William Kenji Kiplinger

<https://github.com/Kenjum/CS380-P5>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_UdpClient.java\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

import java.io.InputStream;

import java.io.OutputStream;

import java.net.Socket;

import java.util.Random;

import java.util.concurrent.TimeUnit;

import javax.xml.bind.DatatypeConverter;

public class UdpClient {

public static void main(String[] args) {

try (Socket socket = new Socket("18.221.102.182", 38005)) {

System.out.println("Connection Established");

InputStream is = socket.getInputStream();

OutputStream os = socket.getOutputStream();

Random rd = new Random();

// Header for ipv4

byte[] ipv4 = new byte[20];

// [0100][0101]...

// Version, Internet Header Length size 20 bytes

ipv4[0] = 69;

ipv4[1] = 0;

ipv4[2] = 0;

ipv4[3] = 24;

ipv4[4] = 0;

ipv4[5] = 0;

ipv4[6] = 64;

ipv4[7] = 0;

ipv4[8] = 50;

ipv4[9] = 17;

ipv4[10] = 0;

ipv4[11] = 0;

ipv4[12] = 127;

ipv4[13] = 0;

ipv4[14] = 0;

ipv4[15] = 1;

ipv4[16] = 18;

ipv4[17] = (byte) 221;

ipv4[18] = 102;

ipv4[19] = (byte) 182;

// End of header

// checksum applied

short check = checksum(ipv4);

ipv4[11] = (byte) (check & 0xFF);

ipv4[10] = (byte) ((check >> 8) & 0xFF);

// Preparing the "handshake" packet

String handShakeString = "DEADBEEF";

byte[] handShakeArray = DatatypeConverter.parseHexBinary(handShakeString);

// Copy the header and then add the "handshake" packet

byte[] handShake = new byte[24];

for (int i = 0; i < 20; i++) {

handShake[i] = ipv4[i];

}

handShake[20] = handShakeArray[0];

handShake[21] = handShakeArray[1];

handShake[22] = handShakeArray[2];

handShake[23] = handShakeArray[3];

// Send the "handshake"

os.write(handShake);

// Read the reply from server

byte[] serverReply = new byte[4];

serverReply[0] = (byte) is.read();

serverReply[1] = (byte) is.read();

serverReply[2] = (byte) is.read();

serverReply[3] = (byte) is.read();

// If the handshake was successful, we will be given two additional

// bytes after the 4 bytes of reply.

int[] port = new int[2];

port[0] = is.read();

port[1] = is.read();

String serverReplyString = DatatypeConverter.printHexBinary(serverReply);

System.out.println("Handshake response: 0x" + serverReplyString);

System.out.println("Port number received: " + port[0] + port[1]);

// UDP header

byte[] udp = new byte[8];

udp[0] = 0;

udp[1] = 0;

udp[2] = (byte) port[0];

udp[3] = (byte) port[1];

// First half of header

int byteSize = 1;

int averageRTT = 0;

// This will send packets 12 times. Each time, the byte size will

// multiply by 2.

for (int i = 1; i <= 12; i++) {

byteSize \*= 2;

// Total length

ipv4[2] = (byte) (((ipv4.length + byteSize + udp.length) >> 8) & 0xFF);

ipv4[3] = (byte) ((ipv4.length + byteSize + udp.length) & 0xFF);

// Reseting header checksum

ipv4[10] = 0;

ipv4[11] = 0;

// Header checksum

check = checksum(ipv4);

ipv4[10] = (byte) ((check >> 8) & 0xFF);

ipv4[11] = (byte) (check & 0xFF);

// Two more bytes for the UDP header

// UDP length

udp[4] = (byte) (((byteSize + udp.length) >> 8) & 0xFF);

udp[5] = (byte) ((byteSize + udp.length) & 0xFF);

// Provides random bytes of data, as requested on assignment.

byte[] udpPayload = new byte[byteSize];

rd.nextBytes(udpPayload);

byte[] pseudoHeader = new byte[20 + udpPayload.length];

pseudoHeader[0] = 127; // Source IPv4 address

pseudoHeader[1] = 0;

pseudoHeader[2] = 0;

pseudoHeader[3] = 1;

pseudoHeader[4] = 18; // Destination IPv4 address 18.221.102.182

pseudoHeader[5] = (byte) 221;

pseudoHeader[6] = 102;

pseudoHeader[7] = (byte) 182;

pseudoHeader[8] = 0; // Zeroes

pseudoHeader[9] = 17; // Protocol UDP

// UDP Length

pseudoHeader[10] = (byte) (((udp.length + udpPayload.length) >> 8) & 0xFF);

pseudoHeader[11] = (byte) ((udp.length + udpPayload.length) & 0xFF);

pseudoHeader[12] = 0; // Source Port

pseudoHeader[13] = 0;

pseudoHeader[14] = udp[2]; // Destination Port

pseudoHeader[15] = udp[3];

// Length

pseudoHeader[16] = (byte) (((udp.length + byteSize) >> 8) & 0xFF);

pseudoHeader[17] = (byte) ((udp.length + byteSize) & 0xFF);

// Checksum

pseudoHeader[18] = 0;

pseudoHeader[19] = 0;

// Beyond is data

// Start filling the pseudoHeader at 20

for (int j = 20; j < pseudoHeader.length; j++) {

pseudoHeader[j] = udpPayload[j - 20];

}

// Last 2 bytes of UDP header

check = checksum(pseudoHeader);

udp[6] = (byte) ((check >> 8) & 0xFF);

udp[7] = (byte) (check & 0xFF);

// End of UDP header

byte[] packet = new byte[udp.length + udpPayload.length];

// Filling packet with udp header

int index = 0;

for (int j = 0; j < udp.length; j++) {

packet[index] = udp[j];

index++;

}

// Filling packet with udp payload

for (int j = 0; j < udpPayload.length; j++) {

packet[index] = udpPayload[j];

index++;

}

// Combining IPv4 and packet

byte[] finalSend = new byte[ipv4.length + packet.length];

// Filling final package with IPv4 information

index = 0;

for (int j = 0; j < ipv4.length; j++) {

finalSend[index] = ipv4[j];

index++;

}

// Filling final package with packet information

for (int j = 0; j < packet.length; j++) {

finalSend[index] = packet[j];

index++;

}

// Send to server

os.write(finalSend);

// Start timer

long start = System.nanoTime();

// Receive server response

serverReply[0] = (byte) is.read();

serverReply[1] = (byte) is.read();

serverReply[2] = (byte) is.read();

serverReply[3] = (byte) is.read();

String serverReplyString2 = DatatypeConverter.printHexBinary(serverReply);

// End timer

long end = System.nanoTime();

// Find difference between start and end for duration.

long timeElapsed = TimeUnit.NANOSECONDS.toMillis(end - start);

// Display information

System.out.println("\nSending packet with " + byteSize + " bytes of data");

System.out.println("Response: 0x" + serverReplyString2);

System.out.println("RTT: " + timeElapsed + "ms");

averageRTT += timeElapsed;

}

// Display average rtt

System.out.println("\nAverage RTT: " + (averageRTT / 12) + "ms");

} catch (Exception e) {

e.printStackTrace();

}

}

// Similar IPv4 checksum as Project 3

public static short checksum(byte[] b) {

long sum = 0;

int count = b.length;

for (int i = 0; count > 0; --count) {

sum += (b[i++] & 0xFF) << 8;

if ((--count) == 0) {

break;

}

sum += (b[i++] & 0xff);

}

return (short) ((~((sum & 0xFFFF) + (sum >> 16))) & 0xFFFF);

}

}