



Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή

Δίκτυα Υπολογιστών II

Θεόδωρος Κατζάλης
AEM:9282
katzalis@auth.gr

April 30, 2021

Contents

1	Δομή του προγράμματος	2
2	UserApplication.java	3
3	applications	10
3.1	Echo.java	10
3.2	Media.java	11
3.3	Obd.java	20
3.4	Copter.java	23
4	Plots	25

1 Δομή του προγράμματος

```
applications
  Copter.java
  Echo.java
  Media.java
  Obd.java
plots
  plot.py
ascii
  audio.txt
  auto.txt
  copter_tcp.txt
  copter.txt
  echo.txt
  https.txt
  image.txt
  obd_tcp.txt
  obd.txt
  temp.txt
  test.txt
  welcome.txt
UserApplication.java
```

- Το αρχείο που βρίσκεται η main είναι το *UserApplication.java*.
- Στον φάκελο *applications*, δημιουργήσαμε ξεχωριστά αρχεία για κάθε εφαρμογή.
- Στον φάκελο *ascii*, βρίσκονται οι έξοδοι του προγράμματος figlet για ascii art λόγους, όπως έχει αναφερθεί και στο report!
- Στον φάκελο *plots*, έχουμε τέλος ένα python αρχείο για να δημιουργήσουμε τα διαγράμματα μας.

2 UserApplication.java

```
1 // it is considered in general a bad practise to use asterisks to import all the
2 // classes
3 import applications.*;
4 import java.awt.Desktop;
5 import java.io.*;
6 import java.lang.Math.*;
7 import java.lang.System;
8 import java.net.*;
9 import java.nio.charset.StandardCharsets;
10 import java.time.LocalDateTime;
11 import java.util.Arrays;
12 import java.util.Scanner;
13 import javax.sound.sampled.*;
14
15 class UserApplication {
16
17     public static void main(String[] args) throws Exception {
18
19         printWelcome();
20
21         // preamble
22         byte[] clientIP = {(byte)192, (byte)168, (byte)1, (byte)20};
23         byte[] hostIP = {(byte)155, (byte)207, (byte)18, (byte)208};
24         InetAddress clientAddress = InetAddress.getByAddress(clientIP);
25         InetAddress hostAddress = InetAddress.getByAddress(hostIP);
26         int serverPort = 38022;
27         int clientPort = 48022;
28
29         String requestCodeEcho = "E0818 ";
30         String requestCodeImage = "M2685UDP=1024";
31         String requestCodeSound = "A7269";
32         String requestCodeCopter = "Q2797";
33         String requestCodeVehicle = "V4118";
34
35         DatagramSocket socket = new DatagramSocket(clientPort);
36         long timeBefore = 0;
37
38         // control user input
39         int flag = 1;
40         do {
41             System.out.println(
42                 "\nPlease enter a number (1-11). Available options are:\n
43                 1) Echo with delay\n
44                 2) Echo no delay\n
45                 3) Temperature\n
46                 4) Image\n
47                 5) Music\n
48                 6) Vehicle UDP\n
49                 7) Ithakicopter UDP\n
50                 8) Autopilot\n
51                 9) HTTPS TCP\n
52                 10) Ithakicopter TCP\n
53                 11) Vehicle TCP");
54             String choiceApp = (new Scanner(System.in)).nextLine();
55
56             switch (choiceApp) {
57                 case "1":
58
```

```

59  /* ----- Echo with delay ----- */
60  printASCII("ascii/echo.txt");
61
62  File info = new File("logs/echo_info_delay.txt");
63  FileWriter writerInfo = new FileWriter(info);
64  writerInfo.write("Info:\n"
65                  + "The request code is " + requestCodeEcho + "\n");
66  writerInfo.write("Tic: " + LocalDateTime.now() + "\n");
67
68  File fileSamples = new File("logs/echo_samples_delay.txt");
69  FileWriter writerSamples = new FileWriter(fileSamples);
70
71  File fileThroughput = new File("logs/echo_throughput_delay.txt");
72  FileWriter writerThroughput = new FileWriter(fileThroughput);
73
74  File fileRto = new File("logs/rto.txt");
75  FileWriter writerRto = new FileWriter(fileRto);
76
77  // keep track how many 8 seconds have passed
78  int count8sec[] = new int[8];
79
80  // worst case we need 8 elements to store data
81  float throughput[] = new float[8];
82
83  long tic[] = new long[8];
84  timeBefore = System.currentTimeMillis();
85  for (int i = 0; i < 8; i++) {
86      tic[i] = timeBefore + i * 1000; // move per second
87  }
88
89  double rtt = 0;
90  double rtttd = 0;
91  double rto = 0;
92  int isFirst = 1;
93  int cumsum[] = new int[8];
94
95  while ((System.currentTimeMillis() - timeBefore) < 60000 * 4) {
96      long rtt =
97          Echo.execute(socket, hostAddress, serverPort, requestCodeEcho);
98      writerSamples.write(rtt + "\n");
99
100     // throughput moving average
101
102     for (int i = 0; i < 8; i++) {
103         // System.out.println("The element " + i + " has tic: " + tic[i]);
104         long toc = System.currentTimeMillis() - tic[i];
105         System.out.println("The element " + i + " has toc: " + toc);
106         if (toc < 8000 && toc > 0) {
107             cumsum[i] += 32 * 8; // assume no timeouts during the measurements
108             System.out.println("Cumsum: " + cumsum[i]);
109         } else if (toc > 8000) {
110             count8sec[i]++;
111             tic[i] = count8sec[i] * 8000 + timeBefore + i * 1000;
112             throughput[i] = cumsum[i] / (float)8;
113             System.out.println("I will flush " + cumsum[i] + " cumsum");
114             System.out.println("The throughput is: " + throughput[i]);
115
116             writerThroughput.write(throughput[i] + "\n");
117
118             cumsum[i] = 0; // let's start again for the next 8 seconds

```

```

119     }
120 }
121
122 // Retransmission timeout
123
124 // init values
125 if (isFirst == 1) {
126     rtt = rtt;
127     rtttd = rtt / 2;
128     rto = 1; // according to rfc
129     writerRto.write("RTT SRTT RTTd RTO\n");
130 }
131 double temp = rtt;
132 rtt = 0.875 * temp + 0.125 * rtt;
133
134 temp = rtttd;
135 rtttd = 0.75 * temp + 0.25 * Math.abs(rtt - rtt);
136
137 rto = rtt + 1.8 * rtttd;
138 writerRto.write(rtt + " " + rtt + " " + rtttd + " " + rto + "\n");
139
140 System.out.println();
141 isFirst = 0;
142 }
143
144 writerInfo.write("Toc: " + LocalDateTime.now());
145
146 writerInfo.close();
147 writerSamples.close();
148 writerThroughput.close();
149 writerRto.close();
150 socket.close();
151 break;
152
153 case "2":
154     /* ----- Echo no delay ----- */
155     printASCII("ascii/echo.txt");
156
157     info = new File("logs/echo_info_no_delay.txt");
158     writerInfo = new FileWriter(info);
159     writerInfo.write("Info:\n"
160         + "The request code is " + requestCodeEcho + "\n");
161     writerInfo.write("Tic: " + LocalDateTime.now() + "\n");
162
163     fileSamples = new File("logs/echo_samples_no_delay.txt");
164     writerSamples = new FileWriter(fileSamples);
165
166     fileThroughput = new File("logs/echo_throughput_no_delay.txt");
167     writerThroughput = new FileWriter(fileThroughput);
168
169     timeBefore = System.currentTimeMillis();
170     tic = new long[8];
171     // long toc[] = new long[8];
172     cumsum = new int[8]; // cumulative sum
173     throughput =
174         new float[8]; // worst case we need 8 elements to store data
175     count8sec = new int[8];
176     for (int i = 0; i < 8; i++) {
177         tic[i] = timeBefore + i * 1000; // move per second
178     }

