

Redes de Computadores

# Protocolo de Ligação de Dados

Ângela Cardoso e Bruno Madeira



5 de Novembro de 2015

# Sumário

Relatório relativo ao primeiro trabalho prático da unidade curricular Redes de Computadores do curso Mestrado Integrado em Engenharia Informática e Computação que consiste na implementação de uma aplicação que transfere imagens entre dois computadores fazendo uso da porta-série. O principal objectivo do trabalho é colocar em prática alguns dos conceitos leccionados na cadeira relativos a protocolos de ligação de dados.

O relatório apresenta o estado final do projecto e as considerações dos estudantes responsáveis pela sua implementação face ao resultado obtido.

# Conteúdo

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introdução</b>                                      | <b>3</b>  |
| <b>2</b> | <b>Arquitetura, Estrutura de Código e Casos de uso</b> | <b>4</b>  |
| 2.1      | Arquitetura . . . . .                                  | 4         |
| 2.2      | Estrutura do código . . . . .                          | 4         |
| 2.2.1    | Estruturas de dados . . . . .                          | 4         |
| 2.2.2    | Funções privadas da DataLinkProtocol . . . . .         | 4         |
| 2.2.3    | Funções privadas da AppProtocol.c . . . . .            | 5         |
| 2.2.4    | Funções disponibilizadas por FileFuncs.h . . . . .     | 5         |
| 2.2.5    | Funções da App.c . . . . .                             | 5         |
| 2.3      | Casos de Uso . . . . .                                 | 5         |
| <b>3</b> | <b>Protocolo de ligação lógica</b>                     | <b>6</b>  |
| <b>4</b> | <b>Protocolo de aplicação</b>                          | <b>7</b>  |
| <b>5</b> | <b>Elementos de valorização</b>                        | <b>8</b>  |
| 5.1      | Registo de ocorrências . . . . .                       | 8         |
| 5.2      | Definições básicas . . . . .                           | 8         |
| 5.3      | Gerador aleatório de erros e REJs . . . . .            | 8         |
| 5.4      | Representação do progresso . . . . .                   | 8         |
| <b>6</b> | <b>Validação</b>                                       | <b>9</b>  |
| <b>7</b> | <b>Conclusões</b>                                      | <b>10</b> |
| <b>8</b> | <b>Bibliografia</b>                                    | <b>11</b> |
|          | <b>Appendices</b>                                      | <b>12</b> |
| <b>A</b> | <b>Código Fonte</b>                                    | <b>13</b> |
| A.1      | App.c . . . . .  | 13        |
| A.2      | AppProtocol.c . . . . .                                | 18        |
| A.3      | AppProtocol.h . . . . .                                | 23        |
| A.4      | DataLinkProtocol.c . . . . .                           | 24        |
| A.5      | DataLinkProtocol.h . . . . .                           | 43        |
| A.6      | FileFuncs.c . . . . .                                  | 44        |
| A.7      | FileFuncs.h . . . . .                                  | 45        |
| A.8      | user_interface.c . . . . .                             | 45        |
| A.9      | user_interface.h . . . . .                             | 48        |
| A.10     | Utilities.h . . . . .                                  | 48        |

|          |   |           |
|----------|---|-----------|
| <b>B</b> | <b>Tipos de Tramas Usadas</b>                 | <b>50</b> |
| <b>C</b> | <b>Diagrama de chamadas a funções (Fluxo)</b> | <b>51</b> |
| <b>D</b> | <b>Diagrama de Módulos</b>                    | <b>53</b> |

# 1 Introdução

Este relatório incide sobre projecto desenvolvido para o primeiro trabalho prático de Redes de Computadores que consiste na implementação de uma aplicação que transfere imagens entre dois computadores fazendo uso da porta-série. A aplicação deve usar um protocolo de ligação de dados *Stop N Wait ARQ* híbrido que deve assegurar a fiabilidade da transmissão mesmo em caso de desconexão. Deve também usar um protocolo de aplicação que é responsável pelo envio da imagem. O código desenvolvido deve ser estruturado em camadas, respeitando o princípio de encapsulamento, de modo a assegurar cada protocolo funciona de forma independente.

O trabalho foi utilizando a linguagem de programação C num ambiente com um sistema operativo baseado em Linux. O código pode ser verificado em anexo e será referenciado em algumas secções do relatório.

