

# Protocolo de Ligação de Dados

Relatório do 1º trabalho laboratorial



Mestrado Integrado em Engenharia Informática e  
Computação

Redes de Computadores

**Grupo xx:**

Francisco Rodrigues - 2013056271

João Nogueira - 201303882

Marta Lopes - 201208067

Faculdade de Engenharia da Universidade do Porto  
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

30 de Outubro de 2015

## Conteúdo

<b>1</b>	<b>Sumário</b>	<b>3</b>
<b>2</b>	<b>Introdução</b>	<b>3</b>
<b>3</b>	<b>Arquitetura</b>	<b>4</b>
3.1	<i>Application Layer</i> e <i>Link Layer</i> . . . . .	4
3.2	<i>Interface</i> . . . . .	4
<b>4</b>	<b>Estrutura do código</b>	<b>5</b>
4.1	<i>Application Layer</i> . . . . .	5
4.2	<i>Link Layer</i> . . . . .	5
<b>5</b>	<b>Casos de uso principais</b>	<b>6</b>
<b>6</b>	<b>Protocolo de ligação lógica</b>	<b>7</b>
6.1	Principais aspectos funcionais . . . . .	7
6.2	Funções implementadas na <i>linkLayer</i> . . . . .	7
6.2.1	<i>ll.open</i> . . . . .	7
6.2.2	<i>ll.close</i> . . . . .	7
6.2.3	<i>ll.write</i> . . . . .	8
6.2.4	<i>ll.read</i> . . . . .	8
<b>7</b>	<b>Protocolo de aplicação</b>	<b>9</b>
7.1	Principais aspectos funcionais . . . . .	9
7.2	Funções implementadas na <i>applicationLayer</i> . . . . .	9
<b>8</b>	<b>Validação</b>	<b>10</b>
<b>9</b>	<b>Elementos de valorização</b>	<b>12</b>
<b>10</b>	<b>Conclusões</b>	<b>13</b>
<b>11</b>	<b>Anexos</b>	<b>14</b>

## 1 Sumário

Este relatório tem como objetivo explicar o primeiro projeto, realizado para esta unidade curricular, denominado "Protocolo de Ligação de Dados". Este projeto consiste no envio de informação de um computador para outro, através do uso de porta de série. Foram assim implementados programas para ler e escrever a informação a ser enviada.

O projeto foi finalizado com sucesso, sendo que os dados foram enviados e recebidos de forma correcta. Foram também incluídos a prevenção e correção de erros ao longo da transmissão, restabelecendo a transmissão quando os erros acontecem.

## 2 Introdução

Iremos então descrever o nosso trabalho, que consiste em implementar um protocolo de ligação de dados, de acordo com a especificação descrita no guião, de uma forma mais teórica e pormenorizada para assim poderem ser avaliados certos aspectos que não seriam possíveis avaliar durante a apresentação na aula. O ambiente de desenvolvimento utilizado foi em PC's com *Linux*, a linguagem de programação foi C e as portas de série existentes realizavam comunicação assíncrona.

O protocolo de ligação de dados pretende assim fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um cabo de série. As funções utilizadas serão a de criação e sincronismo de tramas que irão organizar os dados a ser enviados (*framing*), a do estabelecimento/conclusão da ligação, a numeração de tramas, o controlo de fluxo, a confirmação de envio sem erros e o controlo dos erros que poderão ser criados por *timeouts*, tramas fora da sequência esperada ou retransmissões.

O nosso relatório terá então as **secções principais** seguintes:

- **Arquitetura:** especificação dos blocos funcionais e da *interface*;
- **Estrutura do código:** API's, principais estruturas de dados, principais funções e a sua relação com a arquitetura;
- **Casos de uso principais:** identificar os principais aspectos, abordando as sequências de chamadas de funções;
- **Protocolo de ligação lógica:** identificar os principais aspectos funcionais da *linkLayer*, descrevendo a estratégia de implementação;
- **Protocolo de aplicação:** identificar os principais aspectos funcionais da *applicationLayer*, descrevendo a estratégia de implementação;
- **Validação:** testes efetuados ao programa com apresentação de resultados;
- **Elementos de valorização:** identificação dos elementos implementados, descrevendo a estratégia de implementação.

## 3 Arquitetura

### 3.1 *Application Layer e Link Layer*

O projeto está organizado em duas *layers* que vão ser responsáveis pela correta funcionalidade do mesmo. A *layer* que vai servir para a aplicação tem os ficheiros *applicationLayer.c* e *applicationLayer.h* e a ligação lógica está representada nos ficheiros *linkLayer.c* e *linkLayer.h* com um ficheiro \*.h auxiliar, o *linkLayerAux.h*.

A **camada de ligação lógica**, contém várias funções, servindo estas para a configuração e abertura da porta de série, o envio de comandos, envio e receção de mensagens e para a realização dos processos de *stuffing* e *destuffing*.

A **camada de aplicação**, que vai depender da camada de ligação lógica utilizando algumas das suas funções, disponibilizará funções onde serão especificados o envio e receção de pacotes de dados, o envio e receção de pacotes de controlo e finalmente a leitura e a escrita do ficheiro a enviar.

### 3.2 *Interface*

A interface na linha de comandos está implementada nos ficheiros *cli.c* e *cli.h* que têm as funções que vão permitir a escolha de valores de alguns parâmetros referentes à transferência do ficheiro, existindo limites impostos de acordo com o guião. Os parâmetros a serem especificados pelo utilizador serão a *baudrate*, o tamanho máximo para a mensagem, o número de *timeouts* máximo no caso de falha de comunicação e a duração de cada *timeout*. Além disso, o utilizador terá também de dizer se será o transmissor ou o recetor e qual é a sua porta de série. Depois de o utilizador especificar cada um destes valores a aplicação vai iniciar, sendo que terá uma informação inicial mostrando os parâmetros com a escolha feita pelo utilizador.

## 4 Estrutura do código

### 4.1 Application Layer

A camada da aplicação está implementada nos ficheiros *applicationLayer.c* e *applicationLayer.h* e é representada por uma estrutura na qual se encontram um *char* que representa o modo do programa (Receptor/Transmissor) e um inteiro chamado *debug* que permite fazer o *debugging* através de prints ao longo do código.

```
1 typedef struct {  
2     char status;  
3  
4     int debug;  
5 } applicationLayer;
```

Struct linkLayer

### 4.2 Link Layer

É nos ficheiros *linkLayer.c*, *linkLayer.h* e *linkLayerAux.h* que está implementada a camada de ligação de dados representada por uma estrutura na qual se encontram:

- A porta de série;
- O Baud Rate;
- O número de sequência;
- O timeout;
- O número de tentativas;
- O tamanho máximo de cada pacote;
- Estrutura de estatísticas.

```
1 typedef struct {  
2     int fd;  
3     char port[20];  
4     int baudRate;  
5     unsigned int sequenceNumber;  
6     unsigned int timeout;  
7     unsigned int numTransmissions;  
8     int packSize;  
9  
10    char status;  
11    int debug;  
12  
13    Statistics * stat;  
14 } linkLayer;
```

Struct linkLayer

## **5 Casos de uso principais**

(identificação; sequências de chamada de funções)

## 6 Protocolo de ligação lógica

O protocolo de ligação lógica está implementado na camada *linkLayer*, camada da qual depende a camada *applicationLayer*.

### 6.1 Principais aspectos funcionais

- Configurar a porta de série como é pretendido;
- Repor a configuração da porta de série como originalmente após a transferência dos dados pretendidos;
- Estabelecer a ligação de dados utilizando a porta de série;
- Enviar comandos;
- Enviar/receber mensagens;
- Processo de *Stuffing* e *Destuffing* dos *packets* recebidos da camada *applicationLayer*.

### 6.2 Funções implementadas na *linkLayer*

#### 6.2.1 *ll\_open*

Esta função é responsável por configurar a porta de série com as opções pretendidas (utilizando, por exemplo, o *baudRate* escolhido pelo utilizador) e por guardar numa variável que passa por argumento à função *configure*. Depois de configurar a porta de série, do lado do emissor envia o comando SET e aguarda a resposta UA do recetor que, ao ser recebida termina a função. Utiliza o alarme para controlar os *timeouts*. Do lado do recetor aguarda a recepção do comando SET e, ao recebê-lo responde com o comando UA.

