**Εργασία Δίκτυα Υπολογιστών ΙΙ –**

**Source Code**



Ονοματεπώνυμο: Μιχαήλ Καρατζάς

ΑΕΜ: 9137

email: [mikalaki@ece.auth.gr](mailto:mikalaki@ece.auth.gr)

Εξάμηνο : 9ο

Δεδομένα και κώδικες της εργασίας [εδώ](https://www.dropbox.com/sh/qyausy4cjhnd5z7/AABS6NhRMhpr8YUmXQ4En1Aja?dl=0)

Στο παρόν αρχείο υπάρχει ο κώδικας Java της εργασίας σε μορφή κειμένου:

/\*

\*

\* Computer Networks 2

\*

\* Experimental Virtual Lab

\*

\* Java network Socket Programming

\*

\* Author :Michael Karatzas

\* AEM:9137

\*

\* November 2020

\*/

import java.net.\*;

import java.io.\*;

import java.nio.ByteBuffer;

import java.nio.ByteOrder;

import java.text.SimpleDateFormat;

import java.util.\*;

import javax.sound.sampled.\*;

import java.util.Arrays;

public class userApplication{

//Connection parameters

private static String client\_public\_address ="87.202.99.120";

private static byte[] hostIP = { (byte)155,(byte)207,(byte)18,(byte)208 };

private static int client\_listening\_port= 48021 ;

private static int server\_listening\_port= 38021 ;

//Request codes

private static String echo\_request\_code="E2685";

private static String echo\_request\_code\_no\_delay="E0000";

private static String image\_request\_code="M7758";

private static String audio\_request\_code="A4860";

private static String ithakicopter\_code="Q8861\r";

private static String vehicle\_obd\_II\_code="V7889\r";

//Programm parameters

private static int N\_OF\_SOUND\_PACKETS = 900;

//Filestreams declaration for files, where programm's data is going to be stored.

private static FileOutputStream console\_full;

private static FileOutputStream echoPacketsTimes\_delay;

private static FileOutputStream echoPacketsTimes\_NoDelay;

private static FileOutputStream image\_cam;

private static FileOutputStream temperatures;

private static FileOutputStream sound\_samples\_diff;

private static FileOutputStream sound\_samples;

private static FileOutputStream sound\_mean\_value;

private static FileOutputStream sound\_steps;

private static FileOutputStream telemetry;

private static FileOutputStream OBD\_engireRunTime;

private static FileOutputStream OBD\_IntakeAirTemperature;

private static FileOutputStream OBD\_ThrottlePosition;

private static FileOutputStream OBD\_EngineRPM;

private static FileOutputStream OBD\_VehicleSpeed;

private static FileOutputStream OBD\_CoolantTemperature;

public static void main(String[] param) throws SocketException,IOException,LineUnavailableException {

//In console\_full.txt File we print all the readable output of our program execution.

console\_full=new FileOutputStream("console\_full.txt");

System.out.println("!!!!!!!!! JAVA SOCKET PROGRAM HAS STARTED !!!!!!!!!");

console\_full.write("!!!!!!!!! JAVA SOCKET PROGRAM HAS STARTED !!!!!!!!!\n".getBytes());

//Get echo packets Response Times for more than 4 minutes - with delay

echoPacketsTimes(echo\_request\_code,echoPacketsTimes\_delay, "echoPacketsResTimes\_delay");

//Get echo packets Response Times for more than 4 minutes - without delay

echoPacketsTimes(echo\_request\_code\_no\_delay,echoPacketsTimes\_NoDelay, "echoPacketsResTimes\_NoDelay");

getSomeEchoPackets(echo\_request\_code,echoPacketsTimes\_delay, "echoPacketsResTimes\_delay");

//get the first image

getImage( image\_cam, "img\_CAM1", "CAM=FIX");

getSomeEchoPackets(echo\_request\_code,echoPacketsTimes\_delay, "echoPacketsResTimes\_delay");

//get the second image

getImage( image\_cam, "img\_CAM2", "CAM=PTZ");

getSomeEchoPackets(echo\_request\_code,echoPacketsTimes\_delay, "echoPacketsResTimes\_delay");

//get the temperatures request's response in the temperatures file

getTemperatures(echo\_request\_code,temperatures, "temperatures");

getSomeEchoPackets(echo\_request\_code,echoPacketsTimes\_delay, "echoPacketsResTimes\_delay");

//get frequency generator samples from simple DPCM

getSoundfromSimpleDPCM("DPCM\_freq\_samples\_diff","DPCM\_freq\_actual\_samples","T" + String.valueOf(N\_OF\_SOUND\_PACKETS));

getSomeEchoPackets(echo\_request\_code,echoPacketsTimes\_delay, "echoPacketsResTimes\_delay");