Este relatório tem como objectivo reportar qual o estado final da aplicação desenvolvia, clarificar detalhes do processo de implementação/código e a opinião dos estudantes face ao projecto realizado.

Do capítulo 2 ao 4 são expostas as estruturas e os mecanismos implementados na concepção da aplicação sem incidir em detalhes específicos destes. Detalhes relativos à implementação dos protocolos são apresentados nos capítulos 5 e 6 e detalhes relativos à implementação de componentes extra são apresentados no capítulo 7. O capítulo 8 é relativo à validação e testes efectuados.

# 2 Arquitetura, Estrutura de Código e Casos de uso

## 2.1 Arquitectura

Cada módulo desenvolvido no projecto pode ser identificado pelos headers files do código fonte que são:

- `DataLinkProtocol` implementa e disponibiliza funcionalidade da camada de ligação de dados.
- `AppProtocol` implementa e disponibiliza funcionalidades da camada de aplicação relacionadas com o envio/recepção de pacotes.
- `user_interface` implementa segmentos e disponibiliza funcionalidades ligadas ao interface de utilizador.
- `FileFuncs` disponibiliza funções para leitura e escrita de ficheiros.
- `App.c` onde está a função `main` e são chamadas as principais funções de outro módulos em conjunto com as operações adicionais necessárias.
- `utilities.h` foi criada com o intuito de ter métodos, estruturas e funcionalidades úteis a todos os módulos desenvolvidos. O seu uso principal no projecto é auxiliar o debug dos diversos modelos desenvolvidos.

Pode verificar-se no diagrama de módulos, disponível no anexo D, as relações entre estes módulos.

## 2.2 Estrutura do código

Seguidamente são apresentadas as principais estruturas e funções desenvolvidas em cada módulo sendo algumas ocultadas uma vez que são referidas em maior detalhe nos capítulos relativos aos protocolos implementados.

### 2.2.1 Estruturas de dados

- `applicationLayer`: declarada na `App.c` contem informações básicas ao programa como o seu status (transmissor/receptor) por exemplo.
- `linkLayer`: declarada no `DataLinkProtocol.c` serve para guardar as definições básicas da camada de ligação de dados como qual a porta série a ser usada.
- `occurrences\Log`: declarada no `DataLinkProtocol.h` serve para registar ocorrências como o número de REJs recebidos.

### 2.2.2 Funções privadas da `DataLinkProtocol`

- `genBCC2` e `validateBCC2` tratam respectivamente de gerar e de validar o BCC2.
- `write_UorS` e `write_I` responsáveis pelo envio de tramas.

- `apply_stuffing` e `apply_destuffing` realizam o stuffing e destuffing dos dados nas tramas I.
- `update_state_machine` função auxiliar que funciona como máquina de estados.
- `llopen_receiver`, `llopen_transmitter`, `llclose_receiver` e `llclose_transmitter` ajudam a organizar o código do `llopen` e `llclose` uma vez que a sua execução difere do receptor para o emissor.
- `startAlarm` e `stopAlarm` gerem o uso do alarme em conjunto com as funções invocadoras.

### 2.2.3 Funções privadas da `AppProtocol.c`

- `receivePacket` trata da recepção de um único pacote que pode ser de controlo ou de dados.
- `getControlPacket` e `getInfoPacket` responsáveis, respectivamente, pela criação de um único pacote de controlo e de dados.
- `sendControlPacket` e `sendInfoPacket` responsáveis, respectivamente, pela transmissão de um único pacote de controlo e de dados.
- `show_progress` responsável por mostrar ao utilizador o estado actual da transferência/recepção do ficheiro.

### 2.2.4 Funções disponibilizadas por `FileFuncs.h`

- `getFileBytes` e `save2File` são responsáveis respectivamente pela escrita e leitura dos ficheiros.

### 2.2.5 Funções da `App.c`

- `receiveImage` e `sendImage` chamam as funções do Protocolo de Aplicação com os ajustes necessários de modo a enviar/receber uma imagem.
- `config` quando o utilizador termina a selecção de opções no interface esta função transmite-as para o Protocolo de Ligação de Dados.

## 2.3 Casos de Uso

A aplicação desenvolvida deve ser chamada na linha de comandos recebendo como argumentos a porta de série a usar (`/dev/ttyS0` ou `/dev/ttyS4`) e um caracter indicador(`t` ou `r`) se a aplicação deve correr em modo transmissor ou receptor.