#### 6.2.2 *ll\_close*

Do lado do receptor, esta função começa por aguardar a recepção do comando DISC, respondendo com o mesmo comando. Após o envio deste comando, e imediatamente antes de reestabelecer as configurações originais da porta de série aguarda a recepção do comando UA por parte do emissor. Do lado do emissor, esta função começa por enviar o comando DISC e aguarda pela resposta com o mesmo comando por parte do receptor (implementando o alarme para manter controlo dos *timeouts*). Após a correta recepção de DISC envia o último comando do programa, comando UA. Termina por fazer *resetConfiguration*, função na qual é feito um *sleep* de um segundo por forma a garantir que as configurações originais da porta de série não são restabelecidas antes de que toda a informação tenha sido passada.

### 6.2.3 llwrite

Esta função começa por alocar memória num *buffer* no qual a informação será organizada (antes do processo de *stuffing*). Após alocar a memória necessária começa por atribuir os valores de FLAG, A, C e BCC de acordo com o número de sequência da trama. A partir de BCC é copiada para o *buffer* toda a informação recebida por argumento desta função da *applicationLayer* correspondente ao *packet* a enviar. Antes de terminar o preenchimento do *buffer* coloca o BCC2 ('ou' exclusivo dos octetos do *packet*) e a FLAG para terminar. Depois de ter a trama preenchida e antes de começar o processo de envio, envia o *buffer* para a função *stuff* que retorna um *buffer* com a trama pronta para envio. Envia a trama através da porta de série e aguarda a resposta por parte do recetor, o que vai determinar o caminho a seguir pela função, que pode variar entre terminar a função pois houve a recepção de RR, ou re-enviar a mesma trama por ter sido rejeitada. A mesma trama pode ser re-enviada por ocorrência de *timeout*, o que também é controlado nesta função através da implementação de alarmes. Esta função pode também terminar pela ocorrência de *timeouts* maior do que o número máximo definido previamente pelo utilizador.

### 6.2.4 llread

A função *llread* começa por alocar memória para o *buffer* que vai ser recebido, entrando imediatamente a seguir num ciclo do qual apenas sai quando algo é lido da porta de série. Após a leitura o *buffer* lido é passado para a função responsável pelo processo de *destuffing* que retorna a trama de Informação recebida "descodificada". Os valores de BCC e de BCC2 são verificados por forma a garantir que a trama foi recebida sem erros, caso o tenha sido é interpretado o número de sequência e se tudo for o pretendido é enviado o comando RR, caso contrário é enviado o comando REJ, pedindo que a mesma seja enviada novamente.



## 7 Protocolo de aplicação

O protocolo de aplicação está implementado na camada *applicationLayer*, camada esta que depende, como dito anteriormente, da camada *linkLayer*.

### 7.1 Principais aspectos funcionais

- Envio e recepção de pacotes de dados;
- Envio e recepção de pacotes de controlo;
- Leitura do ficheiro a enviar;
- Escrita do ficheiro a enviar.

### 7.2 Funções implementadas na *applicationLayer*

A primeira função a ser implementada em *applicationLayer* foi a função **main** que chama e gere as diversas funções da *Interface* para o programa inicializar com as configurações desejadas pelo utilizador. Vai também chamar as funções *ll\_open*, *ll\_close*, *readFile* e *sendFile*.

De seguida, é implementada a função **initStat** que será responsável por inicializar os valores da *struct* que vai conter as estatísticas finais do programa.

É implementada também a função **fillLinkLayer** que inicializa a *struct linkLayer* onde estão atributos como *baudRate*, *timeout*, etc.

A próxima função implementada foi a **createFirstControlPacket** que vai gerir o primeiro pacote de controlo, é neste pacote que é enviado o tamanho do ficheiro a enviar bem como o nome do mesmo.

Foram também implementadas as funções **sendFile** e **readFile**. Na função *sendFile* vai ser lida toda a informação do ficheiro a enviar e guardá-la num *buffer* de chars, a partir do qual a informação será lida para o envio. Após a leitura chama a função responsável pelo envio do primeiro *control packet* e envia-o para o recetor, o *llwrite*. Depois do envio do primeiro *control packet* é calculado o número de pacotes necessários a enviar tendo em conta o tamanho máximo definido pelo utilizador. Os pacotes de dados vão sendo criados com a informação necessária e com os dados do ficheiro em si, sendo chamada a função *llwrite* para enviar os vários pacotes individualmente. Em *readFile* o ficheiro vai sendo recebido e escrito em disco. Cada pacote vai ser interpretado, depois de passar pela função *llread*, e vão ser tomadas medidas diferentes dependendo de que pacote for (primeiro, último ou um pacote de dados). Se for o primeiro *control packet* é guardada a informação do nome e tamanho do ficheiro. Se for o último é criado o ficheiro e toda a informação lida é escrita em disco. Se for um pacote de dados a informação é adicionada ao final do *buffer* de leitura.

## 8 Validação

Para verificar se a transferência do ficheiro *pinguim.gif* foi efetuada com sucesso decidimos colocar uma verificação de início e fim de envio, durante o envio do ficheiro adicionamos uma barra de progresso que iria sendo actualizada à medida que os packets iam sendo enviados.

```
*****
***** RCOM project - 1 *****
*****
# Initial Information #

      Mode: Transmitter
    Baud rate: B57600
    Msg Max Size: 300
    Attempts: 3
    Timeout: 3
*****
```

Figura 1: Informações iniciais - Modo Transmissor

```
#### Attempting connection... ####
#### Connection established. ####

Size of file expected: 10968
Completed: 100.00% [=====]
```

Figura 2: Início do envio e barra de progresso - Modo Transmissor

```
# Final Statistics #

Sent Messages: 38
Received RR: 40
Received REJ: 0

      Total Timeouts: 2
*****

#### Terminating connection... ####
#### Connection terminated. ####
```

Figura 3: Fim do envio e estatísticas - Modo Transmissor

```

*****
***** RCOM project - 1 *****
*****

# Initial Information #

      Mode: Receiver
      Baud rate: B57600
      Msg Max Size: 300
      Attempts: 3
      Timeout: 3
*****

```

Figura 4: Informações iniciais - Modo Recetor

```

#### Attempting connection...  ####
#### Connection established.  ####

Completed: 100.00% [=====]

```

Figura 5: Início do envio e barra de progresso - Modo Recetor

```

*****
***** RCOM project - 1 *****
*****

# Initial Information #

      Mode: Receiver
      Baud rate: B57600
      Msg Max Size: 300
      Attempts: 3
      Timeout: 3
*****

```

Figura 6: Fim do envio e estatísticas - Modo Recetor

Adicionamos também o modo *debug* em que iriam ser feitos mais *prints* à medida que o ficheiro ia sendo enviado, sendo também possível verificar quando houvesse *timeouts* ou envio/receção de REJ's.

## **9 Elementos de valorização**

(identificação dos elementos de valorização implementados; descrição da estratégia de implementação com apresentação de pequenos extratos de código)

## 10 Conclusões

(síntese da informação apresentada nas secções anteriores; reflexão sobre os objectivos de aprendizagem alcançados)

## 11 Anexos

```
1  #include "linkLayer.h"
2  //#include "cli.h"
3
4  int main(int argc, char** argv);
5
6  void fillLinkLayer();
7
8  char * createFirstControlPacket(int * packetSize, char **
   fileSizeChar, char ** name);
9
10 int sendFile();
11
12 void getNameAndSizeOfFile(char ** packet_1, int
   sizeofPacket, int * fileSize, char ** fileName);
13
14 Statistics* initStat(Statistics * stats);
```

Anexo 1 - applicationLayer.h

```
1  #include "utilities.h"
2  #include "linkLayerAux.h"
3
4
5  int ll_open(int * flag, int * stop, int * count, linkLayer
   * ll, struct termios * oldtio);
6
7  int ll_close(int * flag, int * stop, int * count, linkLayer
   * ll, struct termios * oldtio);
8
9  int llwrite(int * stop, linkLayer * ll, char * buffer, int
   length);
10
11 int llread(linkLayer * ll, char ** buffer);
```

Anexo 2 - linkLayer.h

```
1  void writeMsg(int * fd, char aFlag, char cFlag);
2
3  int readResponse(int * fd, int * flag, char aFlag, char cFlag);
4
5  void configure(linkLayer * ll, struct termios * oldtio);
6
7  void resetConfiguration(int * fd, struct termios * oldtio);
8
9  void triggerAlarm();
10
11 int readSenderResponse(linkLayer * ll);
12
13 char * stuff(char **deStuffed, int deStuffedLength, int *
   bufSize);
14
15 int readInfo(int * fd, int * flag, char * buffer);
16
```

```

17 int desStuff(char * destuffed, char ** stuffed, linkLayer * ll)
    ;
18
19 int removeFrameHeaderAndTrailer(char ** buffer, int
    sizeofInfoRead);