//get song samples from simple DPCM

getSoundfromSimpleDPCM("DPCM\_song\_samples\_diff","DPCM\_song\_actual\_samples","F" + String.valueOf(N\_OF\_SOUND\_PACKETS));

getSomeEchoPackets(echo\_request\_code,echoPacketsTimes\_delay, "echoPacketsResTimes\_delay");

//get song samples from AQDPCM --- song1

getSoundfromAQDPCM("AQDPCM\_song1\_samples\_diff","AQDPCM\_song1\_actual\_samples","AQDPCM\_song1\_meanValues",

"AQDPCM\_song1\_steps","F" +"AQ" + String.valueOf(N\_OF\_SOUND\_PACKETS));

getSomeEchoPackets(echo\_request\_code,echoPacketsTimes\_delay, "echoPacketsResTimes\_delay");

//get song samples from AQDPCM --- song2

getSoundfromAQDPCM("AQDPCM\_song2\_samples\_diff","AQDPCM\_song2\_actual\_samples","AQDPCM\_song2\_meanValues",

"AQDPCM\_song2\_steps","F" +"AQ" + String.valueOf(N\_OF\_SOUND\_PACKETS));

getSomeEchoPackets(echo\_request\_code,echoPacketsTimes\_delay, "echoPacketsResTimes\_delay");

//Ithakicopters telemetry 1

IthakiCopter("Telemetry\_LLL\_RRR\_AAA\_TTTT\_PPPPPP\_1");

getSomeEchoPackets(echo\_request\_code,echoPacketsTimes\_delay, "echoPacketsResTimes\_delay");

//Ithakicopters telemetry 2

IthakiCopter("Telemetry\_LLL\_RRR\_AAA\_TTTT\_PPPPPP\_2");

getSomeEchoPackets(echo\_request\_code,echoPacketsTimes\_delay, "echoPacketsResTimes\_delay");

//On Board Diagnostics

OBD();

//closing filestream to console\_full.txt

console\_full.close();

}

//Function to get the times of echopackets

private static void echoPacketsTimes( String code, FileOutputStream resTimes\_file, String responseTimesFilename) throws IOException {

//Print message to indicate the beginning of echo packets application

SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

Date date = new Date();

System.out.print("Echo Packets START ");

System.out.println(formatter.format(date));

System.out.println("Echo Packets:");

//In console\_full.txt is written all the readable output

console\_full.write("Echo Packets START ".getBytes());

console\_full.write(formatter.format(date).getBytes());

console\_full.write("\nEcho Packets:\n".getBytes());

//Variable to store the start timestamp of the 4+ minutes echo packets exchange

long startTime=0;

//Variable to store the start timestamp of an echo serverResponse packet

long packetStartTime=0;

//Variable to store the end timestamp of an echo serverResponse packet

long packetEndTime=0;

//opening a stream to the file

resTimes\_file=new FileOutputStream( responseTimesFilename + ".csv");

//Datagram Socket for sending request packets initialization.

DatagramSocket s = new DatagramSocket();

byte[] txbuffer = (code).getBytes();

InetAddress hostAddress = InetAddress.getByAddress(hostIP);

DatagramPacket clientRequest = new DatagramPacket(txbuffer,txbuffer.length,

hostAddress,server\_listening\_port);

//Datagram Socket for receiving response packets initialization.

DatagramSocket r = new DatagramSocket(client\_listening\_port);

r.setSoTimeout(8000);

byte[] rxbuffer = new byte[2048];

DatagramPacket serverResponse = new DatagramPacket(rxbuffer,rxbuffer.length);

startTime=System.currentTimeMillis();

// We want our echo packets interchange to long for 4 minutes at least - 4 minutes is equal to 240000 milliseconds

// So by executing our echo packets interchange for 255000 milliseconds, it lasts for 4 minutes and 15 seconds.

while (packetEndTime-startTime< 255000) {

s.send(clientRequest);

//The timestamp when we sent the clientRequest packet, is the start of the server response time.

packetStartTime=System.currentTimeMillis();

try {

r.receive(serverResponse);

//The timestamp when we get the serverResponse packet, is the end of the server response time.

packetEndTime=System.currentTimeMillis();

String message = new String(rxbuffer,0,serverResponse.getLength());

//Print the size of a package.