```

```

179 while ((System.currentTimeMillis() - timeBefore) < 60000 * 4) {
180     long value = Echo.execute(socket, hostAddress, serverPort, "E0000");
181     writerSamples.write(value + "\n");
182
183     // throughput moving average
184
185     for (int i = 0; i < 8; i++) {
186         // System.out.println("The element " + i + " has tic: " + tic[i]);
187         long toc = System.currentTimeMillis() - tic[i];
188         System.out.println("The element " + i + " has toc: " + toc);
189         if (toc < 8000 && toc > 0) {
190             cumsum[i] += 32 * 8; // assume no timeouts during the measurements
191             System.out.println("Cumsum: " + cumsum[i]);
192         } else if (toc > 8000) {
193             count8sec[i]++;
194             tic[i] = count8sec[i] * 8000 + timeBefore + i * 1000;
195             throughput[i] = cumsum[i] / (float)8;
196             System.out.println("I will flush " + cumsum[i] + " cumsum");
197             System.out.println("The throughput is: " + throughput[i]);
198
199             writerThroughput.write(throughput[i] + "\n");
200
201             cumsum[i] = 0; // let's start again for the next 8 seconds
202         }
203     }
204     System.out.println();
205 }
206
207 writerInfo.write("Toc: " + LocalDateTime.now());
208
209 writerInfo.close();
210 writerSamples.close();
211 writerThroughput.close();
212 socket.close();
213 break;
214
215 case "3":
216     /* ----- Temperature ----- */
217     printASCII("ascii/temp.txt");
218
219     FileWriter writerTemp = new FileWriter(new File("logs/temp_info.txt"));
220     writerTemp.write("Info Temperature app:\n" + requestCodeEcho + "\n" +
221                     LocalDateTime.now() + "\n");
222
223     for (int i = 0; i < 1; i++) {
224         Echo.execute(socket, hostAddress, serverPort,
225                     requestCodeEcho + "T00");
226         System.out.println();
227     }
228
229     writerTemp.write(LocalDateTime.now() + "\n");
230     writerTemp.close();
231     socket.close();
232     break;
233
234 case "4":
235     /* ----- Image ----- */
236     printASCII("ascii/image.txt");
237
238

```

```

239 String encodingImage = "CAM=PTZDIR=R";
240 FileWriter writerImage =
241     new FileWriter(new File("logs/image_info_" + encodingImage));
242 writerImage.write(encodingImage + "\n" + requestCodeImage + "\n" +
243     LocalDateTime.now() + "\n");
244 for (int i = 0; i < 1; i++) {
245     Media.image(socket, hostAddress, serverPort,
246         requestCodeImage + encodingImage);
247     System.out.println();
248 }
249 writerImage.write(LocalDateTime.now() + "\n");
250 writerImage.close();
251 socket.close();
252 break;
253
254 case "5":
255     /* ----- Audio ----- */
256     printASCII("ascii/audio.txt");
257
258     String numAudioPackets = "999";
259     String[] type = {"F", "T"};
260     String[] encoding = {"AQ", ""};
261     String completeRequest =
262         requestCodeSound + encoding[0] + type[0] + numAudioPackets;
263
264     File infoMusic =
265         new File("logs/music_info_" + encoding[1] + type[0] + ".txt");
266     FileWriter writerInfoMusic = new FileWriter(infoMusic);
267     writerInfoMusic.write(requestCodeSound + "\nEncoding: " + encoding[1] +
268         "\nType: " + type[0] + LocalDateTime.now() +
269         "\n");
270
271     Media.audio(socket, hostAddress, serverPort, completeRequest);
272     System.out.println();
273
274     writerInfoMusic.write(LocalDateTime.now() + "\n");
275     writerInfoMusic.close();
276     socket.close();
277     break;
278
279 case "6":
280     /* ----- Vehicle OBD UDP ----- */
281     printASCII("ascii/obd.txt");
282
283     Obd.udpTelemetry(socket, hostAddress, serverPort, requestCodeVehicle);
284     socket.close();
285     break;
286
287 case "7":
288     /* ----- Ithakicopter UDP ----- */
289     printASCII("ascii/copter.txt");
290
291     socket = new DatagramSocket(48078);
292
293     System.out.println(
294         "For Ithakicopter UDP telemetry you need to open ithakicopter.jar");
295     System.out.print("Did you open it? If yes press ENTER to continue");
296     System.in.read();
297     Thread.sleep(1000); // pause a bit to catch up with the user
298     System.out.println("Press ENTER to exit");

```



```

299 Thread.sleep(1000);
300
301 FileWriter writerCopter =
302     new FileWriter(new File("logs/copter_info.txt"));
303 writerCopter.write("Info Ithakicopter app:\n" + LocalDateTime.now() +
304     "\n");
305 writerCopter.write("MOTOR ALTITUDE TEMPERATURE PRESSURE");
306
307 for (int i = 0; i < 4; i++)
308     Echo.execute(socket, hostAddress, serverPort, requestCodeEcho);
309
310 while (System.in.available() == 0) {
311     Copter.udpTelemetry(socket, hostAddress, serverPort, writerCopter);
312 }
313
314 writerCopter.write(LocalDateTime.now() + "\n");
315 writerCopter.close();
316 socket.close();
317 break;
318
319 case "8":
320     /* ----- Autopilot ----- */
321     printASCII("ascii/auto.txt");
322
323     socket = new DatagramSocket(48078);
324     Socket socketAuto = new Socket(hostAddress, 38048);
325
326     int lowerBound = 160;
327     int higherBound = 190;
328     Copter.autopilot(socket, hostAddress, serverPort, socketAuto,
329         Math.min(200, Math.max(150, lowerBound)),
330         Math.min(200, Math.max(150, higherBound)));
331     socketAuto.close();
332     break;
333
334 case "9":
335     /* ----- HTTPS TCP ----- */
336     printASCII("ascii/https.txt");
337
338     Socket httpsSocket = new Socket(hostAddress, 80);
339     https(httpsSocket);
340     httpsSocket.close();
341     break;
342
343 case "10":
344     /* ----- IthakicopterTCP ----- */
345     printASCII("ascii/copter_tcp.txt");
346
347     for (int i = 0; i < 4; i++)
348         Echo.execute(socket, hostAddress, serverPort, requestCodeEcho);
349
350     int target = 180;
351     for (int i = 0; i < 20; i++) {
352         System.out.println(
353             new String(Copter.tcpTelemetry(hostAddress, target)));
354     }
355
356     // socketCopter.close();
357     break;
358