```

### Anexo 3 - linkLayerAux.h

```

1  #include <sys/types.h>
2  #include <sys/stat.h>
3  #include <fcntl.h>
4  #include <termios.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <unistd.h>
9  #include <signal.h>
10 #include <stdint.h>
11
12 #define BAUDRATE B9600
13 #define MODEMDEVICE "/dev/ttyS0"
14 #define _POSIX_SOURCE 1 // POSIX compliant source
15 #define FALSE 0
16 #define TRUE 1
17
18 #define FLAG 0x7e
19 #define A_1 0x03
20 #define A_2 0x01
21
22 #define C_0 0x0
23 #define C_1 0x20
24
25 #define C_RR_0 0x1
26 #define C_RR_1 0x21
27 #define C_REJ_0 0x5
28 #define C_REJ_1 0x25
29
30 #define C_SET 0x07
31 #define C_UA 0x03
32 #define C_DISC 0x0b
33
34 #define ESCAPE 0x7d
35 #define FLAG_EXC 0x5e
36 #define ESCAPE_EXC 0x5d
37
38 #define BCC A^C_SET
39 #define SETLEN 5
40 #define TIMEOUT 3
41 #define ATTEMPTS 3
42
43 #define MAX_PACKET_SIZE 350
44
45 #define UALENGTH 5
46
47 typedef struct {
48     char status;
49

```

```

50     int debug;
51 } applicationLayer;
52
53 typedef struct {
54     int sentMessages;
55     int receivedMessages;
56
57     int timeouts;
58
59     int numSentRR;
60     int numReceivedRR;
61
62     int numSentREJ;
63     int numReceivedREJ;
64 } Statistics;
65
66 typedef struct {
67     int fd;
68     char port[20];
69     int baudRate;
70     unsigned int sequenceNumber;
71     unsigned int timeout;
72     unsigned int numTransmissions;
73     int packSize;
74
75     char status;
76     int debug;
77
78     Statistics * stat;
79 } linkLayer;
80
81
82
83 int getFileSize(FILE* file);
84
85 char * getLine(void);

```

#### Anexo 4 - utilities.h

```

1  // #include <stdio.h>
2  #include "utilities.h"
3
4  void clearScreen();
5
6  int initialMenu();
7
8  void flushIn();
9
10 int chooseBaudrate();
11
12 int chooseMaxSize();
13
14 int chooseTimeout();
15
16 int chooseNumTransmissions();
17
18 void showInitialInfo(linkLayer * ll, applicationLayer * al);

```



```

19
20 void printProgressBar(int current, int total);
21
22 void printStats(applicationLayer * al, Statistics * stats);
23
24 void chooseFileName(char ** fileName);

```

#### Anexo 5 - cli.h

```

1  #include "applicationLayer.h"
2
3
4  int * flag;
5  int * count;
6  int * stop;
7
8  applicationLayer * al;
9  linkLayer * ll;
10
11 int main(int argc, char** argv) {
12
13     //***** Check if the arguments are
14     corrected *****
15     if (argc != 1)
16     {
17         printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/
18             ttyS1\n");
19         exit(1);
20     }
21     //
22     *****
23
24     al = (applicationLayer *) malloc(sizeof(applicationLayer));
25     ll = (linkLayer *) malloc(sizeof(linkLayer));
26     fillLinkLayer();
27
28     int firstChoice = -1;
29     while (firstChoice < 0) { firstChoice = initialMenu(); }
30
31     if (firstChoice == 1) { (*al).status = 'W'; }
32     else { (*al).status = 'R'; }
33
34     firstChoice = -1;
35     while (firstChoice < 0) { firstChoice = choosePort(); }
36
37     char port[] = "/dev/ttySX\0";
38     if (firstChoice == 1) { port[9] = '0'; }
39     else { port[9] = '4'; }
40     strncpy((*ll).port, port, 11);
41
42     firstChoice = -1;
43     while (firstChoice == -1) { firstChoice = chooseBaudrate();
44         }
45     switch(firstChoice) {

```

```

44     case 1:
45         (*ll).baudRate = B300;
46         break;
47     case 2:
48         (*ll).baudRate = B600;
49         break;
50     case 3:
51         (*ll).baudRate = B1200;
52         break;
53     case 4:
54         (*ll).baudRate = B1800;
55         break;
56     case 5:
57         (*ll).baudRate = B2400;
58         break;
59     case 6:
60         (*ll).baudRate = B4800;
61         break;
62     case 7:
63         (*ll).baudRate = B9600;
64         break;
65     case 8:
66         (*ll).baudRate = B19200;
67         break;
68     case 9:
69         (*ll).baudRate = B38400;
70         break;
71     case 10:
72         (*ll).baudRate = B57600;
73         break;
74     case 11:
75         (*ll).baudRate = B115200;
76         break;
77     default:
78         printf("\nError in choice of baudrate\n");
79         return -1;
80 }
81
82 firstChoice = -1;
83 while (firstChoice < 0) { firstChoice = chooseMaxSize(); }
84 (*ll).packSize = firstChoice;
85
86 firstChoice = -1;
87 while (firstChoice < 0) { firstChoice = chooseTimeout(); }
88 (*ll).timeout = firstChoice;
89
90 firstChoice = -1;
91 while (firstChoice < 0) { firstChoice =
    chooseNumTransmissions(); }
92 (*ll).numTransmissions = firstChoice;
93
94 count = malloc(sizeof(int));
95 flag = malloc(sizeof(int));
96 stop = malloc(sizeof(int));
97 *count = 0;
98 *flag = TRUE;

```

```

99     *stop = FALSE;
100
101     struct termios oldtio;
102
103     (void) signal(SIGALRM, triggerAlarm); // instala rotina que
104                                             atende interrupcao
105
106     showInitialInfo(ll, al);
107
108     printf("\n#### Attempting connection...   ####\n");
109     if (ll_open(flag, stop, count, ll, &oldtio) < 0) {
110         printf("\nError in ll_open\n");
111         return -1;
112     }
113     printf("\n#### Connection established.   ####\n");
114
115     if ((*al).status == 'W') {
116         if ((*al).debug == TRUE) {
117             printf("\n-----Sending control
118                 packet 1-----\n");
119         }
120         if (sendFile() < 0) {
121             printf("\nError in sendFile\n");
122             return -1;
123         }
124         if ((*al).debug == TRUE) {
125             printf("\n-----Sent control
126                 packet 1-----\n");
127         }
128     }
129     else if ((*al).status == 'R') {
130         if ((*al).debug == TRUE) {
131             printf("\n-----Receiving
132                 control packet 1
133                 -----\n");
134         }
135     }
136     printf("\n");
137     if (readFile() < 0) {
138         printf("\nError in readFile\n");
139         return -1;
140     }
141     if ((*al).debug == TRUE) {
142         printf("\n-----Received
143                 control packet 1
144                 -----\n");
145     }
146 }
147
148 printStats(al, ll->stat);
149
150 printf("\n\n#### Terminating connection...   ####\n");
151 if (ll_close(flag, stop, count, ll, &oldtio) < 0) {
152     printf("\nError in ll_close\n");
153     return -1;
154 }

```

```

147     }
148     printf("\n#### Connection terminated.   ####\n");
149
150     free(count);
151     free(flag);
152     free(stop);
153
154     free(al);
155     free((*ll).stat);
156     free(ll);
157
158     //printf("\nTerminated!\n");
159
160     return 0;
161 }
162
163 Statistics* initStat(Statistics * stats) {
164     (*stats).sentMessages = 0;
165     (*stats).receivedMessages = 0;
166     (*stats).timeouts = 0;
167     (*stats).numSentRR = 0;
168     (*stats).numReceivedRR = 0;
169     (*stats).numSentREJ = 0;
170     (*stats).numReceivedREJ = 0;
171     return;
172 }
173
174 void fillLinkLayer() {
175     (*ll).baudRate = BAUDRATE;
176     (*ll).sequenceNumber = 0;
177     (*ll).timeout = TIMEOUT;
178     (*ll).numTransmissions = ATTEMPTS;
179     (*ll).packSize = MAX_PACKET_SIZE + 6;
180     (*al).debug = FALSE;
181
182     (*ll).status = (*al).status;
183
184     (*ll).stat = malloc(sizeof(Statistics));
185     initStat((*ll).stat);
186 }
187
188 char * createFirstControlPacket(int * packetSize, char **
    fileSizeChar, char ** name) {
189     *packetSize = 24;
190     char * control = malloc(sizeof(char) * (*packetSize));
191     int sizeOfName = 11;
192     uint8_t size = (sizeOfName & 0xFF);
193     //char con[] = "10 ola.txt18";
194     char * co = malloc(sizeof(char) * 100);
195     *co = '1';
196     *(co+1) = '0';
197     *(co+2) = size;
198     int i = 3;
199     while (i < (i + (int)strlen(*name))) {
200         *(co + i) = *(*name + (i-3));
201         i++;