//System.out.println("The size of a package = " + serverResponse.getLength() + "bytes");

resTimes\_file.write(( (packetEndTime-packetStartTime ) +"\n" ).getBytes());

System.out.println(message);

console\_full.write((message+"\n").getBytes());

} catch (Exception x) {

System.out.println(x);

}

}

//closing the UDP sockets opened before

s.close();

r.close();

resTimes\_file.close();

//Print message to indicate the end of echo packets application

date = new Date();

System.out.print("Echo Packets END ");

console\_full.write("Echo Packets END ".getBytes());

System.out.print(formatter.format(date));

console\_full.write(formatter.format(date).getBytes());

System.out.print(" \n \n \n");

console\_full.write(" \n \n \n".getBytes());

}

//Function to get 5 echopackets before every other request -- For the wireshark screenshots.

private static void getSomeEchoPackets( String code, FileOutputStream resTimes\_file, String responseTimesFilename) throws IOException {

//Print message to indicate the beginning of echo packets small application

SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

Date date = new Date();

System.out.print("5 Echo Packets START ");

System.out.println(formatter.format(date));

System.out.println("Echo Packets:");

//In console\_full.txt is written all the readable output

console\_full.write("5 Echo Packets START ".getBytes());

console\_full.write(formatter.format(date).getBytes());

console\_full.write("\nEcho Packets:\n".getBytes());

//Datagram Socket for sending request packets initialization.

DatagramSocket s = new DatagramSocket();

byte[] txbuffer = (code).getBytes();

InetAddress hostAddress = InetAddress.getByAddress(hostIP);

DatagramPacket clientRequest = new DatagramPacket(txbuffer,txbuffer.length,

hostAddress,server\_listening\_port);

//Datagram Socket for receiving response packets initialization.

DatagramSocket r = new DatagramSocket(client\_listening\_port);

r.setSoTimeout(8000);

byte[] rxbuffer = new byte[2048];

DatagramPacket serverResponse = new DatagramPacket(rxbuffer,rxbuffer.length);

// We want to get some (5) echo packets before other requests.

for(int i=0; i<6 ;i++) {

s.send(clientRequest);

try {

r.receive(serverResponse);

String message = new String(rxbuffer,0,serverResponse.getLength());

System.out.println(message);

console\_full.write((message+"\n").getBytes());

} catch (Exception x) {

System.out.println(x);

}

}

//closing the UDP sockets opened before

s.close();

r.close();

//Print message to indicate the end of echo packets small application

date = new Date();

System.out.print("5 Echo Packets END ");

console\_full.write("5 Echo Packets END ".getBytes());

System.out.print(formatter.format(date));

console\_full.write(formatter.format(date).getBytes());

System.out.print(" \n \n");

console\_full.write(" \n \n".getBytes());

}

//Function for getting the image files.

private static void getImage( FileOutputStream img\_file, String imgFilename,String Image\_request\_params) throws IOException {

//Print message to indicate the beginning of image application.

SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

Date date = new Date();

System.out.print("Image Download START ");

System.out.println(formatter.format(date));

System.out.print("\n");

//In console\_full.txt is written all the readable output

console\_full.write("Image Download START ".getBytes());

console\_full.write(formatter.format(date).getBytes());

console\_full.write("\n".getBytes());

//opening a stream to the file

img\_file=new FileOutputStream( imgFilename + ".jpg");

//Datagram Socket for sending request packets initialization.

DatagramSocket s = new DatagramSocket();

//Creating the request string from the function input arguments.

String request\_string = image\_request\_code + Image\_request\_params ;

byte[] txbuffer = request\_string.getBytes();

InetAddress hostAddress = InetAddress.getByAddress(hostIP);

DatagramPacket clientRequest = new DatagramPacket(txbuffer,txbuffer.length,

hostAddress,server\_listening\_port);

//Datagram Socket for receving response packets initialization.

DatagramSocket r = new DatagramSocket(client\_listening\_port);

r.setSoTimeout(8000);

//Set the size to 1024, because this is the biggest possible packet length (L) the servers sends.

byte[] rxbuffer = new byte[1024];

DatagramPacket serverResponse = new DatagramPacket(rxbuffer,rxbuffer.length);

s.send(clientRequest);

//The 2 final bytes of a datagram received from server indicates if we are at the end of our image (image end delimeter)

byte[] datagramFinalTwoBytes = new byte[2];

do {

try {

r.receive(serverResponse);

byte[] datagramBytes = Arrays.copyOfRange(rxbuffer,0,serverResponse.getLength());

img\_file.write(datagramBytes);

//getting the packet's last 2 Bytes

datagramFinalTwoBytes[0]=datagramBytes[datagramBytes.length-2];

datagramFinalTwoBytes[1]=datagramBytes[datagramBytes.length-1];

} catch (Exception x) {

System.out.println(x);

}

//

}while (! (datagramFinalTwoBytes[0] == (byte)(0xFF) && datagramFinalTwoBytes[1]== (byte)(0xD9) ) );

//closing the UDP sockets opened before

s.close();

r.close();

img\_file.close();

//Print message to indicate the end of image application

formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

date = new Date();