```

```

359 case "11":
360     /* ----- Vehicle OBD TCP ----- */
361     printASCII("ascii/obd_tcp.txt");
362
363     for (int i = 0; i < 4; i++)
364         Echo.execute(socket, hostAddress, serverPort, requestCodeEcho);
365
366     Socket socketVehicle = new Socket(hostAddress, 29078);
367
368     FileWriter writerVehicleInfo =
369         new FileWriter(new File("logs/car_info.txt"));
370     writerVehicleInfo.write("Info Vehicle app:\n" + LocalDateTime.now() +
371                             "\n");
372     FileWriter writerVehicleData =
373         new FileWriter(new File("logs/car_telemetry.txt"));
374
375     timeBefore = System.currentTimeMillis();
376     float engineTime = 0;
377     while (engineTime < 60 * 4) {
378         engineTime = Obd.tcpTelemetry(socketVehicle, writerVehicleData);
379         System.out.println("The engine run time is " + engineTime + "\n");
380     }
381
382     writerVehicleInfo.write(LocalDateTime.now() + "\n");
383     writerVehicleInfo.close();
384     writerVehicleData.close();
385     socketVehicle.close();
386     break;
387
388 default:
389     System.out.println(
390         "Please provide a valid input. If you want to exit then press Control-C.\n");
391     flag = 0;
392 }
393 } while (flag == 0);
394
395 /* ----- Close UDP sockets ----- */
396 if (!socket.isClosed()) {
397     socket.close();
398     System.out.println("\nShuting down UDP sockets...");
399 }
400
401 System.out.println(
402     "\nx-----Hooray! Java application finished successfully
403     !-----x");
404 }
405
406 /**
407  * Print ASCII text
408  * @param filePath The path of the file with the ASCII characters to be printed
409  */
410 private static void printASCII(String filePath) {
411     try {
412         Scanner input = new Scanner(new File(filePath));
413         while (input.hasNextLine()) {
414             System.out.println(input.nextLine());
415         }
416         Thread.sleep(1500); // pause a little bit to enjoy the view
417     } catch (Exception x) {
418         System.out.println(x);
419     }
420 }

```

```

418     }
419 }
420
421 /**
422  * Print welcome screen ASCII with CYAN color
423  */
424 private static void printWelcome() {
425     // windows users may be not able to view colors on terminal
426     final String ANSI_CYAN = "\u001B[36m";
427     final String ANSI_RESET = "\u001B[0m";
428
429     try {
430         Scanner input = new Scanner(new File("ascii/welcome.txt"));
431         while (input.hasNextLine()) {
432             System.out.print(ANSI_CYAN); // add some color!
433             System.out.print(input.nextLine());
434             System.out.println(ANSI_RESET);
435         }
436         System.out.println();
437         System.out.print("Press ENTER to continue");
438         System.in.read(); // pause a little bit to enjoy the view
439
440     } catch (Exception x) {
441         System.out.println(x);
442     }
443 }
444
445 private static void https(Socket socket) {
446     try {
447         InputStream in =
448             socket.getInputStream(); // what I receive from the server
449         OutputStream out = socket.getOutputStream(); // what i send to the server
450
451         long timeBefore = System.currentTimeMillis();
452         out.write(
453             "GET /netlab/hello.html HTTP/1.0\r\nHost: ithaki.eng.auth.gr:80\r\n\r\n"
454             .getBytes());
455
456         byte[] inputBuffer = in.readAllBytes();
457         String message = new String(inputBuffer, StandardCharsets.US_ASCII);
458         System.out.println("Ithaki responded via TCP with: \n" + message);
459         System.out.println(
460             "Time response: " +
461             (System.currentTimeMillis() - timeBefore) / (float)1000 + " seconds");
462         socket.close();
463     } catch (Exception x) {
464         System.out.println(x + "TCP application failed");
465     }
466 }
467 }

```

3 applications

3.1 Echo.java

```

1 package applications;
2
3 import java.net.DatagramSocket;
4 import java.net.DatagramPacket;
5 import java.net.InetAddress;

```

```

6 import java.nio.charset.StandardCharsets;
7
8 public class Echo {
9
10     /*
11     * UDP TX/RX Echo application with delay
12     *
13     * WARNING: It doesn't close the DatagramSocket. You should do it manually if it is
14     * desired after the call of the function.
15     *
16     * @param requestCode If request code is set to E000 then the execute will have no delay
17     * for the RX
18     */
19     public static long execute(DatagramSocket socket, InetAddress hostAddress, int serverPort,
20     String requestCode) {
21         System.out.println("\n-----Echo application-----");
22
23         if (requestCode.equals("E0000")) System.out.println("Delay: OFF");
24         else if (requestCode.length()>5) System.out.println("Mode: Temperature\nDelay: OFF");
25         else System.out.println("Delay: OFF");
26
27         byte[] txbuffer = requestCode.getBytes();
28         byte[] rxbuffer = new byte[64];
29         long diff = 0;
30         try {
31             socket.setSoTimeout(3000);
32             DatagramPacket sendPacket = new DatagramPacket(txbuffer, txbuffer.length,
33             hostAddress, serverPort);
34             DatagramPacket receivePacket= new DatagramPacket(rxbuffer, rxbuffer.length);
35
36             // ACTION
37             socket.send(sendPacket);
38             System.out.println("The request code is: " + requestCode + "\nThe destination port
39 is: " + serverPort + "\nMy listening port (clientPort): " + socket.getLocalPort());
40             long timeBefore = System.currentTimeMillis();
41             System.out.println("My system time, when the request is sent, is: " + timeBefore);
42
43             // LISTEN
44             socket.receive(receivePacket);
45             long timeAfter = System.currentTimeMillis();
46             diff = timeAfter - timeBefore;
47             System.out.println("The time required to reveive a packet is: " + diff + "
48 milliseconds");
49             //System.out.println("The port that opened ithaki to send the request is : " +
50 receivePacket.getPort() + " and the address of ithaki is: " + receivePacket.getAddress());
51             String message = new String(receivePacket.getData(), StandardCharsets.US_ASCII);
52             // convert binary to ASCII
53             System.out.println("Ithaki responded with: " + message);
54         }
55         catch (Exception x) {
56             // x.printStackTrace(); // a more detailed diagnostic call
57             System.out.println(x);
58             System.out.println("Something went wrong about Echo application mode");
59         }
60         return diff;
61     }
62 }

