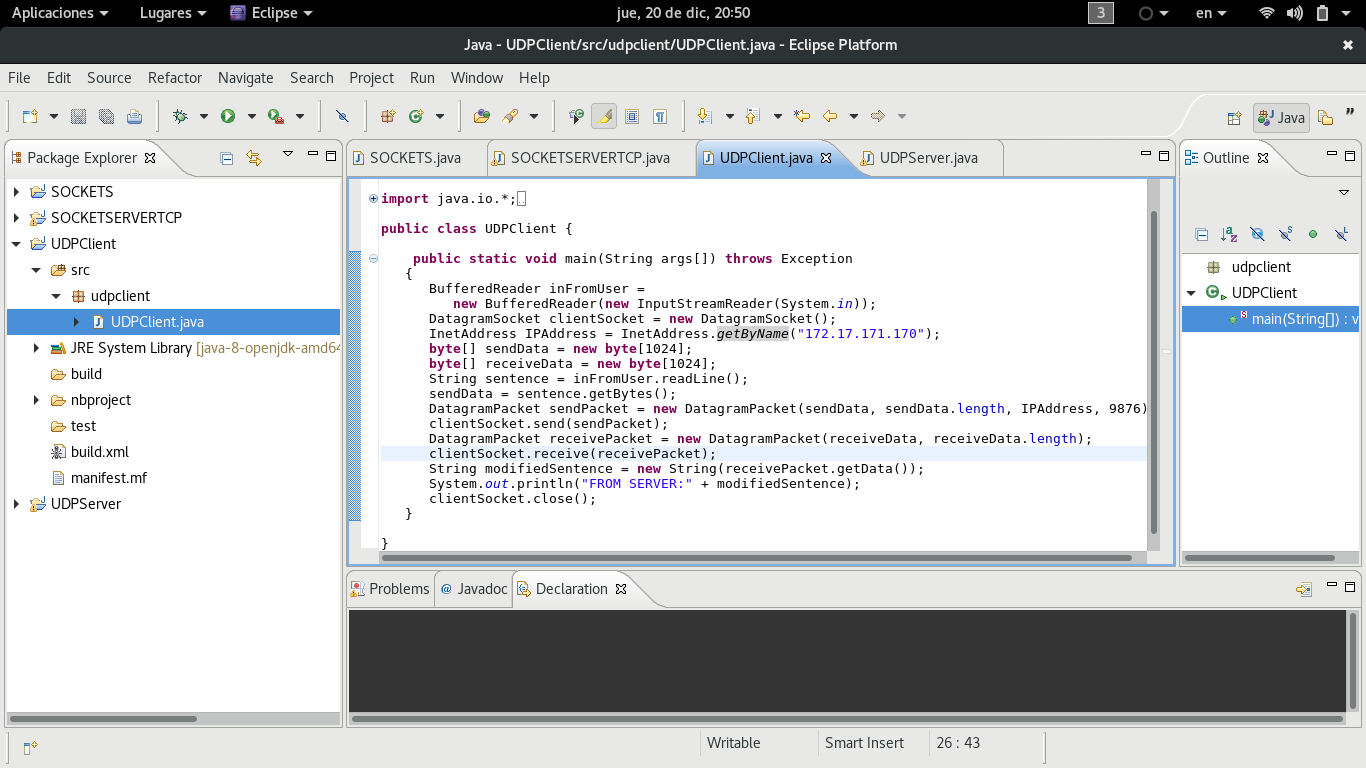


Fig.4. Código Cliente UDP