System.out.print("Image Download END ");

console\_full.write("Image Download END ".getBytes());

System.out.print(formatter.format(date));

console\_full.write(formatter.format(date).getBytes());

System.out.print(" \n \n \n");

console\_full.write(" \n \n \n".getBytes());

}

//Function to get the temperatures of the stations

private static void getTemperatures( String code, FileOutputStream temperatures\_file, String temperaturesFilename) throws IOException {

//Print message to indicate the beginning of temperature application

SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

Date date = new Date();

System.out.print("Temperatures Download START ");

System.out.println(formatter.format(date));

System.out.println("Temperatures:");

//In console\_full.txt is written all the readable output

console\_full.write("Temperatures Download START ".getBytes());

console\_full.write(formatter.format(date).getBytes());

console\_full.write("\nTemperatures:\n".getBytes());

//opening a stream to the file

temperatures\_file=new FileOutputStream( temperaturesFilename + ".csv");

//Datagram Socket for sending request packets initialization.

DatagramSocket s = new DatagramSocket();

//Datagram Socket for receiving response packets initialization.

DatagramSocket r = new DatagramSocket(client\_listening\_port);

// running a loop for 00 to 99 to see how many temperature stations work ( update: only T00 works)

for( int i =0 ;i< 100 ;i++) {

String tempParam = String.format("%02d", i);

byte[] txbuffer = (code+"T"+tempParam).getBytes();

InetAddress hostAddress = InetAddress.getByAddress(hostIP);

DatagramPacket clientRequest = new DatagramPacket(txbuffer,txbuffer.length,

hostAddress,server\_listening\_port);

r.setSoTimeout(8000);

byte[] rxbuffer = new byte[2048];

DatagramPacket serverResponse = new DatagramPacket(rxbuffer,rxbuffer.length);

s.send(clientRequest);

try {

r.receive(serverResponse);

String message = new String(rxbuffer,0,serverResponse.getLength());

System.out.println("The length of a package = " + serverResponse.getLength() + "bytes");

temperatures\_file.write(( message +"\n" ).getBytes());

System.out.println(message);

console\_full.write((message+"\n").getBytes());

} catch (Exception x) {

System.out.println(x);

}

}

//closing the UDP sockets opened before

s.close();

r.close();

temperatures\_file.close();

//Print message to indicate the end of temperatures download

formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

date = new Date();

System.out.print("Temperatures END ");

console\_full.write("Temperatures END ".getBytes());

System.out.print(formatter.format(date));

console\_full.write(formatter.format(date).getBytes());

System.out.print(" \n \n \n");

console\_full.write(" \n \n \n".getBytes());

}

//Function for getting the DPCM samples of sound.

private static void getSoundfromSimpleDPCM(String samples\_diff\_filename, String samples\_filename,String Sound\_request\_params) throws IOException, LineUnavailableException {

//Print message to indicate the beginning DPCM sound application

SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

Date date = new Date();

System.out.print("Sound Sample DPCM Download START ");

System.out.println(formatter.format(date));

System.out.print("\n");

//In console\_full.txt is written all the readable output

console\_full.write("Sound Sample DPCM START ".getBytes());

console\_full.write(formatter.format(date).getBytes());

console\_full.write("\n".getBytes());

//opening streams to the files where application's data will be stored.

sound\_samples\_diff = new FileOutputStream( samples\_diff\_filename + ".csv");

sound\_samples = new FileOutputStream( samples\_filename + ".csv");

//Datagram Socket for sending request packets initialization.

DatagramSocket s = new DatagramSocket();

//Creating the request string from the function input arguments.

String request\_string = audio\_request\_code + Sound\_request\_params;

byte[] txbuffer = request\_string.getBytes();

InetAddress hostAddress = InetAddress.getByAddress(hostIP);

DatagramPacket clientRequest = new DatagramPacket(txbuffer,txbuffer.length,

hostAddress,server\_listening\_port);

//Datagram Socket for receiving request packets initialization

DatagramSocket r = new DatagramSocket(client\_listening\_port);

r.setSoTimeout(8000);

//Set the size to 128, because for simpple DPCM applications that's the length of the packets we reveice in bytes.

byte[] rxbuffer = new byte[128];

DatagramPacket serverResponse = new DatagramPacket(rxbuffer,rxbuffer.length);

s.send(clientRequest);

//masks to get the nibbles of the bytes

int nibble1mask = 0b11110000;

int nibble2mask = 0b00001111;

//values of B (beta) and mean value (mean) in DPCM is 1 and 0 (we can skip this parameters) in DPCM applixation

int beta = 1 ;

int mean\_value= 0;

ArrayList<Integer> ActualSamples\_Int= new ArrayList<>(); // arraylists for store the actual 2 samples values after demodulation