Uma vez invocado o programa será mostrado um menu ao utilizador. No transmissor pode escolher seleccionar uma imagem a enviar ou enviar uma que já tenha seleccionado. Do lado do receptor pode ser iniciada a recepção. Ambas as versões da aplicação têm um submenu de para seleccionar opções, estas serão abordadas numa outra secção do relatório.

As funções invocadas no seguimento das escolhas do utilizador podem ser averiguadas no diagrama de chamada de funções que pode ser consultado no anexo C.

# 3 Protocolo de ligação lógica

Para implementar o protocolo de ligação lógica, seguimos as indicações do enunciado do projeto. Sendo assim, usamos a variante *Stop and Wait*, o que significa que o Emissor, após cada mensagem, aguarda uma resposta do Recetor antes de enviar a mensagem seguinte. Isto significa, entre outras coisas, que podemos utilizar uma numeração módulo 2 para as mensagens, dado que nunca temos mais do que 2 mensagens em jogo (aquela que foi enviada e a que se pretende enviar de seguida).

O interface deste protocolo disponibiliza 6 funções.

Quatro são as definidas pelo guião do trabalho (`llopen`, `llread`, `llwrite`, `llclose`) sendo que as únicas alterações efectuadas foram relativas à assinatura e funcionamento do `llclose` de modo a permitir fechar a porta-série em caso de erro e no `llopen` que não recebe a porta a abrir.

Estas 4 funções utilizam a função `update_state_machine` para determinar a cada instante o estado em que está. Com a concepção desta função evitou-se alguma repetição de código mas foi necessário cuidado extra na implementação pois são necessárias trocas de informação entre esta e a função que a invoca. A `update_state_machine` recebe qual o tipo de trama esperada que serve para identificar a função que a invocou e atualizar os estados corretamente e guarda numa variável auxiliar, `received_C_type`, o tipo de trama recebida. Quando a função invocadora detecta que se atingiu o estado `STATE_MACHINE_STOP` verifica o `received_C_type` para saber tipo de trama recebido e como deve proceder.

Para obter a trama que se pretende enviar são usadas as funções `write_UorS` e `write_I` sendo estas respectivamente para o envio de tramas de Supervisão ou Não numeradas e para tramas de Informação. Elas fazem uso da função `getMessage` que tem como parâmetros o Emissor original, o tipo e o número da trama e o array de caracteres onde será colocada a flag e o cabeçalho da trama. Posteriormente a estes bytes, em tramas de Informação, serão adicionados os dados e o respetivo BCC2. Nas restantes tramas, é apenas reutilizada a flag que está na primeira posição do array. A sua especificação pode ser encontrada no Anexo B.

As outras 2 funções disponibilizadas pelo interface são `set_basic_definitions` e `get_occurrences_log` que servem respectivamente para guardar as opções escolhidas pelo utilizador e para receber o registo de ocorrências na camada da aplicação.



## 4 Protocolo de aplicação

O protocolo de aplicação foi implementado no ficheiro `AppProtocol.c`. e apenas disponibiliza as funções `sendFile` e `receiveFile` para módulos externos. As duas funções são responsáveis pelo envio e recepção de uma imagem completa, sendo que a recepção e envio de pacotes individuais é tratada dentro de `AppProtocol.c`.

O protocolo foi implementado de acordo com o enunciado e é responsável pelo envio de pacotes de controlo e de dados. Os pacotes de controlo são enviados antes e após o envio dos dados e têm como parâmetros nome e tamanho (em bytes) da imagem. Quando a informação destes pacotes coincide não é realizado o output da imagem mesmo que todos os pacotes de dados tenham sido recebidos dado que não há garantias que pertençam todos à mesma imagem ou que todos foram recebidos.

O envio/recepção de imagens atualiza uma variável, recebida por um apontador, externa ao Protocolo (definida num outro módulo), que indica os bytes já enviados/recebidos. Tal variável é usada atualmente dentro do protocolo para o display do estado de envio da imagem que inicialmente estava implementado no ficheiro `user_interface.c`. Esta variável serviria também para tentar retransmitir uma imagem em caso de um envio falhar a meio. Este mecanismo de retransmissão não foi completado e o display foi movido pois a sua implementação inicial interferia com o envio/recepção de dados, a estrutura do código actual reflecte os planos inicialmente concebidos.