```

3.2 Media.java

```

1 package applications;

```

```

2
3 import java.net.DatagramSocket;
4 import java.net.DatagramPacket;
5 import java.net.InetAddress;
6 import java.io.ByteArrayOutputStream;
7 import java.io.ByteArrayInputStream;
8 import java.io.File;
9 import java.io.FileWriter;
10 import java.io.FileOutputStream;
11 import java.io.IOException;
12 import java.awt.Desktop;
13 import java.util.Arrays;
14 import javax.sound.sampled.*;
15
16 public class Media {
17
18     private static String pathFileImage = "/home/tkatz/repos/ece-networks2/media/image/sandbox
/ithaki_image.jpg";
19     private static String pathFileSound = "/home/tkatz/repos/ece-networks2/media/music/sandbox
/track.wav";
20
21     public static void image(DatagramSocket socket, InetAddress hostAddress, int serverPort,
String requestCode) throws IOException {
22
23         //if (numImage != (Integer.parseInt(args[1])-1)){
24         //    continue; // readjust the camera as many time is requested via the command line
argument. Print only the result, the last request
25         //}
26
27         byte[] txbufferImage = requestCode.getBytes();
28         byte[] rxbufferImage = new byte[1024];
29         int countPackets = 0;
30         long timeBefore = System.currentTimeMillis();
31         long timeBeforePerPacket = System.currentTimeMillis();
32         //System.out.println("My system time, when the request is sent, is: " + timeBefore);
33
34         System.out.println("The request code is " + requestCode);
35         DatagramPacket sendPacket = new DatagramPacket(txbufferImage, txbufferImage.length,
hostAddress, serverPort);
36         DatagramPacket receivePacket= new DatagramPacket(rxbufferImage, rxbufferImage.length);
37
38         // TX
39         System.out.println("I am sleeping... Camera needs time to readjust");
40         try {
41             socket.send(sendPacket);
42             Thread.sleep(5000); // sleep in order for the camera to readjust
43         }
44         catch (Exception x) {
45             // x.printStackTrace(); // a more detailed diagnostic call
46             System.out.println(x);
47             System.out.println("Image application TX failed");
48         }
49
50         // RX
51         ByteArrayOutputStream bufferImage = new ByteArrayOutputStream();
52     outerloop:
53         try {
54             socket.setSoTimeout(3000);
55             for (;;) {
56                 socket.receive(receivePacket); // blocking command

```

```

57         countPackets++;
58
59         long timeAfterPerPacket = System.currentTimeMillis();
60         System.out.println("The time required to receive a packet is: " + (
timeAfterPerPacket - timeBeforePerPacket)/(float)1000 + " seconds");
61         timeBeforePerPacket = System.currentTimeMillis();
62
63         System.out.println("Packet No" + countPackets + ". Length of data: " +
rxbufferImage.length + ". The received bytes in hexadecimal format are:");
64         for (int i = 0; i<rxbufferImage.length; i++) {
65             System.out.print(String.format("%02X", rxbufferImage[i])); // convert
bytes to hexa string
66
67             bufferImage.write(rxbufferImage[i]); // dynamic byte allocation
68             if ((String.format("%02X", rxbufferImage[i]).equals("D9")) && (i!=0)) {
69                 if ((String.format("%02X", rxbufferImage[i-1]).equals("FF"))) {
70                     break outerloop; // stop writing when EOF (0xFFD9 delimiter)
71                 }
72             }
73         }
74         System.out.println();
75     }
76 }
77 catch (Exception x) {
78     // x.printStackTrace(); // a more detailed diagnostic call
79     System.out.println(x);
80     System.out.println("Image application RX failed");
81 }
82 long timeAfter = System.currentTimeMillis(); // get the time when the image is
received in bytes
83
84 // logs for the received byte content
85 System.out.println("\nComplete byte content of the image file in hexadecimal format:");
86 ;
87 byte[] completeDataImage = bufferImage.toByteArray();
88 for (byte i : completeDataImage) {
89     System.out.print(String.format("%02X", i)); // print hexadecimal the content of
the byte array
90 }
91 System.out.println("\n\nTotal number of packages: " + (countPackets));
92 System.out.println("How many Kbytes is the image? " + completeDataImage.length/(float)
1000);
93
94 // save image to a file
95 File imageFile = new File(pathFileImage);
96 FileOutputStream fos = null;
97 try {
98     fos = new FileOutputStream(imageFile);
99     fos.write(completeDataImage);
100     System.out.println("File has been written successfully");
101 }
102 catch (Exception x) {
103     // x.printStackTrace(); //
104     System.out.println("Image application error when writing the file:");
105 }
106
107 fos.close(); // close the OutputStream
108
109 // what time is o'clock?
110 System.out.println("Total amount of time to receive a frame: " + (timeAfter -

```

```

timeBefore)/(float)1000 + " seconds");
    timeAfter = System.currentTimeMillis(); // get the time when the file is ready
    System.out.println("Total amount of time to receive and write a frame in a .jpg file:
" + (timeAfter - timeBefore)/(float)1000 + " seconds");

    // open file image
    Desktop desktop = Desktop.getDesktop();
    if (imageFile.exists()) {
        //desktop.open(imageFile);
    }
}

public static void audio(DatagramSocket socket, InetAddress hostAddress, int serverPort,
String requestCode) {

    // parsing the requestCode
    // expecting requestCode: AXXXX + ("AQ" or "") + ("T" or "F") + numAudioPackets
    String encoding = "";
    String type = "F";
    String numAudioPackets = "";
    if (requestCode.length() == 11) {
        encoding = "AQ";
        type = requestCode.substring(7, 8);
        numAudioPackets = requestCode.substring(8, 11);
    }
    else {
        type = requestCode.substring(5, 6);
        numAudioPackets = requestCode.substring(6, 9);
    }
    System.out.println("Requested: Encoding: " + encoding + ". Type: " + type + ". Number
of packets: " + numAudioPackets);

    // TX
    byte[] txbufferSound = ("L02" + requestCode).getBytes();
    DatagramPacket sendPacket = new DatagramPacket(txbufferSound, txbufferSound.length,
hostAddress, serverPort);
    try {
        socket.send(sendPacket);
    }
    catch (Exception x) {
        // x.printStackTrace(); // a more detailed diagnostic call
        System.out.println(x);
        System.out.println("Audio application TX failed");
    }

    // RX
    byte[] dataSound = new byte[128];
    DatagramPacket receivePacket= new DatagramPacket(dataSound, dataSound.length);
    ByteArrayOutputStream bufferSound = new ByteArrayOutputStream();
    int countPackets = 0;
    int packetsSize = 0;
    long timeBefore = System.currentTimeMillis();
    long timeBeforePerPacket = System.currentTimeMillis();

    // create files
    File diffSamples = new File("logs/" + encoding + type + "diff_samples.txt") ;
    File fileSamples = new File("logs/" + encoding + type + "samples.txt") ;
    File fileMean = new File("logs/aqdpcm_mean.txt");
    File fileStep = new File("logs/aqdpcm_step.txt");

