Εργασία Δίκτυα Υπολογιστών II – Source Code

Ονοματεπώνυμο: Μιχαήλ Καρατζάς
ΑΕΜ: 9137
email: mikalaki@ece.auth.gr
Εξάμηνο : 9ο
Δεδομένα και κώδικες της εργασίας εδώ

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_sam
ples","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_sam
ples","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 AQDPCM 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 AQDPCM 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 AQDPCM Download and Play END ");
        console_full.write("Sound Sample AQDPCM 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());
      }


}
```