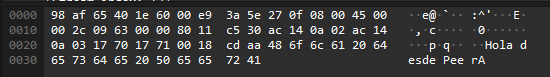
**Laboratorio 2**

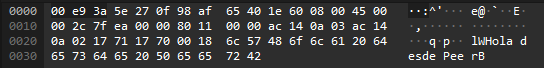
*Hecho por:*

*Jose Miguel Armas*

*Santiago Zapata*

1. Si, en la captura de wireshark es posible ver el contenido del mensaje enviado (y recibido).





1. Wireshark muestra el checksum como "unverified" porque muchas veces el sistema operativo usa una técnica llamada *checksum offloading*, donde la verificación del checksum se delega al hardware de red (tarjeta de red) y no se incluye en la captura. Esto es común cuando Wireshark corre en el equipo emisor o receptor, por lo que el paquete capturado puede no tener un checksum visible para el programa, aunque en realidad sí se haya enviado correctamente.
2. En el desarrollo de un programa basado en red como este, se pueden aplicar varios patrones de diseño y arquitectura para mejorar la organización del código y su mantenibilidad. En primer lugar, la arquitectura cliente-servidor permite una clara separación de responsabilidades entre los componentes que envían y reciben datos. El patrón Singleton es útil para manejar la instancia de la conexión de red, asegurando un acceso centralizado. El patrón Observer facilita la notificación automática a otros componentes cuando se reciben mensajes. Además, el patrón Strategy permite cambiar dinámicamente el protocolo o la lógica de comunicación sin alterar el resto del sistema. Finalmente, un patrón Facade puede abstraer las operaciones complejas de red detrás de una interfaz simple, como la clase UDPConnection, facilitando su reutilización y comprensión.
3. En la implementación propuesta ya se establece esto. Mientras el código se este ejecutando, estará abierto a recibir mensajes. Esto se da por la implementación de la estructura while(true) dentro del método run(), lo que permite que el socket siga esperando datagramas de forma indefinida sin finalizar tras recibir el primer mensaje.
4. Se modificó el código para que la lógica de transmisión de paquetes se ejecute en un hilo dedicado y persistente, separado del hilo principal. Para ello, se utilizó una cola (BlockingQueue) que almacena los mensajes a enviar. El nuevo hilo toma los mensajes de la cola y los transmite, permitiendo así una gestión más eficiente y ordenada de los envíos sin crear un nuevo hilo por cada mensaje.

package util;

import java.io.IOException;

import java.net.\*;

import java.util.concurrent.BlockingQueue;

import java.util.concurrent.LinkedBlockingQueue;

public class UDPConnection extends Thread {

private DatagramSocket socket;

private int port;

private BlockingQueue<Message> messageQueue;

private volatile boolean running;

public UDPConnection(int port) {

this.port = port;

this.messageQueue = new LinkedBlockingQueue<>();

this.running = true;

try {

this.socket = new DatagramSocket(port);

} catch (SocketException e) {

e.printStackTrace();

}

// Inicia el hilo de transmisión

Thread senderThread = new Thread(this::handleSending);

senderThread.start();

}

public void close() {

running = false;

socket.close();

}

@Override

public void run() {

try {

byte[] buffer = new byte[1024];

DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

System.out.println("Esperando mensajes en puerto " + port + "...");

while (running) {

socket.receive(packet);

String msg = new String(packet.getData(), 0, packet.getLength());

System.out.println("Mensaje recibido de " + packet.getAddress() + ": " + msg);

}

} catch (IOException e) {

if (running) e.printStackTrace(); // Evita mostrar errores al cerrar intencionalmente

}

}

public void sendDatagram(String msg, String ipDest, int portDest) {

messageQueue.offer(new Message(msg, ipDest, portDest));

}

private void handleSending() {

while (running) {

try {

Message m = messageQueue.take();

InetAddress ip = InetAddress.getByName(m.ipDest);

byte[] data = m.msg.getBytes();

DatagramPacket packet = new DatagramPacket(data, data.length, ip, m.portDest);

socket.send(packet);

System.out.println("Mensaje enviado a " + m.ipDest + ":" + m.portDest);

} catch (InterruptedException | IOException e) {

if (running) e.printStackTrace();

}

}

}

private static class Message {

String msg;

String ipDest;

int portDest;

Message(String msg, String ipDest, int portDest) {

this.msg = msg;

this.ipDest = ipDest;

this.portDest = portDest;

}

}

}

1. Para poder cambiar la estructura del código y que funcione con TCP en lugar de UDP se deben hacer los siguientes cambios:

* En vez de usar DatagramSocket, se debe utilizar un ServerSocket para escuchar conexiones entrantes y un Socket para enviar y recibir datos.
* Se debe cambiar el método run() para poder aceptar conexiones TCP. Ya que en USP solo se escucha los mensajes sin conexión, pero en TCP se debe aceptar conexiones primero. Cuando un cliente se conecta, se obtiene un Socket para la comunicación.
* Se modifica el método sendDatagram() a sendMessage() para usar TCP. En lugar de un datagrama, se debe abrir un Socket al destino y enviar el mensaje por un OutputStream.
* En PeerA y PeerB se asegura que el método de envío ha cambiado de sendDatagram a sendMessage.
* Por ultimo se debe cerrar los streams y los sockets para liberar los recursos.

1. Para evitar conexiones no deseadas se usaría una lista de IPs permitidas (un filtro de IPs) . Es decir, cuando un cliente envía un datagrama al servidor, este lo recibe sin verificar previamente si el cliente está autorizado. El filtrado de IPs en UDP ocurre después de recibir el datagrama, no antes, porque el servidor solo conoce la IP del emisor al momento de recibir el paquete.

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.SocketException;

import java.util.Set;

public class UDPConnection extends Thread {

private DatagramSocket socket;

private int port;

// Lista blanca de IPs permitidas

private static final Set<String> ALLOWED\_IPS = Set.of("172.20.10.3", "172.30.188.93");

public UDPConnection(int port) {

this.port = port;

try {

this.socket = new DatagramSocket(port);

} catch (SocketException e) {

e.printStackTrace();

}

}

public void close() {

socket.close();

}

@Override

public void run() {

try {

byte[] buffer = new byte[1024];

DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

System.out.println("Esperando mensajes en puerto " + port + "...");

while (true) {

socket.receive(packet); // Se recibe un datagrama

InetAddress senderAddress = packet.getAddress();

String senderIP = senderAddress.getHostAddress();

// Se hace la verificación de IP en la whitelist

if (ALLOWED\_IPS.contains(senderIP)) {

String msg = new String(packet.getData(), 0, packet.getLength());

System.out.println("Mensaje recibido de " + senderIP + ": " + msg);

} else {

System.out.println("Paquete descartado de IP no autorizada: " + senderIP);

}

}

} catch (IOException e) {

e.printStackTrace();

}

}

public void sendDatagram(String msg, String ipDest, int portDest) {

new Thread(() -> {

try {

InetAddress ip = InetAddress.getByName(ipDest);

byte[] data = msg.getBytes();

DatagramPacket packet = new DatagramPacket(data, data.length, ip, portDest);

socket.send(packet);

System.out.println("Mensaje enviado a " + ipDest + ":" + portDest);

} catch (IOException e) {

e.printStackTrace();

}

}).start();

}

}