```

```

165 FileWriter writerDiffSamples = null;
166 FileWriter writerSamples = null;
167 FileWriter writerMean = null;
168 FileWriter writerStep = null;
169 try {
170     writerDiffSamples = new FileWriter(diffSamples);
171     writerSamples = new FileWriter(fileSamples);
172     writerMean = new FileWriter(fileMean);
173     writerStep = new FileWriter(fileStep);
174 }
175 catch (Exception x) {
176     System.out.println(x);
177     System.out.println("Failed to create a file writer for the DPCM");
178 }
179
180
181 try {
182     socket.setSoTimeout(3000);
183     for (int l = 0; l < Integer.parseInt(numAudioPackets); l++) {
184         timeBeforePerPacket = System.currentTimeMillis();
185         socket.receive(receivePacket);
186         countPackets++;
187         long timeAfterPerPacket = System.currentTimeMillis();
188         System.out.println("The time required to receive a packet is: " + (
timeAfterPerPacket - timeBeforePerPacket)/(float)1000 + " seconds");
189
190         packetsSize += dataSound.length;
191         System.out.println("Packet No" + countPackets + ". Length of data: " +
dataSound.length);
192
193         if (encoding.equals("")) {
194             // DPCM
195             bufferSound.write(dpcm(dataSound, writerDiffSamples, writerSamples));
196         }
197         else if (encoding.equals("AQ")) {
198             // AQ-DPCM
199
200             bufferSound.write(adpcm(dataSound, writerDiffSamples, writerSamples,
writerMean,
201                                     writerStep));
202         }
203         else {
204             System.out.println("This is not a valid request code");
205         }
206         System.out.println();
207     }
208 }
209 catch (Exception x) {
210     System.out.println(x);
211     System.out.println("Receiving/writing the audio data failed");
212 }
213
214 // close files
215 try {
216     writerSamples.close();
217     writerDiffSamples.close();
218     writerMean.close();
219     writerStep.close();
220 }
221 catch (Exception x) {

```



```

222     System.out.println(x);
223     System.out.println("Failed to close audio files");
224 }
225
226 long timeAfter = System.currentTimeMillis();
227
228 System.out.println("\nComplete byte content of the sound file in hexadecimal format:");
229 ;
230 byte[] completeDataSound = bufferSound.toByteArray();
231 for (byte i : completeDataSound) {
232     String hexa = String.format("%02X", i); // print hexadecimal the content of the
byte array
233     System.out.print(hexa);
234 }
235
236 System.out.println("\n\nTotal number of packages: " + (countPackets));
237 System.out.println("How many Kbytes is the sound? " + completeDataSound.length/(float)
1000 + "\nHow many Kbytes is the data that was actually sent? " + packetsSize/(float)1000);
238 System.out.println("Total amount of time to receive sound data: " + (timeAfter -
timeBefore)/(float)1000 + " seconds");
239
240 boolean isBigEndian = false; // only in 16 bit samples does matter. In AQ-DPCM we use
16 bit encoding
241 int encodingBits = 8;
242 if (encoding.equals("AQ")) {
243     isBigEndian = true;
244     encodingBits = 16;
245 }
246 AudioFormat modulationPCM = new AudioFormat(8000, encodingBits, 1, true, isBigEndian);
247 // play sound
248 try {
249     SourceDataLine outputAudio = AudioSystem.getSourceDataLine(modulationPCM);
250     //outputAudio.open(modulationPCM, 3200);
251     outputAudio.open(modulationPCM);
252     outputAudio.start();
253
254     System.out.println("Getting ready to hear some music?");
255     Thread.sleep(2000);
256     System.out.print("In 3");
257     Thread.sleep(1000);
258     System.out.print(", 2");
259     Thread.sleep(1000);
260     System.out.println(", 1...");
261     Thread.sleep(500);
262     System.out.println("Listening...");
263     Thread.sleep(500);
264     outputAudio.write(completeDataSound, 0, completeDataSound.length);
265     outputAudio.stop();
266     outputAudio.close();
267     System.out.println("\nSound application success!");
268 }
269 catch (Exception x) {
270     System.out.println(x);
271     System.out.println("Sound playing failed");
272 }
273
274 // save music to file
275 try{
276     ByteArrayInputStream bufferSoundInput = new ByteArrayInputStream(completeDataSound
);

```

```

276     AudioInputStream streamSoundInput = new AudioInputStream(bufferSoundInput,
modulationPCM, completeDataSound.length / modulationPCM.getFrameSize());
277     AudioSystem.write(streamSoundInput, AudioFileFormat.Type.WAVE, new File(
pathFileSound));
278     System.out.println("Sound file creation success");
279 }
280 catch (Exception x) {
281     System.out.println(x);
282     System.out.println("Sound file creation failed");
283 }
284 }
285
286 private static byte[] dpcm(byte[] dataSound, FileWriter writerDiffSamples,
287                             FileWriter writerSamples) {
288
289     ByteArrayOutputStream bufferSound = new ByteArrayOutputStream();
290     int init = 0;
291     int step = 1; // Trials: for 100 is pure noise, 4 good, 10 bad. I think below 4 you
are good. In general it shouldn't
292
293
294
295     for (int i = 0; i<dataSound.length; i++) {
296
297         String hexa = String.format("%02X", dataSound[i]); // print hexadecimal the
content of the byte array
298         System.out.print("Input: decimal: " + dataSound[i] + ", unsigned: " + Byte.
toUnsignedInt(dataSound[i]) + " and the hexa: " + hexa + ", ");
299         // get nibbles
300         int maskLow = 0x0F;
301         int maskHigh = 0xF0;
302         int nibbleLow = dataSound[i] & maskLow; // D[i] = x[i] - x[i-1]
303         int nibbleHigh = (dataSound[i] & maskHigh)>>4; // D[i-1] = x[i-1] - x[i-2]
304
305         // differences
306         int diffHigh = (nibbleHigh - 8)*step;
307         int diffLow = (nibbleLow - 8)*step;
308
309         // get samples
310         int sampleFirst = init + diffHigh;
311         int sampleSecond = sampleFirst + diffLow;
312         System.out.print("Masks high and low: " + maskHigh + ", " + maskLow + ". Masks in
hex: " + String.format("%02X", maskHigh) + ", " + String.format("%02X", maskLow) + ". Result
of mask: " + String.format("%02X", nibbleHigh) + ", " + String.format("%02X", nibbleLow) +
". Nibbles high and low: " + nibbleHigh + ", " + nibbleLow + ", so the actual differences
are: " + (nibbleHigh-8) + ", " + (nibbleLow-8) + " and samples: " + sampleFirst + ", " +
sampleSecond);
313         init = sampleSecond;
314
315         // check range
316         int max8 = (int)(Math.pow(2,7)) - 1;
317         int min8 = -(int)(Math.pow(2,7));
318         int[] samples = {sampleFirst, sampleSecond};
319         for (int j=0; j< samples.length; j++) {
320             if (samples[j]>max8) samples[j] = max8;
321             else if (samples[j]<min8) samples[j] = min8;
322         }
323
324         // write data to files
325         try {