// the remaining number of sound packets

int nOfPacketsRemaining = N\_OF\_SOUND\_PACKETS;

do {

try {

r.receive(serverResponse);

for (int i = 0; i < 128; i++) {

int diff1, diff2, sample1;

int sample2=0;

// Getting the two difference between 2 samples from the two nibbles

diff1 =( ((nibble1mask & rxbuffer[i]) >> 4) - 8 ) \* beta; // beta =1 for DPCM, so we can skip it if we want to

diff2 =( (nibble2mask & rxbuffer[i]) - 8) \* beta; // beta =1 for DPCM, so we can skip it if we want to

// Stroring the samples' differences to the proper file.

sound\_samples\_diff.write(( (diff1) +"\n" ).getBytes());

sound\_samples\_diff.write(( (diff2) +"\n" ).getBytes());

//getting the two samples values

sample1 = diff1 + sample2;

sample2 = diff2 + sample1;

// Stroring the actual samples to the proper file.

sound\_samples.write(( (sample1) +"\n" ).getBytes());

sound\_samples.write(( (sample2) +"\n" ).getBytes());

//storring values of 2 samples difference in order

ActualSamples\_Int.add(sample1);

ActualSamples\_Int.add(sample2);

}

} catch (Exception x) {

System.out.println(x);

}

//Decrease the number of remaining packets.

nOfPacketsRemaining--;

}while (nOfPacketsRemaining > 0);

//Getting our clip in byte[] buffer form, for the lineOut.write() method.

byte[] clip = new byte[ActualSamples\_Int.size()];

for(int i = 0; i < ActualSamples\_Int.size(); i++) {

clip[i] = ActualSamples\_Int.get(i).byteValue();

}

//playing our sample in our computer's sound output device.

AudioFormat linearPCM = new AudioFormat(8000,8,1,true,false);

SourceDataLine lineOut = AudioSystem.getSourceDataLine(linearPCM);

lineOut.open(linearPCM,32000);

lineOut.start();

//each sound Packet we receice corresponds to 256 bytes of sound.

lineOut.write(clip,0,256\*N\_OF\_SOUND\_PACKETS);

lineOut.stop();

lineOut.close();

//closing the UDP sockets opened before

s.close();

r.close();

//closing the filestreams which store the application data to the proper files.

sound\_samples\_diff.close();

sound\_samples.close();

//Print message to indicate the end of DPCM sound application.

formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

date = new Date();

System.out.print("Sound Sample DPCM Download and Play END ");

console\_full.write("Sound Sample DPCM Download and Play END ".getBytes());

System.out.print(formatter.format(date));

console\_full.write(formatter.format(date).getBytes());

System.out.print(" \n \n \n");

console\_full.write(" \n \n \n".getBytes());

}

//Function for getting the AQDPCM samples of sound.

private static void getSoundfromAQDPCM(String samples\_diff\_filename, String samples\_filename,String meanValues\_filename,

String steps\_filename,String Sound\_request\_params) throws IOException, LineUnavailableException {

//Print message to indicate the start of AQDPCM sound application

SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

Date date = new Date();

System.out.print("Sound Sample ΑQDPCM Download START ");

System.out.println(formatter.format(date));

System.out.print("\n");

//In console\_full.txt is written all the readable output

console\_full.write("Sound Sample ΑQDPCM START ".getBytes());

console\_full.write(formatter.format(date).getBytes());

console\_full.write("\n".getBytes());

//opening streams to the files where data will be stored

sound\_samples\_diff = new FileOutputStream( samples\_diff\_filename + ".csv");

sound\_samples = new FileOutputStream( samples\_filename + ".csv");

sound\_mean\_value = new FileOutputStream( meanValues\_filename + ".csv");

sound\_steps = new FileOutputStream( steps\_filename + ".csv");

//Datagram Socket for sending request packets initialization.

DatagramSocket s = new DatagramSocket();

//Creating the request string from the function input arguments.

String request\_string = audio\_request\_code + Sound\_request\_params;

byte[] txbuffer = request\_string.getBytes();

InetAddress hostAddress = InetAddress.getByAddress(hostIP);

DatagramPacket clientRequest = new DatagramPacket(txbuffer,txbuffer.length,

hostAddress,server\_listening\_port);

//Datagram Socket for receiving response packets initialization.