```

```

202     }
203     *(co + i) = '1';
204     *(co + i + 1) = '8';
205     *(co + i + 2) = '\0';
206     /*char con[] = "10 pinguim.gif18";
207     con[2] = size;*/
208     strcpy(control, co);
209     free(co);
210
211     //strncpy(control + 12, *fileSizeChar, 8);
212     strncpy(control + 16, *fileSizeChar, 8);
213
214     return control;
215 }
216
217 int sendFile() {
218     int packetSize;
219
220     char * fileName;
221     fileName = malloc(sizeof(char) * 100);
222     chooseFileName(&fileName);
223
224     printf("\n\nChosen is: [%s] Size is: [%d]\n\n", fileName, (
225         int)strlen(fileName));
226
227     FILE * pfd = fopen("./pinguim.gif", "r");
228     int fileSize = getFileSize(pfd);
229
230     printf("\n\nSize of file expected: %d\n\n", fileSize);
231
232     char * fullFile = malloc(sizeof(char) * fileSize);
233     char * fullFileStart = fullFile;
234     fread(fullFile, fileSize, 1, pfd);
235     char * fileSizeChar = malloc(sizeof(char) * 8);
236     sprintf(fileSizeChar, "%d", fileSize);
237
238     char * packet_1 = createFirstControlPacket(&packetSize, &
239         fileSizeChar, &fileName);
240     free(fileSizeChar);
241
242     //Sends First control packet
243     if ((*al).debug == TRUE) {
244         printf("\n_-----
245             \nFirst Control Packet");
246     }
247     llwrite(stop, ll, packet_1, packetSize);
248
249     int packetCounter = 0;
250     int numberOfPackets = fileSize / (((*ll).packSize - 6) - 4)
251     ;
252     if ((fileSize % (((*ll).packSize - 6) - 4)) > 0)
253         numberOfPackets++;
254     while (packetCounter < numberOfPackets) {
255         if ((*al).debug == TRUE) {
256             printf("\n

```

```

254         }
255         char * infoPacket = malloc(sizeof(char) * ((*ll).
                packSize - 6));
256         *infoPacket = '0';
257         *(infoPacket + 1) = (char) packetCounter;
258         char k = (char) (((*ll).packSize - 6) - 4);
259         uint8_t l1 = ((k & 0xFF00) >> 8);
260         *(infoPacket + 2) = l1;
261         uint8_t l2 = (k & 0x00FF);
262         *(infoPacket + 3) = l2;
263
264         int i = 4;
265         while (i < ((*ll).packSize - 6) && ((packetCounter*(((
                ll).packSize - 6)-4) + (i-4)) < fileSize)) {
266             *(infoPacket + i) = *fullFile;
267             fullFile++;
268             i++;
269         }
270
271         if ((*al).debug == TRUE) {
272             printf("\nPacket size is: %d\n", i);
273         }
274         if (llwrite(stop, ll, infoPacket, i) < 0)
275         {
276             printf("\nError in llwrite\n");
277             free(packet_1);
278             free(fullFileStart);
279             fclose(pfd);
280             free(infoPacket);
281
282             printProgressBar((packetCounter*(((ll).packSize - 6)
                -4) + (i-4)), fileSize);
283
284             return -1;
285         }
286
287         printProgressBar((packetCounter*(((ll).packSize - 6)
                -4) + (i-4)), fileSize);
288
289         free(infoPacket);
290         packetCounter++;
291         (*ll).stat->sentMessages++;
292     }
293     printf("\n");
294
295     //Sends Last control Packet
296     *packet_1 = '2';
297     if ((*al).debug == TRUE) {
298         printf("\n-----
                \nLast Control Packet");
299     }
300     llwrite(stop, ll, packet_1, packetSize);
301
302     free(packet_1);

```

```

303     free(fullFileStart);
304     fclose(pfd);
305     free(fileName);
306     return 0;
307 }
308
309 int readFile() {
310
311     char * finalFile;
312     char * finalFileToStore;
313     char * fileName;
314     int fileSize;
315     int packetCounter = 0;
316
317     int cn = FALSE;
318     while (cn == FALSE) {
319         char * packet_1;
320         int sizeOfPacket = -1;
321         if ((*al).debug == TRUE) {
322             printf("\n
n-----
\nPacket Received\n");
323         }
324         while (sizeOfPacket < 0) {
325             sizeOfPacket = llread( ll, &packet_1);
326             if (sizeOfPacket < 0)
327                 free(packet_1);
328         }
329         if ((*al).debug == TRUE) {
330             printf("\nSize of packet is: %d\n", sizeOfPacket);
331         }
332
333         if (sizeOfPacket < 0) {
334             free(packet_1);
335             continue;
336         } else if (*packet_1 == '2') {
337             cn = TRUE;
338
339             char * name = malloc(sizeof(char) * 50);
340             char path [] = "./received/";
341             int i = 0;
342             while (i < 11) {
343                 *(name + i) = path[i];
344                 i++;
345             }
346             strcpy(name+i, fileName);
347
348             FILE * pfd = fopen(name, "w");
349             fwrite(finalFile, fileSize, 1, pfd);
350
351             fclose(pfd);
352         } else if (*packet_1 == '1') {
353             getNameAndSizeOfFile(&packet_1, sizeOfPacket, &
fileSize, &fileName);
354
355             finalFile = malloc(sizeof(char) * fileSize);

```

```

356         finalFileToStore = finalFile;
357     } else if (*packet_1 == '0') {
358         int i = 4;
359         while (i < sizeofPacket) {
360             *finalFileToStore = *(packet_1 + i);
361             i++;
362             finalFileToStore++;
363         }
364
365         printProgressBar((packetCounter*((*ll).packSize -
366             6)-4) + (i-4)), fileSize);
367         packetCounter++;
368         ll->stat->receivedMessages++;
369     } else {
370         free(packet_1);
371     }
372 }
373 printf("\n");
374
375 free(fileName);
376 free(finalFile);
377
378 return 0;
379 }
380
381 void getNameAndSizeOfFile(char ** packet_1, int sizeofPacket,
382     int * fileSize, char ** fileName) {
383     int j = (*packet_1 + 2);
384     int i = 3;
385
386     *fileName = malloc(sizeof(char) * (j + 2));
387
388     //Get the name of the file
389     while (i < (j + 3)) {
390         *(*fileName + (i - 3)) = (*packet_1 + i);
391         i++;
392     }
393     *(*fileName + (i - 3)) = '\0';
394
395     //Get the size of the file
396     //i++; //Size of the byte that describes the size
397     //Because of the 8 (size of 2 bytes)
398     char rest[sizeofPacket - i];
399     j = i;
400     j++;
401
402     while (i < sizeofPacket) {
403         rest[i - j] = (*packet_1 + i);
404         i++;
405     }
406
407     rest[i - j] = '\0';
408
409     *fileSize = atoi(rest);