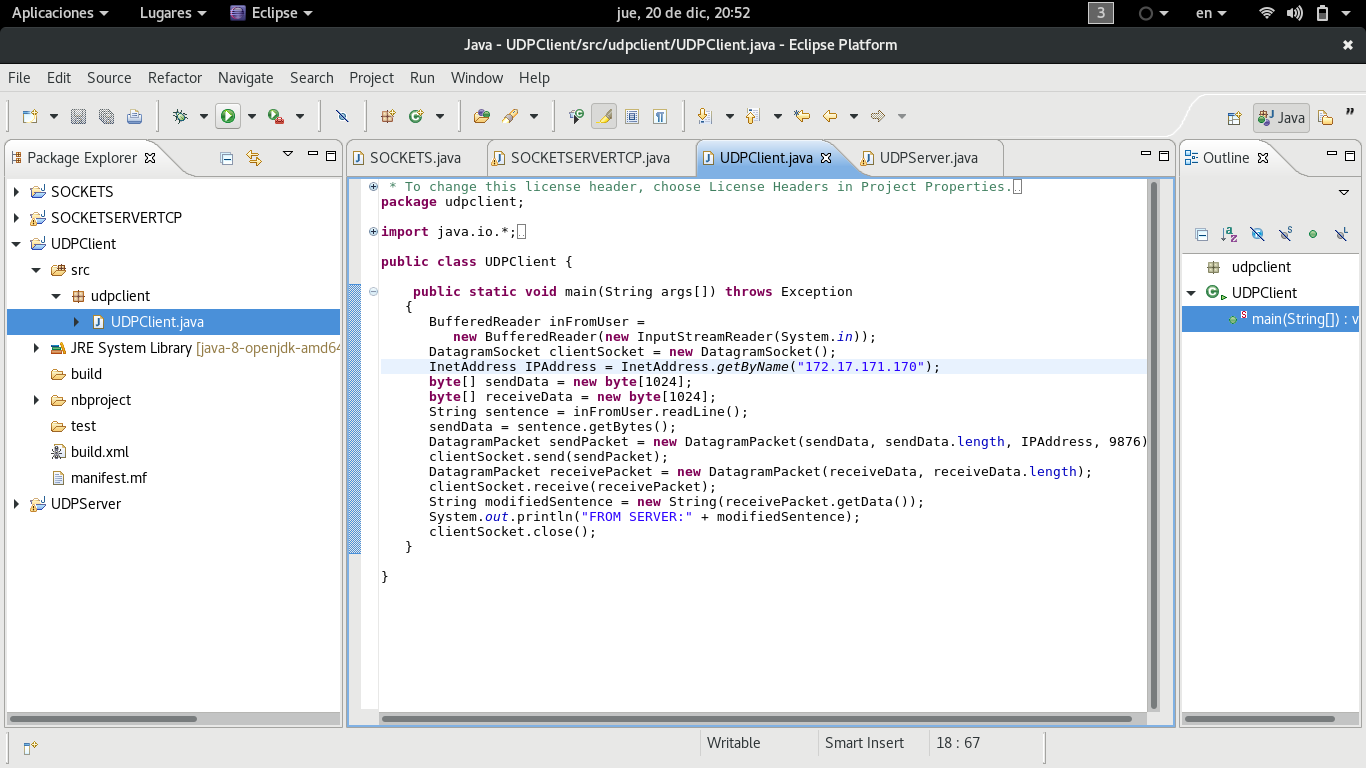


Fig.5. Ejecución código ClienteUDP

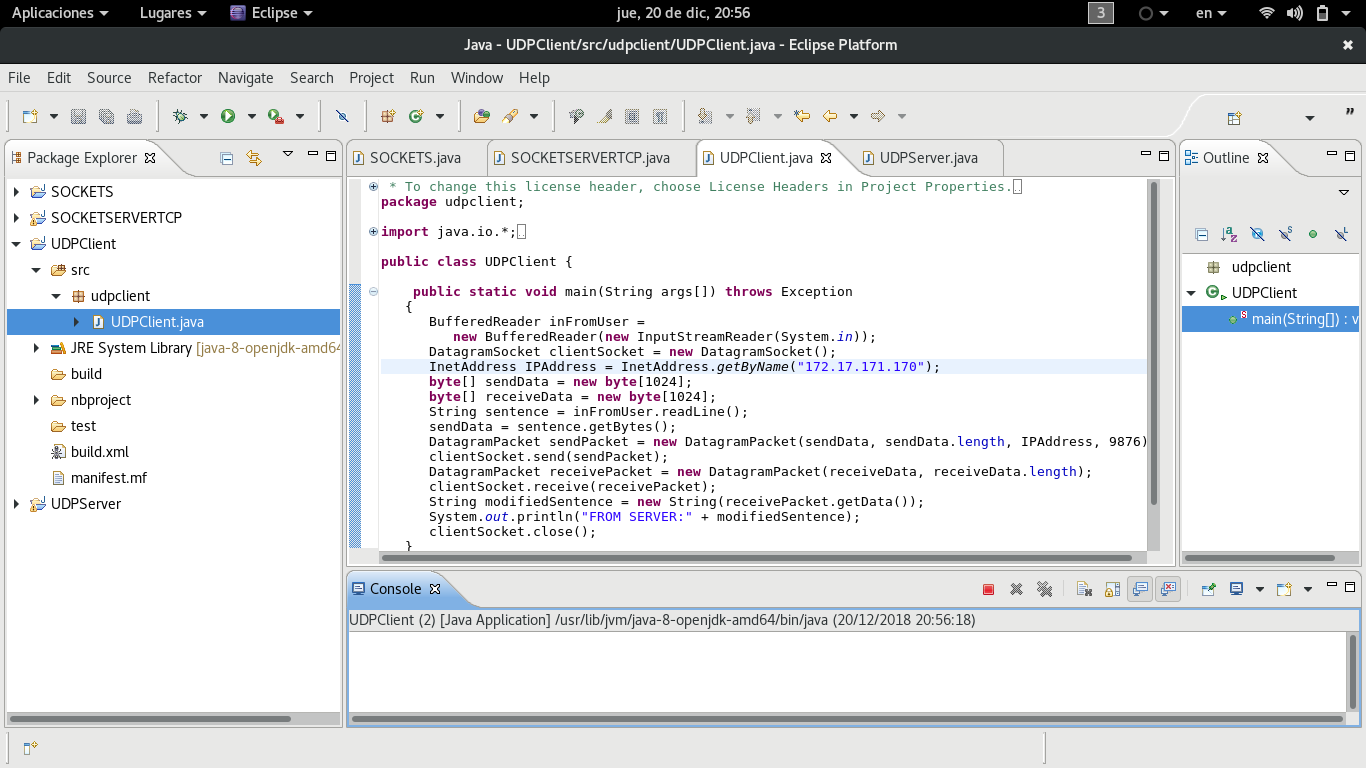