```

```

326         writerDiffSamples.write(diffHigh + "\n" + diffLow + "\n");
327         writerSamples.write(samples[0] + "\n" + samples[1] + "\n");
328     }
329     catch (Exception x) {
330         System.out.println(x);
331         System.out.println("Failed to write data to DPCM file");
332     }
333
334     // write to buffer
335     byte[] decodedSound = new byte[2];
336     decodedSound[0] = (byte)sampleFirst;
337     decodedSound[1] = (byte)sampleSecond;
338     System.out.println(". Output: " + String.format("%02X", decodedSound[0]) + String.
format("%02X", decodedSound[1]));
339     try {
340         bufferSound.write(decodedSound);
341     }
342     catch (Exception x) {
343         System.out.println(x);
344         System.out.println("Decoding DPCM failed");
345     }
346 }
347
348
349
350     return bufferSound.toByteArray();
351
352 }
353
354 private static byte[] adpcm(byte[] dataSound, FileWriter writerDiffSamples,
355                             FileWriter writerSamples, FileWriter writerMean, FileWriter
writerStep) {
356
357     // get the header first
358     int mean = (Byte.toUnsignedInt(dataSound[1])<<8 | Byte.toUnsignedInt(dataSound[0]));
359     // be sure to not preserve the byte sign
360     int meanSigned = (dataSound[1]<<8 | dataSound[0]); // this is wrong. Not sure though?
361     System.out.println("dataSound[1]: " + String.format("%02X", dataSound[1]) + ",
dataSound[1]<<8: " + String.format("%02X", (Byte.toUnsignedInt(dataSound[1]))<<8));
362     System.out.println("The MSB of mean is " + String.format("%02X", dataSound[1]) + " and
the LSB of the mean is " + String.format("%02X", dataSound[0]) + ". The mean is " + mean +
" and signed " + meanSigned + " and in hex unsigned: " + String.format("%02X", mean) + "
and signed " + String.format("%02X", meanSigned));
363     int step = (Byte.toUnsignedInt(dataSound[3])<<8 | Byte.toUnsignedInt(dataSound[2]));
364     System.out.println("The MSB of step is " + String.format("%02X", dataSound[3]) + " and
the LSB of the step is " + String.format("%02X", dataSound[2]) + ". The step is " + step +
" and in hex: " + String.format("%02X", step));
365
366     try {
367         writerMean.write(meanSigned + "\n");
368         writerStep.write(step + "\n");
369     }
370     catch (Exception x) {
371         System.out.println(x);
372         System.out.println("Failed to write mean and step files for AQ-DPCM");
373     }
374
375     ByteArrayOutputStream bufferSound = new ByteArrayOutputStream();
376     int init = meanSigned; // in DPCM we don't know the init value, we assume zero. But
here we have data in the header.

```

```

376     for (int i = 3; i<dataSound.length; i++) {
377         // the sample may be bigger than byte. So you will need 16 bit encoding and store
each int to 2 bytes.
378         System.out.print("Input: " + String.format("%02X", dataSound[i]) + ", "); // print
hexadecimal the content of the byte array
379
380         // get nibbles
381         int maskLow = 0x0F;
382         int maskHigh = 0xF0;
383         int nibbleLow = (dataSound[i] & maskLow); // D[i] = x[i] - x[i-1], should be
unsigned
384         int nibbleHigh = (dataSound[i] & maskHigh)>>4; // D[i-1] = x[i-1] - x[i-2], should
be unsigned
385
386         // differences
387         int diffHigh = (nibbleHigh - 8)*step;
388         int diffLow = (nibbleLow - 8)*step;
389
390         // get samples (implement recursive formula)
391         int sampleFirst = init + diffHigh;
392         int sampleSecond = sampleFirst + diffLow;
393         System.out.print("Masks high and low: " + maskHigh + ", " + maskLow + ". Masks in
hex: " + String.format("%02X", maskHigh) + ", " + String.format("%02X", maskLow) + ". Result
of mask: " + String.format("%02X", nibbleHigh) + ", " + String.format("%02X", nibbleLow) +
". Nibbles high and low: " + nibbleHigh + ", " + nibbleLow + ", so the actual differences
are: " + (nibbleHigh-8)*step + ", " + (nibbleLow-8)*step + " and samples: " + sampleFirst +
", " + sampleSecond);
394         init = sampleSecond;
395
396         // check range
397         int max16 = (int)(Math.pow(2,15)) - 1;
398         int min16 = -(int)(Math.pow(2,15));
399         int[] samples = {sampleFirst, sampleSecond};
400         for (int j=0; j<samples.length; j++) {
401             if (samples[j]>max16) samples[j] = max16;
402             else if (samples[j]<min16) samples[j] = min16;
403         }
404         System.out.print(". The actual samples due to 16-bit restriction are: " + samples
[0] + " and " + samples[1] + " and in hex format: " + String.format("%02X", samples[0]) + "
, " + String.format("%02X", samples[1]) + ". In short " + (short)samples[0] + ", " + (short)
samples[1] + " and in hex format as a short: " + String.format("%02X", (short)samples[0])
+ ", " + String.format("%02X", (short)samples[1]));
405
406         // write data to files
407         try {
408             writerDiffSamples.write(diffHigh + "\n" + diffLow + "\n");
409             writerSamples.write(samples[0] + "\n" + samples[1] + "\n");
410         }
411         catch (Exception x) {
412             System.out.println(x);
413             System.out.println("Failed to write data to AQ-DPCM file");
414         }
415
416         // write to buffer
417         byte[] decodedSound = new byte[4];
418         decodedSound[0] = (byte)(samples[0]>>8); // MSB of sample 15-8
419         decodedSound[1] = (byte)samples[0]; // LSB of sample 7-0
420         decodedSound[2] = (byte)(samples[1]>>8);
421         decodedSound[3] = (byte)samples[1];
422         System.out.println(". Output: First sample " + String.format("%02X", decodedSound

```

```

[0]) + String.format("%02X", decodedSound[1]) + " second sample: " + String.format("%02X",
decodedSound[2]) + String.format("%02X", decodedSound[3]));
423     try {
424         bufferSound.write(decodedSound);
425     }
426     catch (Exception x) {
427         System.out.println(x);
428         System.out.println("Decoding DPCM failed");
429     }
430 }
431
432
433     return bufferSound.toByteArray();
434 }
435 }

```

3.3 Obd.java

```

1 package applications;
2 import java.io.File;
3 import java.io.FileWriter;
4 import java.io.BufferedReader;
5 import java.io.InputStream;
6 import java.io.InputStreamReader;
7 import java.io.OutputStream;
8 import java.net.DatagramSocket;
9 import java.net.DatagramPacket;
10 import java.net.Socket;
11 import java.net.InetAddress;
12 import java.nio.charset.StandardCharsets;
13
14 public class Obd {
15
16     private static String[] header = {"01 1F", "01 0F", "01 11", "01 0C", "01 0D", "01 05"};
17
18
19     public static void udpTelemetry(DatagramSocket socket, InetAddress hostAddress, int
serverPort, String requestCode) {
20
21         byte[] rxbuffer = new byte[16];
22         DatagramPacket receivePacket = new DatagramPacket(rxbuffer, rxbuffer.length);
23
24         for (int i = 0; i < header.length; i++) {
25
26             // TX
27             String completeCode = (requestCode + "OBD=" + header[i]);
28             byte[] txbuffer = completeCode.getBytes();
29             DatagramPacket sendPacket = new DatagramPacket(txbuffer, txbuffer.length,
hostAddress, serverPort);
30             System.out.println("Complete request: " + completeCode);
31             try {
32                 socket.send(sendPacket);
33             }
34             catch (Exception x) {
35                 // x.printStackTrace(); // a more detailed diagnostic call
36                 System.out.println(x);
37                 System.out.println("OBD vehicle application TX failed");
38             }
39             long timeBefore = System.currentTimeMillis();
40
41             // RX