```



410 }

## Anexo 6 - applicationLayer.c

```
1  #include "linkLayer.h"
2
3  int * flagPointer;
4  int * countPointer;
5  int * timeoutAlarm;
6
7  //***** Function to send the message
   *****
8  void writeMsg(int * fd, char aFlag, char cFlag) {
9      tcflush(*fd, TCOFLUSH); // Clean output buffer
10
11     //***** Setting the flags to send *****
12     unsigned char SET[SETLEN];
13     SET[0] = FLAG;
14     SET[1] = aFlag;
15     SET[2] = cFlag;
16     SET[3] = SET[1] ^ SET[2]; //BCC
17     SET[4] = FLAG;
18     //*****
19
20     write(*fd, SET, SETLEN); //Sending the info
21 }
22 //
   *****
23
24 //***** Read the response of the receiver
   *****
25 int readResponse(int * fd, int * flag, char aFlag, char cFlag)
26 {
27     int res;
28     char response[5];
29
30
31     //***** While that controls the reading of the
       response of the receiver *****
32     unsigned int stateMachine = 0;
33     while (stateMachine < 5) { // state machine control
34         char readChar;
35         res = read(*fd,&readChar,1); // returns after 1 char
           input
36
37         if (!*flag && (res == 1)) {
38             switch (stateMachine) {
39                 case 0:
40                     if (readChar == FLAG) {
41                         response[stateMachine] = readChar;
42                         stateMachine = 1;
43                     }
44                     break;
45                 case 1:
46                     if (readChar == FLAG) {
```

```

47         break;
48     } else if (readChar == aFlag) {
49         response[stateMachine] = readChar;
50         stateMachine = 2;
51         break;
52     } else {
53         stateMachine = 0;
54         break;
55     }
56     case 2:
57     if (readChar == FLAG) {
58         stateMachine = 1;
59         break;
60     } else if (readChar == cFlag) {
61         response[stateMachine] = readChar;
62         stateMachine = 3;
63         break;
64     } else {
65         stateMachine = 0;
66         break;
67     }
68     case 3:
69     if (readChar == FLAG) {
70         stateMachine = 1;
71         break;
72     } else if (readChar == (aFlag^cFlag)) {
73         response[stateMachine] = readChar;
74         stateMachine = 4;
75         break;
76     } else {
77         stateMachine = 0;
78         break;
79     }
80     case 4:
81     switch(readChar) {
82         case FLAG:
83             response[stateMachine] = readChar;
84             stateMachine = 5;
85             break;
86         default:
87             stateMachine = 0;
88             break;
89     }
90     break;
91     }
92 }
93 else if (*flag)
94     break;
95 }
96
97 //
98
99 if(*flag)
100 return -1;

```

```

101
102     return 0;
103 }
104
105 //***** Read the response of the receiver
106 //*****
107 int readInfo(int * fd, int * flag, char * buffer) {
108     int res;
109     int end = FALSE;
110     char * bufferP = buffer;
111     int c = 0;
112
113     tcflush(*fd, TCIFLUSH);
114     //***** While that controls the reading of the
115     //***** response of the receiver *****
116     while (end == FALSE) { // state machine control
117         char readChar;
118         res = read(*fd,&readChar,1); // returns after 1 char
119         // input
120
121         if (!*flag && (res == 1)) {
122             *bufferP = readChar;
123
124             if (c > 3) {
125                 if ((*bufferP) == FLAG) {
126                     end = TRUE;
127                 }
128             }
129             c++;
130             bufferP++;
131         }
132         else if (*flag)
133             break;
134     }
135     //
136     //*****
137
138     if(*flag)
139         return -1;
140
141     return c;
142 }
143
144 //***** Function to configure the port and store the
145 //***** old configurations *****
146 void configure(linkLayer * ll, struct termios * oldtio) {
147     //Initialized variable to set the new config to the port
148     struct termios newtio;
149     if (ll->debug == TRUE) {
150         printf("\nConfiguration started!");
151     }

```

```

151 //Open the serial port
152 ll->fd = open((*ll).port, O_RDWR | O_NOCTTY | O_NONBLOCK );
153
154 //Check for errors of opening the port
155 if (ll->fd < 0) {
156     perror((*ll).port);
157     exit(-1);
158 }
159
160 if ( tcgetattr(ll->fd, oldtio) == -1) { // save current port
161     settings
162     perror("tcgetattr");
163     exit(-1);
164 }
165
166 if (ll->debug == TRUE) {
167     printf("\nOld config saved...");
168 }
169
170 //***** Set the new configuration of the port
171 //*****
172 bzero(&newtio, sizeof(newtio));
173 newtio.c_cflag = (*ll).baudRate | CS8 | CLOCAL | CREAD;
174 newtio.c_iflag = IGNPAR;
175 newtio.c_oflag = OPOST;
176 // set input mode (non-canonical, no echo,...)
177 newtio.c_lflag = 0;
178 newtio.c_cc[VTIME] = 0.1;
179 newtio.c_cc[VMIN] = 1; // blocking read until 1 chars
180 // received
181
182 if (ll->debug == TRUE) {
183     printf("\nSaving new config! ");
184 }
185
186 if ( tcsetattr(ll->fd, TCSANOW, &newtio) == -1)
187 {
188     perror("tcsetattr");
189     exit(-1);
190 }
191 //
192 //*****
193
194 if (ll->debug == TRUE) {
195     printf("\nConfiguration set");
196 }
197 }
198 //
199 //*****
200
201 //***** Reset the serial port configuration
202 //*****
203 void resetConfiguration(int * fd, struct termios * oldtio) {

```

```

199     //Sleep before resetting the configuration to prevent errors
        in communication
200     sleep(1);
201     if ( tcsetattr(*fd,TCSANOW,oldtio) == -1)
202     {
203         perror("tcsetattr");
204         exit(-1);
205     }
206
207     close(*fd);
208 }
209 //
        *****

210
211 //***** Function to trigger alarm *****
212 void triggerAlarm() {
213     *flagPointer = TRUE;
214     *countPointer = *countPointer + 1;
215     *timeoutAlarm = *timeoutAlarm + 1;
216
217     //printf("\nTimeout Expired");
218
219 }
220 //*****
221
222 int ll_open(int * flag, int * stop, int * count, linkLayer * ll
        , struct termios * oldtio) {
223
224     ll->debug = FALSE;
225
226     if (ll->debug == TRUE) {
227         printf("\n
                -----\n
                nStarted ll_open()");
228     }
229
230     configure(ll, oldtio);
231
232     flagPointer = flag;
233     countPointer = count;
234     timeoutAlarm = &(ll->stat->timeouts);
235
236     //***** While cycle to control the sending of the
        message *****
237     if (ll->status == 'W') {
238         if (ll->debug == TRUE) {
239             printf("\nThis is the sender...");
240         }
241
242         tcflush(ll->fd, TCIFLUSH);
243         while(*count < (*ll).numTransmissions) {
244
245             if(&flag) {
246                 alarm((*ll).timeout);
247

```

```

248         //printf("\nAttempts remaining: %d ", (ATTEMPTS
           - *count - 1));
249
250         tcflush(ll->fd, TCOFLUSH);
251
252         writeMsg(&(ll->fd), A_1, C_SET);
253         tcflush(ll->fd, TCIFLUSH);
254         *flag = FALSE;
255         if (ll->debug == TRUE) {
256             printf("\nwaiting...");
257         }
258         if(readResponse(&(ll->fd), flag, A_1, C_UA) ==
           0) {
259             if (ll->debug == TRUE) {
260                 printf("\n-> Response received!");
261             }
262             break;
263         }
264     }
265 }
266 //
           *****

267     if (*count == (*ll).numTransmissions)
268         return -1;
269 }
270 else if (ll->status == 'R') {
271     if (ll->debug == TRUE) {
272         printf("\nThis is the receiver");
273     }
274     int flagT = FALSE;
275     tcflush(ll->fd, TCIFLUSH);
276     while (readResponse(&ll->fd, &flagT, A_1, C_SET) != 0)
277         { continue; }
278     writeMsg(&ll->fd, A_1, C_UA);
279 }
280 if (ll->debug == TRUE) {
281     printf("\n
           -----\n
           nFinished ll_open()");
282 }
283 return 0;
284 }
285
286 int ll_close(int * flag, int * stop, int * count, linkLayer *
           ll, struct termios * oldtio) {
287
288     if (ll->debug == TRUE) {
289         printf("\n
           -----\n
           nStarted ll_close()");
290     }
291
292     flagPointer = flag;
293     countPointer = count;