Fig.6. Ejecución código ClienteUDP

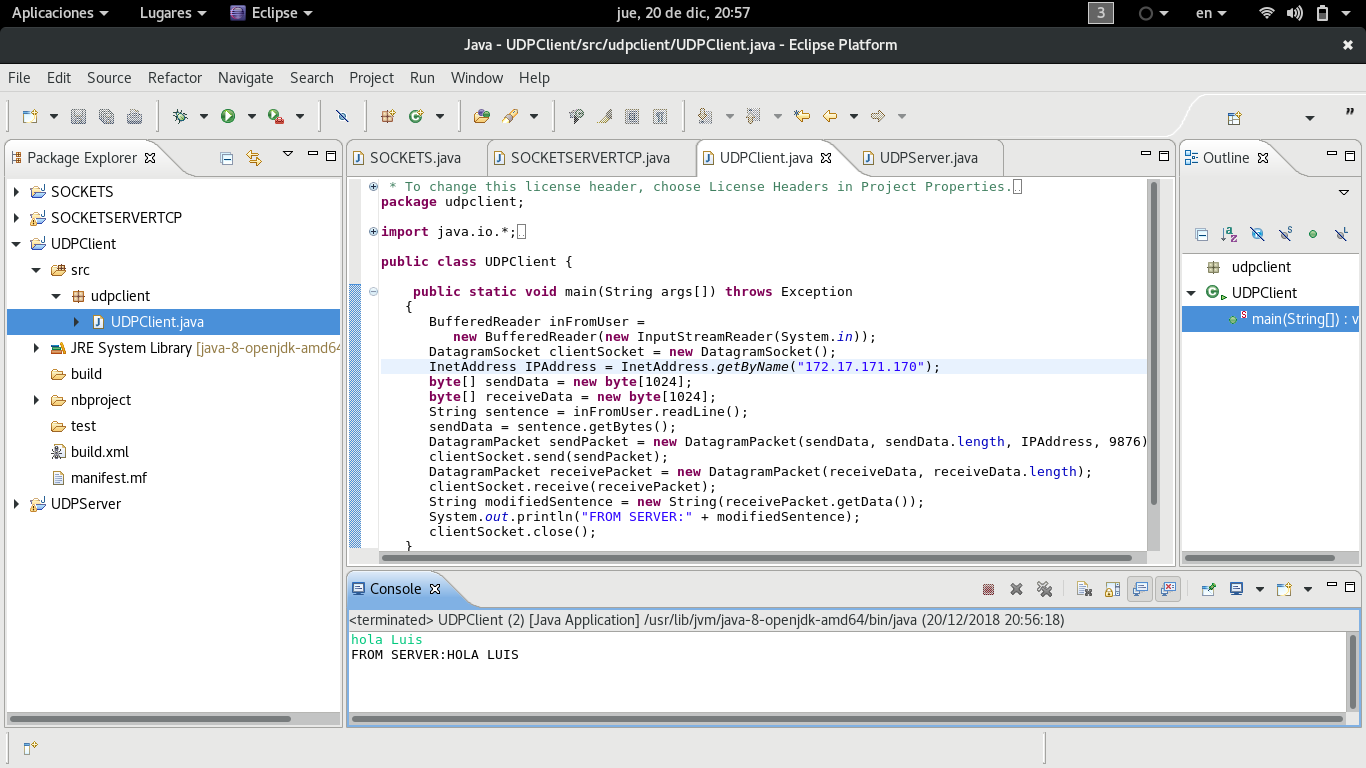