```

```

42     try{
43         socket.setSoTimeout(3000);
44         socket.receive(receivePacket);
45         String message = new String(rxbuffer, StandardCharsets.US_ASCII);
46         System.out.println("Ithaki responded via UDP with: " + message);
47         System.out.println("Ithaki UDP time response: " + (System.currentTimeMillis()-
timeBefore)/(float)1000 + " seconds");
48
49         int[] values = parser(message);
50         formula(values[0], values[1], header[i]);
51     }
52     catch (Exception x) {
53         System.out.println(x);
54         System.out.println("RX UDP vehicle failed");
55     }
56 }
57
58 }
59
60 public static float tcpTelemetry(Socket socket, FileWriter writerVehicle) {
61
62     float engineTime = 0;
63
64     try {
65         InputStream in = socket.getInputStream();
66         OutputStream out = socket.getOutputStream();
67         BufferedReader bf = new BufferedReader(new InputStreamReader(in)); // wrapper on
top of the wrapper as java docs recommends
68
69         for (int i = 0; i < header.length; i++) {
70             out.write((header[i] + "\r").getBytes());
71             //out.flush();
72             long timeBefore = System.currentTimeMillis();
73             System.out.println("Created TCP socket and set output stream... Waiting for
response");
74
75             System.out.println("Header: " + header[i]);
76             String data = bf.readLine();
77             System.out.println("Ithaki responded via TCP with: " + data);
78             System.out.println("Ithaki TCP time response: " + (System.currentTimeMillis()-
timeBefore)/(float)1000 + " seconds");
79
80             int[] values = parser(data);
81             float value = formula(values[0], values[1], header[i]);
82             writerVehicle.write(value + " ");
83             if (header[i] == "01 1F") engineTime = value;
84         }
85
86         writerVehicle.write("\n");
87     }
88     catch (Exception x) {
89         System.out.println(x);
90         System.out.println("Oops... Vehicle OBD TCP failed");
91     }
92 }
93
94 return engineTime;
95 }
96
97 private static float formula(int first, int second, String header) {

```

```

98     float value = 0;
99     switch (header) {
100         case "01 1F":
101             int engineRunTime = first*256 + second;
102             System.out.println("Engine run time: " + engineRunTime);
103             value = engineRunTime;
104             break;
105
106         case "01 0F":
107             int intakeAirTemp = first - 40;
108             System.out.println("Intake Air Temperature: " + intakeAirTemp);
109             value = intakeAirTemp;
110             break;
111
112         case "01 11":
113             float throttlePos = (first*100)/(float)255;
114             System.out.println("Throttle position: " + throttlePos);
115             value = throttlePos;
116             break;
117
118         case "01 0C":
119             float engineRpm = ((first*256) + second)/(float)4;
120             System.out.println("Engine RPM: " + engineRpm);
121             value = engineRpm;
122             break;
123
124         case "01 0D":
125             int speed = first;
126             System.out.println("Vehicle speed: " + speed);
127             value = speed;
128             break;
129
130         case "01 05":
131             int coolantTemp = first -40;
132             System.out.println("Coolant Temperature: " + coolantTemp);
133             value = coolantTemp;
134             break;
135
136         default:
137             System.out.println("Something went wrong calculating formual for vehicle stats
138 ");
139     }
140     System.out.println();
141     return value;
142 }
143
144 private static int[] parser(String data) {
145     String byte1 = data.substring(6,8);
146     // how to convert hexadecimal string to int?
147     int first = Integer.parseInt(byte1, 16);
148     System.out.print("Parsing the data: 1st byte: " + byte1 + " and as an integer: " +
149 first);
150     String byte2 = "";
151     int second = 0;
152     if (data.length()>8) {
153         byte2 = data.substring(9,11);
154         second = Integer.parseInt(byte2, 16);
155         System.out.print(", 2nd byte: " + byte2 + " and as an integer: " + second);
156     }
157 }

```

```

156     System.out.println();
157
158     int[] temp = {first, second};
159     return temp;
160 }
161 }

```

3.4 Copter.java

```

1 package applications;
2
3 import java.io.FileWriter;
4 import java.io.BufferedReader;
5 import java.io.ByteArrayOutputStream;
6 import java.io.InputStream;
7 import java.io.InputStreamReader;
8 import java.io.OutputStream;
9 import java.net.DatagramSocket;
10 import java.net.DatagramPacket;
11 import java.net.Socket;
12 import java.net.InetAddress;
13 import java.nio.charset.StandardCharsets;
14 import java.io.ByteArrayOutputStream;
15
16 public class Copter {
17     public static String udpTelemetry(DatagramSocket socket, InetAddress hostAddress, int
serverPort, FileWriter writerCopter) {
18         // TX
19         // open ithakicopter.jar
20
21
22         //RX only
23         byte[] rxbuffer = new byte[128];
24         DatagramPacket receivePacket = new DatagramPacket(rxbuffer, rxbuffer.length);
25
26         long timeBefore = System.currentTimeMillis();
27         String telemetry = new String();
28         try{
29             socket.setSoTimeout(3000);
30             socket.receive(receivePacket);
31             telemetry = new String(rxbuffer, StandardCharsets.US_ASCII);
32             //System.out.print("Time repsonse: " + (System.currentTimeMillis() - timeBefore)/(
float)1000);
33             System.out.println("Received data via UDP: " + telemetry);
34
35             String[] tokensMotor = telemetry.split("LMOTOR=");
36             String[] tokensAltitude = telemetry.split("ALTITUDE=");
37             String[] tokensTemp = telemetry.split("TEMPERATURE=");
38             String[] tokensPress = telemetry.split("PRESSURE=");
39             writerCopter.write(tokensMotor[1].substring(0, 3) + " ");
40             writerCopter.write(tokensAltitude[1].substring(0, 3) + " ");
41             writerCopter.write(tokensTemp[1].substring(1, 6) + " ");
42             writerCopter.write(tokensPress[1].substring(0, 7) + "\n");
43         }
44         catch (Exception x) {
45             System.out.println(x);
46             System.out.println("RX UDP ithakicopter failed");
47         }
48         return telemetry;
49     }
50 }