```

```

294
295 *flagPointer = FALSE;
296
297 //***** While cycle to control the sending of the
      message *****
298 if (ll->status == 'W') {
299     if (ll->debug == TRUE) {
300         printf("\nWill send DISC...");
301     }
302
303     tcflush(ll->fd, TCIFLUSH);
304     while(*count < (*ll).numTransmissions) {
305
306         if(&flag) {
307             alarm((*ll).timeout);
308
309             //printf("\nAttempts remaining: %d ", (ATTEMPTS
              - *count - 1));
310             writeMsg(&ll->fd, A_1, C_DISC);
311             tcflush(ll->fd, TCIFLUSH);
312             *flag = FALSE;
313             if (ll->debug == TRUE) {
314                 printf("\nwaiting...");
315             }
316             if(readResponse(&ll->fd, flag, A_2, C_DISC) ==
              0) {
317                 if (ll->debug == TRUE) {
318                     printf("\n-> Response received!");
319                 }
320                 break;
321             }
322         }
323     }
324     if (*count == (*ll).numTransmissions)
325         return -1;
326
327     if (ll->debug == TRUE) {
328         printf("\nSending last message (C-UA)");
329     }
330     writeMsg(&ll->fd, A_2, C-UA);
331     //
      *****
332 }
333 else if (ll->status == 'R') {
334     if (ll->debug == TRUE) {
335         printf("\nWill receive DISC and respond the same\n");
336     }
337     int flagT = FALSE;
338     tcflush(ll->fd, TCIFLUSH);
339     while (readResponse(&(ll->fd), &flagT, A_1, C_DISC) !=
      0) { continue; }
340     writeMsg(&(ll->fd), A_2, C_DISC);
341     while (readResponse(&(ll->fd), &flagT, A_2, C-UA) != 0)
      { continue; }
342 }

```

```

343
344     resetConfiguration(&ll->fd, oldtio);
345
346     if (ll->debug == TRUE) {
347         printf("\n
-----\n
nFinished ll_close()");
348     }
349 }
350
351 int llread(linkLayer * ll, char ** buffer) {
352     int flagT = FALSE;
353
354     char * buffer_2 = malloc(sizeof(char) * (((*ll).packSize -
        6) * 2) + 16));
355
356     int nRead = -1;
357     while (nRead < 0) {
358         nRead = readInfo(&ll->fd, &flagT, buffer_2);
359     }
360
361     int sizeOfInfoRead = deStuff(buffer_2, buffer, ll);
362
363     if ((*buffer + 2) == C_0) {
364         if (ll->debug == TRUE) {
365             printf("\nSequence number received was: 0");
366         }
367     }
368     else if ((*buffer + 2) == C_1) {
369         if (ll->debug == TRUE) {
370             printf("\nSequence number received was: 1");
371         }
372     }
373
374     if (ll->debug == TRUE) {
375         printf(" | Sequence number asked was: %d", (*ll).
            sequenceNumber);
376
377         printf("\nA: 0x%x", (*buffer + 1));
378         printf("\nC: 0x%x", (*buffer + 2));
379         printf("\nBcc1: 0x%x", (*buffer + 3));
380         printf("\nBcc2: 0x%x", (*buffer + sizeOfInfoRead - 2));
381     }
382
383     int length = sizeOfInfoRead - 6;
384     int q = 4;
385     char bcc2 = 0x0;
386     while (q < (length + 4)) {
387         bcc2 = bcc2 ^ (*buffer + q);
388         q++;
389     }
390
391     if (ll->debug == TRUE) {
392         printf("\nBcc2-2: 0x%x", bcc2);
393     }
394

```



```

395     if ((*buffer + 3) != ((*buffer + 1) ^ (*buffer + 2)))
396         || ((*buffer + sizeofInfoRead - 2) != bcc2)) {
397         if ((*ll).sequenceNumber == 0) {
398             if (ll->debug == TRUE) {
399                 printf("\n****\n2nd ERROR receiving 1, wanted 0,
400                     sending REJ 0\n****");
401             }
402             ll->stat->numSentREJ++;
403             writeMsg(&(ll->fd), A_1, C_REJ_0);
404             free(buffer_2);
405             return -5;
406         } else if ((*ll).sequenceNumber == 1) {
407             if (ll->debug == TRUE) {
408                 printf("\n****\n2nd ERROR receiving 0, wanted 1,
409                     sending REJ 1\n****");
410             }
411             ll->stat->numSentREJ++;
412             writeMsg(&(ll->fd), A_1, C_REJ_1);
413             free(buffer_2);
414             return -4;
415         }
416     }
417     else if ((*ll).sequenceNumber == 0 && (*buffer + 2) == C_0
418         ) {
419         writeMsg(&(ll->fd), A_1, C_RR_1);
420         ll->stat->numSentRR++;
421         (*ll).sequenceNumber = 1;
422     } else if ((*ll).sequenceNumber == 1 && (*buffer + 2) ==
423         C_1) {
424         writeMsg(&(ll->fd), A_1, C_RR_0);
425         ll->stat->numSentRR++;
426         (*ll).sequenceNumber = 0;
427     } else if ((*ll).sequenceNumber == 0 && (*buffer + 2) ==
428         C_1) {
429         writeMsg(&(ll->fd), A_1, C_REJ_0);
430         if (ll->debug == TRUE) {
431             printf("\n****\nERROR receiving 1, wanted 0, sending
432                 REJ 0\n****");
433         }
434         ll->stat->numSentREJ++;
435         free(buffer_2);
436         return -2;
437     } else if ((*ll).sequenceNumber == 1 && (*buffer + 2) ==
438         C_0) {
439         writeMsg(&(ll->fd), A_1, C_REJ_1);
440         if (ll->debug == TRUE) {
441             printf("\n****\nERROR receiving 0, wanted 1, sending
442                 REJ 1\n****");
443         }
444         ll->stat->numSentREJ++;
445         free(buffer_2);
446         return -3;
447     }
448 }
449 free(buffer_2);

```

```

442
443     sizeofInfoRead = removeFrameHeaderAndTrailer(buffer,
444         sizeofInfoRead);
445
446     return sizeofInfoRead;
447 }
448
449 int llwrite(int * stop, linkLayer * ll, char * buffer, int
450     length) {
451
452     //Fill the toSend char array
453     char * toSend = malloc(sizeof(char) * (length + 6));
454     *toSend = FLAG;
455     *(toSend + 1) = A_1;
456
457     if ((*ll).sequenceNumber == 0) {
458         *(toSend + 2) = C_0;
459     }
460     else {
461         *(toSend + 2) = C_1;
462     }
463
464     *(toSend + 3) = *(toSend + 1) ^ *(toSend + 2);
465
466     unsigned int j = 0;
467     while (j < length) {
468         *(toSend + j + 4) = *buffer;
469         buffer++;
470         j++;
471     }
472
473     *(toSend + length + 4) = 0x0;
474     int q = 4;
475     while (q < (length + 4)) {
476         *(toSend + length + 4) = *(toSend + length + 4) ^ *(
477             toSend + q);
478         q++;
479     }
480
481     *(toSend + length + 5) = FLAG;
482
483     if (ll->debug == TRUE) {
484         printf("\nA: 0x%x", *(toSend + 1));
485         printf("\nC: 0x%x", *(toSend + 2));
486         printf("\nBcc1: 0x%x", *(toSend + 3));
487         printf("\nBcc2: 0x%x", *(toSend + length + 4));
488     }
489
490     //Finished filling the char array
491
492     int bufSize = 0;
493
494     char *toSendStuffed = stuff(&toSend, length + 6, &bufSize);
495
496     *countPointer = 0;
497     //***** While cycle to control the sending of the

```

```

message *****
495 int rr = FALSE;
496 while(*countPointer < (*ll).numTransmissions && rr == FALSE
    ) {
497     if (ll->debug == TRUE) {
498         printf("\nSending Packet with seqNum: 0x%x", *(
            toSendStuffed + 2));
499     }
500     if(&flagPointer) {
501         alarm((*ll).timeout);
502         //printf("\nAttempts remaining: %d ", (ATTEMPTS - *
            countPointer - 1));
503
504         tcflush(ll->fd, TCOFLUSH);
505         write(ll->fd, toSendStuffed, bufSize); //Sending
            the info
506         //*****
507
508         *flagPointer = FALSE;
509         int resp = readSenderResponse(ll);
510         if(resp == 0) {
511             (*ll).sequenceNumber = 0;
512             if (ll->debug == TRUE) {
513                 printf("\n-> Received RR | Receiver asking
                    for packet 0");
514             }
515             ll->stat->numReceivedRR++;
516             rr = TRUE;
517         }
518         else if(resp == 1) {
519             (*ll).sequenceNumber = 1;
520             if (ll->debug == TRUE) {
521                 printf("\n-> Received RR | Receiver asking
                    for packet 1");
522             }
523             ll->stat->numReceivedRR++;
524             rr = TRUE;
525         }
526         else if (resp == -2) {
527             if (ll->debug == TRUE) {
528                 printf("\n-> Received REJ | Receiver asking
                    for packet 0");
529             }
530             ll->stat->numReceivedREJ++;
531         }
532         else if (resp == -3) {
533             if (ll->debug == TRUE) {
534                 printf("\n-> Received REJ | Receiver asking
                    for packet 1");
535             }
536             ll->stat->numReceivedREJ++;
537         } else {
538             if (ll->debug == TRUE) {
539                 printf("\nTIMEOUT - did not read response");
540             }
541         }
    }