Fig.7. Envío de información de a Server UDP

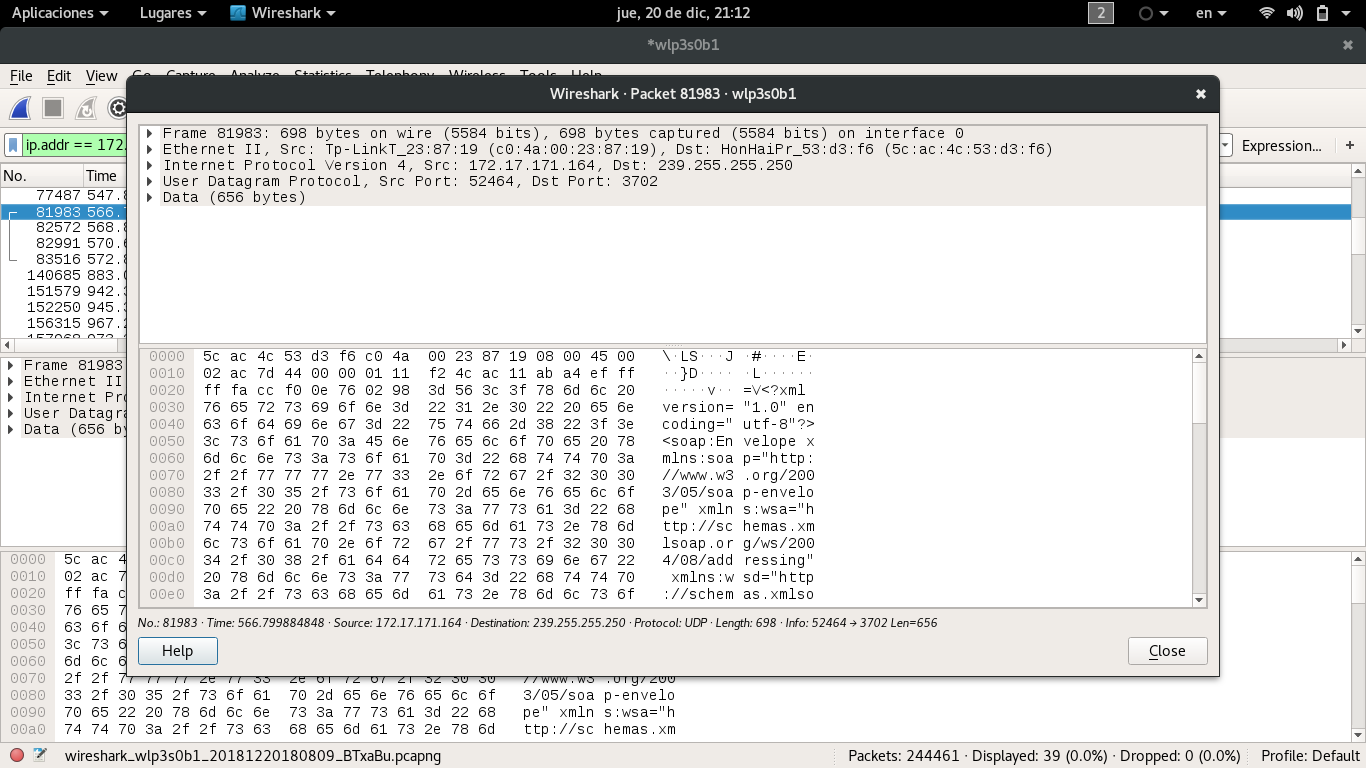