```



```

51 public static String tcpTelemetry(InetAddress hostAddress, int target) {
52     String telemetry = "";
53     Socket socket = new Socket();
54     try {
55         socket = new Socket(hostAddress, 38048);
56         InputStream in = socket.getInputStream();
57         OutputStream out= socket.getOutputStream();
58         BufferedReader bf = new BufferedReader(new InputStreamReader(in)); // wrapper on
59 top of the wrapper as java docs recommends
60         ByteArrayOutputStream bos = new ByteArrayOutputStream();
61
62         String command = "AUTO FLIGHTLEVEL=" + target + " LMOTOR=" + target + " RMOTOR=" +
63 target + " PILOT \r\n";
64         //System.out.print("Request: " + command);
65         out.write(command.getBytes());
66         out.flush();
67
68         //in.skipNBytes(427);
69         for (int i = 0; i < 14; i++) {
70             bos.write((bf.readLine() + "\n").getBytes());
71         }
72         String data = new String(bos.toByteArray(), StandardCharsets.US_ASCII);
73         //System.out.println("Received data via TCP: " + data);
74
75         String[] tokens = data.split("\n");
76         // take only the useful data and skip the info ithaki sent
77         telemetry = tokens[13];
78     }
79     catch (Exception x) {
80         System.out.println(x);
81         System.out.println("Oops... Ithakicopter TCP failed");
82     }
83     try {
84         socket.close();
85     }
86     catch (Exception x) {
87         System.out.println(x);
88         System.out.println("Failed to close socket for ithakicopter TCP");
89     }
90     return telemetry;
91 }
92
93 /*
94  * tcpTelemetry function for the TX and udpTelemetry for RX. The way that these two
95 functions are implemented force the autopilot
96  * to be used with a combination of these two. We want to send a command only if it is
97 needed and we want to listen all the time
98  * to get feedback.
99  */
100 public static void autopilot(DatagramSocket listen, InetAddress hostAddress, int
101 serverPort, Socket send, int lowerBound, int higherBound) {
102     lowerBound = Math.min(lowerBound, higherBound);
103     higherBound = Math.max(lowerBound, higherBound);
104
105     int target = (lowerBound + higherBound)/2;
106     int motor = -1;

```

```

106
107     try {
108         System.out.println("AUTOPILOT: ON");
109         System.out.println("You need to open ithakicopter.jar. Press ENTER to continue...");
110         System.in.read();
111         System.out.println("Press Control-C to exit...");
112         Thread.sleep(1000);
113         for (;;) {
114             if ((motor < (lowerBound)) || (motor > (higherBound))) {
115                 System.out.println("Send packet. Readjust...");
116                 tcpTelemetry(hostAddress, target);
117             }
118
119             String telemetry = Copter.udpTelemetry(listen, hostAddress, serverPort, null);
120             String[] tokens = telemetry.split("LMOTOR=");
121             motor = Integer.parseInt(tokens[1].substring(0,3)); // get motor values
122
123             System.out.println("Parsed motor values: " + motor);
124         }
125     }
126     catch (Exception x) {
127         System.out.println(x);
128         System.out.println("AUTOPILOT failed");
129     }
130 }
131
132 }

```

4 Plots

```

1 from scipy.stats import norm
2 import matplotlib.pyplot as plt
3 import matplotlib.mlab as mlab
4 import pandas as pd
5 from numpy import genfromtxt
6 import numpy as np
7
8
9 # data = pd.read_csv('../logs/echo_samples_delay.csv', sep=',', header=None)
10 # data.plot(kind='bar')
11 # plt.ylabel('frequency')
12 # plt.xlabel('Number of packets')
13 # plt.title('Histogram response time')
14 # plt.show()
15
16 #plt.hist(rtt,histtype = 'bar', bins='auto', density=1, alpha=0.7)
17 # print("Length of the array is: " + str(len(rtt)))
18
19
20 # Response time diagram
21 x = genfromtxt('../logs/session2/echo_samples_delay.txt', delimiter=',')
22 x = genfromtxt('../logs/session2/echo_samples_no_delay.txt', delimiter=',')
23 x = genfromtxt('../logs/session2/echo_throughput_delay.txt', delimiter=',')
24 x = genfromtxt('../logs/session2/echo_throughput_no_delay.txt', delimiter=',')
25 x = genfromtxt('../logs/session1/AQFsamples.txt')
26 x = genfromtxt('../logs/session2/Fsamples.txt')
27 x = genfromtxt('../logs/session2/Tsamples.txt')
28 x = genfromtxt('../logs/session2/second_clip/aqdpcm_mean.txt')
29 x = genfromtxt('../logs/session2/second_clip/aqdpcm_step.txt')
30 x = genfromtxt('../logs/second_clip/aqdpcm_step.txt')

```

```

31 plt.subplot(2,1,1)
32 plt.plot(x, 'm')
33 plt.xlabel('Number of samples', fontsize=12)
34 plt.ylabel('Amplitude', fontsize=12)
35 plt.grid(True)
36
37 plt.subplot(2,1,2)
38 plt.xlabel('Number of samples', fontsize=12)
39 plt.ylabel('Amplitude', fontsize=12)
40 plt.grid(True)
41 plt.plot(x, 'm')
42 plt.xlim(100, 200)
43 plt.show()
44
45
46 #Retransmission timeout plot
47 data = genfromtxt('../logs/session2/rto.txt', delimiter=' ')
48 rtt = data[1:,0]
49 srtt = data[1:,1]
50 rtttd = data[1:,2]
51 rto = data[1:,3]
52 plt.plot(rtt, label = "RTT")
53 plt.plot(srtt, label = "SRTT")
54 plt.plot(rtttd, label = "RTTd")
55 plt.plot(rto, label = "RTO")
56 plt.xlabel('Number of packets', fontsize=12)
57 plt.ylabel("Time response", fontsize=12)
58 plt.legend()
59 plt.show()
60
61 # Copter
62 data = genfromtxt('../logs/session2/copter_2nd_run/copter_info.txt', delimiter=' ')
63 rtt = data[1:,0]
64 srtt = data[1:,1]
65 rtttd = data[1:,2]
66 rto = data[1:,3]
67 plt.plot(rtt, label = "MOTOR")
68 plt.plot(srtt, label = "ALTITUDE")
69 plt.plot(rtttd, label = "TEMPERATURE")
70 plt.plot(rto, label = "PRESSURE")
71 plt.xlabel('Number of packets', fontsize=12, labelpad=10)
72 plt.ylabel("Data", fontsize=12, labelpad=10)
73 plt.legend()
74 plt.grid(True);
75 plt.yticks(np.arange(0, 1200, 50))
76 plt.show()
77
78 #Vehicle
79 data = genfromtxt('../logs/car_telemetry.txt', delimiter=' ')
80 rtt = data[0:,0]
81 srtt = data[0:,1]
82 rtttd = data[0:,2]
83 rto = data[0:,3]
84 s = data[0:, 4]
85 t = data[0:, 5]
86 plt.subplot(211)
87 plt.plot(rtt, label = "Engine run time")
88 plt.plot(rto, label = "Engine RPM")
89 plt.xlabel('Number of packets', fontsize=12, labelpad=10)
90 plt.ylabel("Data", fontsize=12, labelpad=10)

```

```

91 plt.legend()
92 plt.grid(True);
93 plt.subplot(212)
94 plt.plot(srtt, label = "Intake air temperature")
95 plt.plot(rtttd, label = "Throttle position")
96 plt.plot(s, label = "Vehicle speed")
97 plt.plot(t, label = "Coolant temperature")
98 plt.xlabel('Number of packets', fontsize=12, labelpad=10)
99 plt.ylabel("Data", fontsize=12, labelpad=10)
100 plt.legend()
101 plt.grid(True);
102 #plt.yticks(np.arange(0, 2500, 25))
103 plt.show()

```