```

```

542         if (ll->debug == TRUE) {
543             printf("\n");
544         }
545         usleep(0.5 * 1000000);
546         //*****
547     } else {
548         if (ll->debug == TRUE) {
549             printf("\nTIMEOUT expired");
550         }
551     }
552 }
553
554 if (*countPointer == (*ll).numTransmissions) {
555     free(toSendStuffed);
556     free(toSend);
557     return -1;
558 }
559 free(toSendStuffed);
560 free(toSend);
561 //
562     *****
563 }
564 //***** Read the response of the receiver
565     *****
566 int readSenderResponse(linkLayer * ll) {
567     int res;
568     char response[5];
569
570     //***** While that controls the reading of the
571         response of the receiver *****
572     unsigned int stateMachine = 0;
573     while (stateMachine < 5) { // state machine control
574         char readChar;
575         res = read(ll->fd,&readChar,1); // returns after 1 char
576             input
577
578         if (!*flagPointer && (res == 1)) {
579             switch (stateMachine) {
580                 case 0:
581                     if (readChar == FLAG) {
582                         response[stateMachine] = readChar;
583                         stateMachine = 1;
584                     }
585                     break;
586                 case 1:
587                     if (readChar == FLAG) {
588                         break;
589                     } else if (readChar == A_1) {
590                         response[stateMachine] = readChar;
591                         stateMachine = 2;
592                         break;
593                     } else {
594                         stateMachine = 0;

```

```

593         break;
594     }
595     case 2:
596     if (readChar == FLAG) {
597         stateMachine = 1;
598         break;
599     } else if ((readChar == C_REJ_0) || (readChar
600 == C_REJ_1) || (readChar == C_RR_0) || (
601 readChar == C_RR_1)) {
602         response[stateMachine] = readChar;
603         stateMachine = 3;
604         break;
605     } else {
606         stateMachine = 0;
607         break;
608     }
609     case 3:
610     if (readChar == FLAG) {
611         stateMachine = 1;
612         break;
613     } else if (readChar == (response[1]^response
614 [2])) {
615         response[stateMachine] = readChar;
616         stateMachine = 4;
617         break;
618     } else {
619         stateMachine = 0;
620         break;
621     }
622     case 4:
623     switch(readChar) {
624         case FLAG:
625             response[stateMachine] = readChar;
626             stateMachine = 5;
627             break;
628         default:
629             stateMachine = 0;
630             break;
631     }
632     }
633     else if (*flagPointer)
634         break;
635 }
636 //
637
638 if(*flagPointer)
639     return -1;
640
641 if (response[2] == C_REJ_0 && (stateMachine == 5))
642     return -2;
643 else if (response[2] == C_REJ_1 && (stateMachine == 5))

```

```

644     return -3;
645     else if (response[2] == C_RR_0 && (stateMachine == 5)) {
646         return 0;
647     }
648     else if (response[2] == C_RR_1 && (stateMachine == 5)) {
649         return 1;
650     } else {
651         return -1;
652     }
653 }
654
655 char * stuff(char ** deStuffed, int deStuffedLength, int *
        bufSize) {
656     int totalLength = ((deStuffedLength + 1) * 2) + 5;
657
658     char * toRet = malloc(sizeof(char) * totalLength);
659     int i = 4;
660     int j = 0;
661     while (j < 4) {
662         *(toRet + j) = *(*deStuffed + j);
663         j++;
664     }
665     while (i < (deStuffedLength - 1)) {
666         if ((*deStuffed + i) == FLAG) {
667             *(toRet + j) = ESCAPE;
668             j++;
669             *(toRet + j) = FLAG_EXC;
670         }
671         else if ((*deStuffed + i) == ESCAPE) {
672             *(toRet + j) = ESCAPE;
673             j++;
674             *(toRet + j) = ESCAPE_EXC;
675         }
676         else {
677             *(toRet + j) = *(*deStuffed + i);
678         }
679         i++;
680         j++;
681     }
682     *(toRet + j) = FLAG;
683     j++;
684
685     *bufSize = j;
686     return toRet;
687 }
688
689 int deStuff(char * deStuffed, char ** stuffed, linkLayer * ll)
        {
690
691     char * temp = malloc(sizeof(char) * (((*ll).packSize - 6)
        * 2) + 16));
692
693     int stuffedC = 0;
694     int end = FALSE;
695     while (end == FALSE) {
696

```

```

697         if (((*deStuffed) == ESCAPE) && ((*deStuffed + 1) ==
698             FLAG_EXC)) {
699             (*(temp + stuffedC)) = FLAG;
700             deStuffed++;
701         }
702         else if (((*deStuffed) == ESCAPE) && ((*deStuffed + 1)
703             ) == ESCAPE_EXC)) {
704             (*(temp + stuffedC)) = ESCAPE;
705             deStuffed++;
706         }
707         else {
708             (*(temp + stuffedC)) = (*deStuffed);
709         }
710
711         if ((stuffedC > 2) && ((*deStuffed + 1) == FLAG)) {
712             deStuffed++;
713             stuffedC++;
714             (*(temp + stuffedC)) = FLAG;
715             end = TRUE;
716         }
717
718         deStuffed++;
719         stuffedC++;
720     }
721
722     *stuffed = malloc(sizeof(char) * stuffedC);
723     int i = 0;
724     while (i < stuffedC) {
725         (*(stuffed + i)) = *(temp + i);
726         i++;
727     }
728     free(temp);
729
730     return stuffedC;
731 }
732
733 int removeFrameHeaderAndTrailer(char ** buffer, int
734     sizeofInfoRead) {
735     char * finalB = malloc(sizeof(char) * (sizeofInfoRead - 6))
736         ;
737
738     int counter = 4;
739     int i = 0;
740     while (i < sizeofInfoRead - 6) {
741         (*(finalB + i)) = (*(buffer + counter));
742         i++;
743         counter++;
744     }
745
746     free(*buffer);
747     *buffer = finalB;
748
749     return sizeofInfoRead - 6;
750 }

```

```

1  #include "utilities.h"
2
3  int getFileSize(FILE* file) {
4      long int currentPosition = ftell(file);
5
6      if (fseek(file, 0, SEEK_END) == -1) {
7          printf("ERROR: Could not get file size.\n");
8          return -1;
9      }
10
11     long int size = ftell(file);
12
13     fseek(file, 0, currentPosition);
14
15     return size;
16 }
17
18 char * getLine(void) {
19     char * line = malloc(100), * linep = line;
20     size_t lenmax = 100, len = lenmax;
21     int c;
22
23     if(line == NULL)
24         return NULL;
25
26     for(;;) {
27         c = fgetc(stdin);
28         if(c == EOF)
29             break;
30
31         if(--len == 0) {
32             len = lenmax;
33             char * linen = realloc(linep, lenmax * 2);
34
35             if(linen == NULL) {
36                 free(linep);
37                 return NULL;
38             }
39             line = linen + (line - linep);
40             linep = linen;
41         }
42
43         if((*line++ = c) == '\n')
44             break;
45     }
46     *(line-1) = '\0';
47     return linep;
48 }

```

#### Anexo 8 - utilities.c

```

1  #include "cli.h"
2
3  int bRate;
4
5  void clearScreen() {
6      printf("\033[2J");

```



```

7  }
8
9  int initialMenu() {
10     clearScreen();
11
12     printf("\n\n\n"
13         "*****\n
14         "
15         "***** RCOM project - 1 *****\n
16         "
17         "*****\n
18         "
19         "**                                     **\n
20         "
21         "                Choose Which side of the program:                **\n
22         "
23         "                1 - Transmitter                **\n
24         "
25         "                2 - Receiver                **\n
26         "
27         "                **\n
28         "
29         "*****\n
30         "
31         "\n\n");
32
33     int choice = -1;
34
35     scanf("%d", &choice);
36
37     flushIn();
38
39     if (choice == 1 || choice == 2) {
40         return choice;
41     }
42
43     return -1;
44 }
45
46 int choosePort() {
47     clearScreen();
48
49     printf("\n\n\n"
50         "*****\n
51         "
52         "***** RCOM project - 1 *****\n
53         "
54         "*****\n
55         "
56         "**                                     **\n
57         "
58         "                Choose the port you want to use:                **\n
59         "
60         "                **\n