DatagramSocket r = new DatagramSocket(client\_listening\_port);

r.setSoTimeout(8000);

//Set the size to 132, because for AQDPCM applications that's the length of the packets we reveice in bytes.

byte[] rxbuffer = new byte[132];

DatagramPacket serverResponse = new DatagramPacket(rxbuffer,rxbuffer.length);

//masks to get the nibbles

int nibble1mask = 0b11110000;

int nibble2mask = 0b00001111;

//values of B (beta) and mean value (mean) in DPCM

int beta = 0 ;

int meanValue;

byte[] quant\_step = new byte[4];

byte[] meanValueArr = new byte[4];

// arraylist for store the actual samples values after demodulation

ArrayList<Integer> ActualSamples\_Int= new ArrayList<>();

byte sign;

// the remaining number of sound packets, initialize it to the total number o packets we requested.

int nOfPacketsRemaining = N\_OF\_SOUND\_PACKETS;

// send our request

s.send(clientRequest);

do {

try {

r.receive(serverResponse);

// Getting mean value

if(( rxbuffer[1] & 0x80) !=0 ){

sign = (byte)0xff;

}

else{

sign = (byte)0x00;

}

meanValueArr[0] = rxbuffer[0];

meanValueArr[1] = rxbuffer[1];

meanValueArr[2] = sign;

meanValueArr[3] = sign;

meanValue = ByteBuffer.wrap(meanValueArr).order(ByteOrder.LITTLE\_ENDIAN).getInt();

// storing the quantizer's mean value to the proper file

sound\_mean\_value.write(( (meanValue) +"\n" ).getBytes());

// Getting quantizer step (b - beta)

if(( rxbuffer[3] & 0x80) !=0 ){

sign = (byte)0xff;

}

else{

sign = (byte)0x00;

}

quant\_step[0] = rxbuffer[2];

quant\_step[1] = rxbuffer[3];

quant\_step[2] = sign;

quant\_step[3] = sign;

beta = ByteBuffer.wrap(quant\_step).order(ByteOrder.LITTLE\_ENDIAN).getInt();

// storing the quantizer's step value to the proper file

sound\_steps.write(( (beta) +"\n" ).getBytes());

// Getting sample differences - and the actual samples

for (int i = 4; i < 132; i++) {

int diff1, diff2, sample1;

int sample2=0;

// Getting the two difference between 2 samples from the two nibbles

diff1 =( ((nibble1mask & rxbuffer[i]) >> 4) - 8 ) ;

diff2 =( (nibble2mask & rxbuffer[i]) - 8) ;

//storing the samples' differences to the proper file.

sound\_samples\_diff.write(( (diff1) +"\n" ).getBytes());

sound\_samples\_diff.write(( (diff2) +"\n" ).getBytes());

//getting the two samples values

sample1 = diff1\* beta + sample2 + meanValue;

sample2 = diff2\* beta + diff1\* beta + meanValue;

//storing the actual samples to the proper file.

sound\_samples.write(( ( sample1 & 0x000000FF) +"\n" ).getBytes());

sound\_samples.write(( ((sample1 & 0x0000FF00)>> 8) +"\n" ).getBytes());

sound\_samples.write(( ( sample2 & 0x000000FF) +"\n" ).getBytes());

sound\_samples.write(( ((sample2 & 0x0000FF00)>> 8) +"\n" ).getBytes());

//storring values of 2 samples difference in order

ActualSamples\_Int.add(( sample1 & 0x000000FF) );

ActualSamples\_Int.add(((sample1 & 0x0000FF00)>> 8));

ActualSamples\_Int.add(( sample2 & 0x000000FF) );

ActualSamples\_Int.add(((sample2 & 0x0000FF00)>> 8) );

}

} catch (Exception x) {

System.out.println(x);

}

//Decrease the number of remaining packets.

nOfPacketsRemaining--;

}while (nOfPacketsRemaining > 0);

//Getting our clip in byte[] buffer form, for the lineOut.write() method.

byte[] clip = new byte[ActualSamples\_Int.size()];

for(int i = 0; i < ActualSamples\_Int.size(); i++) {

clip[i] = ActualSamples\_Int.get(i).byteValue();

}

AudioFormat linearPCM = new AudioFormat(8000,16,1,true,false);

SourceDataLine lineOut = AudioSystem.getSourceDataLine(linearPCM);

lineOut.open(linearPCM,32000);

lineOut.start();

//each sound Packet we receice corresponds to 256 bytes of sound.

lineOut.write(clip,0,256\*N\_OF\_SOUND\_PACKETS);

lineOut.stop();

lineOut.close();

//closing the UDP sockets opened before

s.close();

r.close();

//Closing the filestreams to the files, where the application's data is stored.

sound\_samples\_diff.close();

sound\_samples.close();

sound\_mean\_value.close();

sound\_steps.close();

//Print message to indicate the end of AQDPCM sound application

formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

date = new Date();

System.out.print("Sound Sample ΑQDPCM Download and Play END ");

console\_full.write("Sound Sample ΑQDPCM Download and Play END ".getBytes());