Fig.8. Wireshark

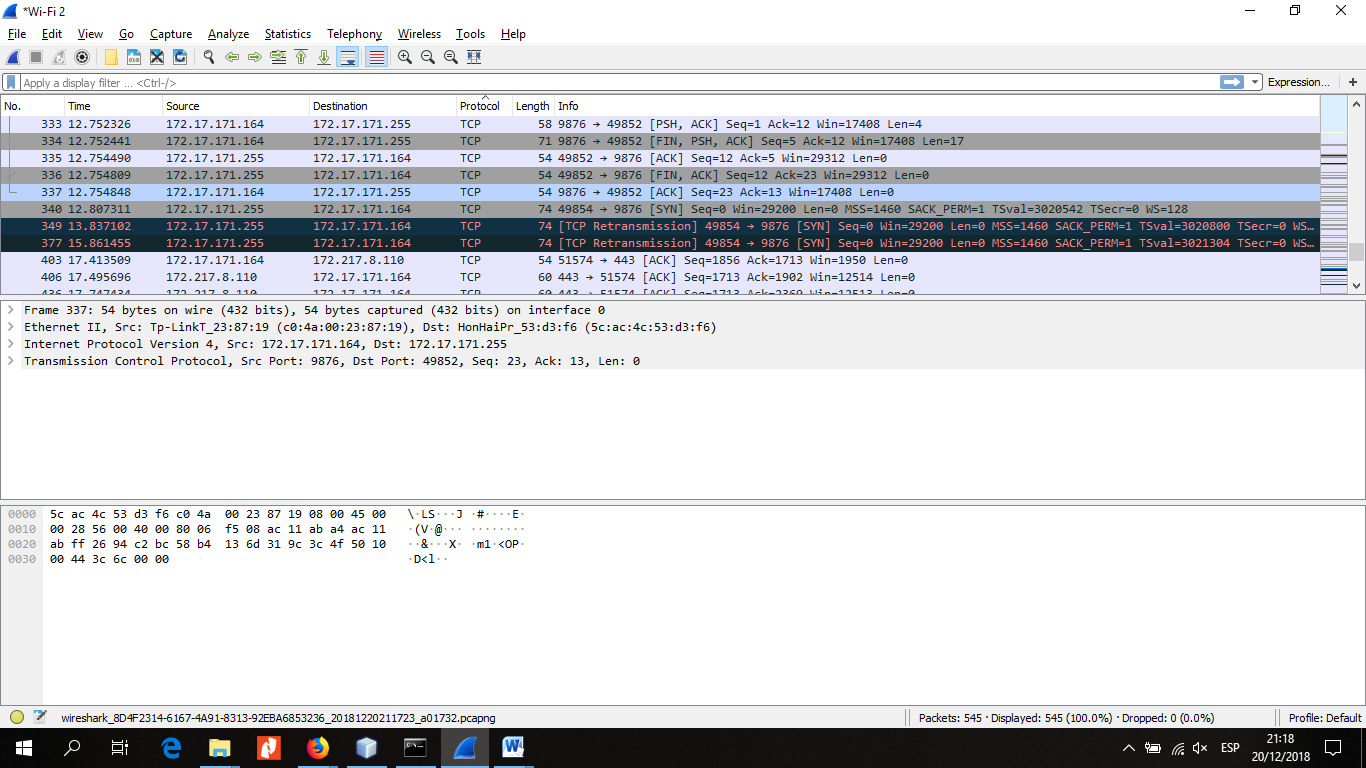


Fig.8. Wireshark