```

```

48     "
    "**          1 - /dev/ttyS0          **\n
    "
49     "**          2 - /dev/ttyS4          **\n
    "
50     "**          **\n
    "
51     "*****\n
    "
52     "\n\n");
53
54     int choice = -1;
55
56     scanf("%d", &choice);
57
58     flushIn();
59
60     if (choice == 1 || choice == 2) {
61         return choice;
62     }
63
64     return -1;
65 }
66
67 int chooseBaudrate() {
68     clearScreen();
69
70     printf("\n\n\n"
71         "*****\n
72         "
73         "***** RCOM project - 1 *****\n
74         "
75         "*****\n
76         "
77         "**          **\n
78         "
79         "**          Choose the Baudrate you want to use:  **\n
80         "
81         "**          **\n
82         "
83         "**          1 - B300          **\n
84         "
85         "**          2 - B600          **\n
86         "
87         "**          3 - B1200         **\n
88         "
89         "**          4 - B1800         **\n
90         "
91         "**          5 - B2400         **\n
92         "
93         "**          6 - B4800         **\n
94         "
95         "**          7 - B9600         **\n
96         "
97         "**          8 - B19200        **\n
98         "

```

```

85     "**          9 - B38400          **\n
      "
86     "**          10 - B57600         **\n
      "
87     "**          11 - B115200        **\n
      "
88     "**                                     **\n
      "
89     "*****\n
      "
90     "\n\n");
91
92     int choice = -1;
93
94     scanf("%d", &choice);
95
96     flushIn();
97
98     if (choice >= 1 && choice <= 11) {
99         switch(choice) {
100             case 1:
101                 bRate = 300;
102                 break;
103             case 2:
104                 bRate = 600;
105                 break;
106             case 3:
107                 bRate = 1200;
108                 break;
109             case 4:
110                 bRate = 1800;
111                 break;
112             case 5:
113                 bRate = 2400;
114                 break;
115             case 6:
116                 bRate = 4800;
117                 break;
118             case 7:
119                 bRate = 9600;
120                 break;
121             case 8:
122                 bRate = 19200;
123                 break;
124             case 9:
125                 bRate = 38400;
126                 break;
127             case 10:
128                 bRate = 57600;
129                 break;
130             case 11:
131                 bRate = 115200;
132                 break;
133             default:
134                 printf("\nError in choice of baudrate\n");
135                 return -1;

```

```

136     }
137     return choice;
138 }
139
140 return -1;
141 }
142
143 int chooseMaxSize() {
144     clearScreen();
145
146     printf("\n\n\n"
147         "*****\n"
148         "***** RCOM project - 1 *****\n"
149         "*****\n"
150         "**\n"
151         "**      Choose the Maximum size of bytes/packet:      **\n"
152         "**\n"
153         "**                      [11 - 2000]                      **\n"
154         "**\n"
155         "*****\n"
156         "\n\n");
157
158     int choice = -1;
159
160     scanf("%d", &choice);
161
162     flushIn();
163
164     if (choice >= 11 && choice <= 2000) {
165         return choice;
166     }
167
168     return -1;
169 }
170
171 int chooseTimeout() {
172     clearScreen();
173
174     printf("\n\n\n"
175         "*****\n"
176         "***** RCOM project - 1 *****\n"
177         "*****\n"
178         "**\n"

```

```

179     "**          Choose the timeout in seconds:          **\n
180     "
181     "**          [1 - 60]          **\n
182     "
183     "**          0 - default value          **\n
184     "
185     "*****\n
186     "\n\n");
187
188     int choice = -1;
189
190     scanf("%d", &choice);
191
192     if (choice == 0) choice = 3;
193
194     flushIn();
195
196     if (choice >= 1 && choice <= 60) {
197         return choice;
198     }
199
200     return -1;
201 }
202
203 int chooseNumTransmissions() {
204     clearScreen();
205
206     printf("\n\n\n"
207     "*****\n
208     " ***** RCOM project - 1 *****\n
209     "
210     " *****\n
211     "
212     "**          Choose the number of attempts:          **\n
213     "
214     "**          [1 - 60]          **\n
215     "
216     "**          0 - default value          **\n
217     "
218     "*****\n
219     "\n\n");

```

```

218
219     int choice = -1;
220
221     scanf("%d", &choice);
222
223     flushIn();
224
225     if (choice >= 1 && choice <= 60) {
226         return choice;
227     }
228
229     return -1;
230 }
231
232 void showInitialInfo(linkLayer * ll, applicationLayer * al) {
233     clearScreen();
234
235     char modeW[] = "Transmitter\0";
236     char modeR[] = "Receiver\0";
237     int s = -1;
238
239     if ((*al).status == 'W') {
240         s = 0;
241     } else {
242         s = 1;
243     }
244
245     if (s == 0) {
246
247         printf("\n\n\n"
248             "*****\n"
249             "***** RCOM project - 1 *****\n"
250             "*****\n"
251             "                                \n"
252             "          # Initial Information #          \n"
253             "                                \n"
254             "                                \n"
255             "                Mode: %s\n"
256             "                Baud rate: B%d\n"
257             "                Msg Max Size: %d\n"
258             "                Attempts: %d\n"
259             "                Timeout: %d\n\n"
260             "*****\n"
261             "\n\n", modeW, bRate, (*ll).packSize, (*ll).
                numTransmissions, (*ll).timeout);
262
263     } else {
264         printf("\n\n\n"
265             "*****\n"
266             "***** RCOM project - 1 *****\n"
267             "*****\n"
268             "                                \n"
269             "          # Initial Information #          \n"
270             "                                \n"
271             "                                \n"
272             "                Mode: %s\n"

```

```

273     "        Baud rate: B%d\n"
274     "        Msg Max Size: %d\n"
275     "        Attempts: %d\n"
276     "        Timeout: %d\n\n"
277     "*****\n"
278     "\n\n", modeR, bRate, (*ll).packSize, (*ll).
        numTransmissions, (*ll).timeout);
279
280 }
281
282 return;
283 }
284
285 void flushIn() {
286     char ch;
287     while ((ch = getchar()) != '\n' && ch != EOF);
288 }
289
290 const int PROGRESS_BAR_LENGTH = 51;
291
292 void printProgressBar(int current, int total) {
293     float percentage = 100.0 * current / (float) total;
294
295     if (percentage > 100.0)
296         percentage = 100.0;
297
298     printf("\rCompleted: %6.2f%% [" , percentage);
299     int i;
300     int len = PROGRESS_BAR_LENGTH;
301     int pos = percentage * len / 100.0;
302     for (i = 0; i < len; i++)
303         i <= pos ? printf("=") : printf(" ");
304     printf("]");
305     fflush(stdout);
306 }
307
308 void printStats(applicationLayer * al, Statistics * stats){
309
310     int numReceivedRR;
311     int numReceivedREJ;
312
313     if ((*al).status == 'W') {
314         printf("\n"
315             "
316             # Final Statistics #
317             "
318             "
319             Sent Messages: %d\n"
320             Received RR: %d\n"
321             Received REJ: %d\n\n"
322             "
323             Total Timeouts: %d\n\n"
324             "*****\n"
325             "\n", stats->sentMessages, stats->numReceivedRR, stats->
        numReceivedREJ, stats->timeouts);

```

```

327     } else {
328         printf("\n"
329             "                                \n"
330             "                # Final Statistics #        \n"
331             "                                \n"
332             "                                \n"
333             "        Received Messages: %d\n"
334             "                Sent RR: %d\n"
335             "                Sent REJ: %d\n\n"
336             "*****\n"
337             "\n", stats->receivedMessages, stats->numSentRR, stats->
                numSentREJ);
338     }
339
340
341     return;
342 }
343
344 void chooseFileName(char ** fileName) {
345     clearScreen();
346
347     printf("\n\n\n"
348         "
349         *****\n"
350         "***** RCOM project - 1\n"
351         "*****\n"
352         "*****\n"
353         "*****\n"
354         "
355         Name is: ");
356
357     *fileName = getLine();
358     printf("\n\n");
359     return;
360 }

```

Anexo 9 - cli.c