System.out.print(formatter.format(date));

console\_full.write(formatter.format(date).getBytes());

System.out.print(" \n \n \n");

console\_full.write(" \n \n \n".getBytes());

}

private static void IthakiCopter(String telemetry\_filename) throws IOException {

//Print message to indicate the beginning of ithaki\_copter\_telemetry application

SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

Date date = new Date();

System.out.print("Ithaki copter Telemetry START ");

System.out.println(formatter.format(date));

System.out.print("\n");

//In console\_full.txt is written all the readable output

console\_full.write("Ithaki copter Telemetry START ".getBytes());

console\_full.write(formatter.format(date).getBytes());

console\_full.write("\n".getBytes());

//open stream to the file where telemetry data will be stored.

telemetry=new FileOutputStream( telemetry\_filename + ".csv");

//Datagram Socket for receiving response packets initialization.

DatagramSocket r = new DatagramSocket(48078);

r.setSoTimeout(8000);

byte[] rxbuffer = new byte[2048];

DatagramPacket serverResponse = new DatagramPacket(rxbuffer,rxbuffer.length);

String left\_motor, right\_motor, altitude, thermocracy, pressure;

// We get 40 responses for telemetry

for(int i=0; i<40 ;i++) {

try {

r.receive(serverResponse);

String telemPacket = new String(rxbuffer,0,serverResponse.getLength());

System.out.println(telemPacket);

console\_full.write((telemPacket+"\n").getBytes());

//getting the values from the servers' response

left\_motor = telemPacket.substring(40, 43);

right\_motor = telemPacket.substring(51, 54);

altitude = telemPacket.substring(64, 67);

thermocracy = telemPacket.substring(80, 86);

pressure = telemPacket.substring(96, 103);

//storing the measurements to the telemetry file.

telemetry.write(( (left\_motor + "," + right\_motor + "," + altitude + "," + thermocracy +

"," + pressure + "\n") ).getBytes());

} catch (Exception x) {

System.out.println(x);

}

}

//Print message to indicate the end of copter telemetry

formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

date = new Date();

System.out.print("Ithaki copter Telemetry END ");

console\_full.write("Ithaki copter Telemetry ".getBytes());

System.out.print(formatter.format(date));

console\_full.write(formatter.format(date).getBytes());

System.out.print(" \n \n \n");

console\_full.write(" \n \n \n".getBytes());

telemetry.close();

r.close();

}

private static void OBD() throws IOException {

//Print message

SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

Date date = new Date();

System.out.print("ODB-II Diagnostics START ");

System.out.println(formatter.format(date));

System.out.print("\n");

//In console\_full.txt is written all the readable output

console\_full.write("ODB-II Diagnostics START ".getBytes());

console\_full.write(formatter.format(date).getBytes());

console\_full.write("\n".getBytes());

//Datagram Socket for sending request packets initialization.

DatagramSocket s = new DatagramSocket();

byte[] txbuffer;

InetAddress hostAddress = InetAddress.getByAddress(hostIP);

//Datagram Socket for receiving response packets initialization.

DatagramSocket r = new DatagramSocket(client\_listening\_port);

r.setSoTimeout(8000);

byte[] rxbuffer = new byte[2048];

DatagramPacket serverResponse = new DatagramPacket(rxbuffer,rxbuffer.length);

//open streams to files when ODB diagnostic parameters will be saved.

OBD\_engireRunTime=new FileOutputStream( "OBD\_engineRunTimes" + ".csv");

OBD\_IntakeAirTemperature=new FileOutputStream( "OBD\_IntakeAirTemperature" + ".csv");

OBD\_ThrottlePosition=new FileOutputStream( "OBD\_ThrottlePosition" + ".csv");

OBD\_EngineRPM=new FileOutputStream( "OBD\_EngineRPM" + ".csv");

OBD\_VehicleSpeed=new FileOutputStream( "OBD\_VehicleSpeed" + ".csv");

OBD\_CoolantTemperature=new FileOutputStream( "OBD\_CoolantTemperature" + ".csv");

long startTime=System.currentTimeMillis();

/// where the value requested will be stored

int value;

//where XX and YY part of the response will be stored

int XXpart;

int YYpart;

//where XX and YY in hexadecimal will be stored

String XXpartinHexadecimal;

String YYpartinHexadecimal;

// 4 minutes are equal to 240000 milliseconds, so we set our programm to get ODB data for 240000 milliseconds.

while (System.currentTimeMillis()-startTime< 240000) {

try {

//Request and Response for engine Runtime

txbuffer = (vehicle\_obd\_II\_code +"OBD=01 1F\n" ).getBytes();

DatagramPacket clientRequest = new DatagramPacket(txbuffer,txbuffer.length,

hostAddress,server\_listening\_port);

s.send(clientRequest);

r.receive(serverResponse);

// storing the response in the proper form

XXpartinHexadecimal=(char)rxbuffer[6]+""+(char)rxbuffer[7];

YYpartinHexadecimal=(char)rxbuffer[9]+""+(char)rxbuffer[10];

XXpart=Integer.parseInt(XXpartinHexadecimal,16);

YYpart=Integer.parseInt(YYpartinHexadecimal,16);

value =256\*XXpart+YYpart;

OBD\_engireRunTime.write((String.valueOf(value) + "\n").getBytes());

//Request and Response for Intake Air Temperature

txbuffer = (vehicle\_obd\_II\_code +"OBD=01 0F\n" ).getBytes();

clientRequest = new DatagramPacket(txbuffer,txbuffer.length,

hostAddress,server\_listening\_port);

s.send(clientRequest);

r.receive(serverResponse);

// storing the response in the proper form

XXpartinHexadecimal=(char)rxbuffer[6]+""+(char)rxbuffer[7];

XXpart=Integer.parseInt(XXpartinHexadecimal,16);

value=XXpart-40;

OBD\_IntakeAirTemperature.write((String.valueOf(value) + "\n").getBytes());

//Request and Response for Throttle Position

txbuffer = (vehicle\_obd\_II\_code +"OBD=01 11\n" ).getBytes();

clientRequest = new DatagramPacket(txbuffer,txbuffer.length,

hostAddress,server\_listening\_port);

s.send(clientRequest);

r.receive(serverResponse);

// storing the response in the proper form

XXpartinHexadecimal=(char)rxbuffer[6]+""+(char)rxbuffer[7];

XXpart=Integer.parseInt(XXpartinHexadecimal,16);

value = (XXpart\*100)/255;

OBD\_ThrottlePosition.write((String.valueOf(value) + "\n").getBytes());

//Request and Response for Engine RPM

txbuffer = (vehicle\_obd\_II\_code +"OBD=01 0C\n" ).getBytes();

clientRequest = new DatagramPacket(txbuffer,txbuffer.length,

hostAddress,server\_listening\_port);

s.send(clientRequest);

r.receive(serverResponse);

// storing the response in the proper form

XXpartinHexadecimal=(char)rxbuffer[6]+""+(char)rxbuffer[7];

YYpartinHexadecimal=(char)rxbuffer[9]+""+(char)rxbuffer[10];

XXpart=Integer.parseInt(XXpartinHexadecimal,16);

YYpart=Integer.parseInt(YYpartinHexadecimal,16);

value= ((XXpart\*256)+YYpart)/4;

OBD\_EngineRPM.write((String.valueOf(value) + "\n").getBytes());

//Request and Response for Vehicle Speed

txbuffer = (vehicle\_obd\_II\_code +"OBD=01 0D\n" ).getBytes();

clientRequest = new DatagramPacket(txbuffer,txbuffer.length,

hostAddress,server\_listening\_port);

s.send(clientRequest);

r.receive(serverResponse);

// storing the response in the proper form

XXpartinHexadecimal=(char)rxbuffer[6]+""+(char)rxbuffer[7];

XXpart=Integer.parseInt(XXpartinHexadecimal,16);

value = XXpart;

OBD\_VehicleSpeed.write((String.valueOf(value) + "\n").getBytes());

//Request and Response for Coolant Temperature

txbuffer = (vehicle\_obd\_II\_code +"OBD=01 05\n" ).getBytes();

clientRequest = new DatagramPacket(txbuffer,txbuffer.length,

hostAddress,server\_listening\_port);

s.send(clientRequest);

r.receive(serverResponse);

XXpartinHexadecimal=(char)rxbuffer[6]+""+(char)rxbuffer[7];

XXpart=Integer.parseInt(XXpartinHexadecimal,16);

value = XXpart-40;

OBD\_CoolantTemperature.write((String.valueOf(value) + "\n").getBytes());

} catch (Exception x) {

System.out.println(x);

}

}

//closing the stream to the files where application data is stored

OBD\_engireRunTime.close();

OBD\_IntakeAirTemperature.close();

OBD\_ThrottlePosition.close();

OBD\_EngineRPM.close();

OBD\_VehicleSpeed.close();

OBD\_CoolantTemperature.close();

//closing the UDP sockets opened before

s.close();

r.close();

//Print end message that indicates the end of OBD application.

formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

date = new Date();

System.out.print("ODB-II Diagnostics END ");

System.out.println(formatter.format(date));

System.out.print("\n");

console\_full.write("ODB-II Diagnostics END ".getBytes());

console\_full.write(formatter.format(date).getBytes());

console\_full.write("\n".getBytes());

}

}