# 5 Elementos de valorização

## 5.1 Registo de ocorrências

A camada de ligação de dados regista o número de tramas do tipo I e REJs recebidas/enviadas e o número de ocorrências de timeouts de uma transmissão. A informação fica registada numa variável do tipo `occurrences_Log` e pode ser acedida pela componente de interface através do método `get_occurrences_log`.

## 5.2 Definições básicas

O utilizador pode definir qual o `baudrate` a usar, o `packetSize` (numero de bytes máximos que envia da imagem em cada trama I antes de se realizar o stuffing) e o número máximo de tentativas consecutivas de reconexão. Uma vez escolhidas as opções o intervalo de duração de um timeout é calculado em função do `baudrate` e do `packetSize` escolhidos pelo utilizador como se pode verificar no anexo A.4 nas linhas 187 a 209. Na nossa implementação o receptor pode e deve configurar o `baudrate` e `packetSize` iguais aos do transmissor, pois, apesar de estes não serem usados na prática pelo receptor, são necessários para calcular um intervalo de timeout que seja fiável face às definições do transmissor.

## 5.3 Gerador aleatório de erros e REJs

São gerados erros aleatoriamente nas tramas enviadas pelo transmissor. São gerados erros no cabeçalho e de nos dados. Estes ocorrem de forma independente e são gerados respectivamente pelas funções `randomErrorCabe` e `randomErrorData`. A frequência destes erros é definida a partir das constantes `CABE_PROB` e `DATA_PROB` que representam quantas tramas são enviadas por cada um que apresenta um erro do tipo correspondente. Pode ser verificada a função `randomErrorData` no anexo A.4 nas linhas 517 a 526. A `randomErrorCabe` está no mesmo anexo e é muito semelhante. Tramas do tipo REJ foram implementadas. Quando o receptor consegue identificar ocorrência de erro(s) no bloco de dados envia uma mensagem ao transmissor e este trata de reenviar a pacote novamente.

## 5.4 Representação do progresso

Na `AppProtocol.c`, quando a aplicação está enviar/receber uma imagem, trata imprimir na consola a quantidade de bytes enviados/recebidos e quantos faltam para receber/enviar a imagem na totalidade. Este display (impressão na consola) é actualizado cada vez que um pacote é enviado/recebido na totalidade se já tiver passado um tempo mínimo definido na constante `UPDATE_DISPLAY_MIN_TIME_INTERVAL` desde a ultima actualização de display de modo a evitar que ocorra com elevada frequência o que poderia tornar difícil a sua leitura ou afectar a

## 6 Validação

Foram realizados cerca de 10 testes ao longo das aulas práticas e fora destas usando uma porta-série. Foram realizados cerca de 30 testes nas máquinas virtuais usadas para desenvolver a aplicação no fim desta estar terminada.

Relativamente à qualidade dos testes não foi usado nenhum auxiliar para verificar se as imagens recebidas eram iguais às enviadas nem nenhuma automatização de testes, sendo estes todos realizados manualmente. Foram usadas imagens de diferentes tamanhos variados entre os 263 bytes (a mais pequena) e 712K bytes (a maior). Os tipos de imagens também variavam entre os tipos gif, jpeg e png. De todos os testes realizados foram bem sucedidos à exceção de 1 que apresentou um aviso inesperado e do qual não conseguimos apurar o problema.

## 7 Conclusões

Foi implementada com sucesso dentro do prazo estabelecido uma aplicação capaz de transferir imagens entre 2 computadores através de uma porta-série. A implementação foi realizada em camadas como sugerido e foram percebidos os conceitos e implicações práticas dos protocolos usados.

## 8 Bibliografia

# Anexos

# A Código Fonte

## A.1 App.c

```
1
2 #include <sys/types.h>
3 #include <sys/stat.h>
4 #include <fcntl.h>
5 #include <termios.h>
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <string.h>
9 #include <unistd.h>
10 #include <pthread.h>
11
12 #include "utilities.h"
13 #include "user_interface.h"
14 #include "DataLinkProtocol.h"
15 #include "FileFuncs.h"
16 #include "AppProtocol.h"
17
18 #define BAUDRATE B38400 //default baudrate
19 #define _POSIX_SOURCE 1 // POSIX compliant source
20
21
22 //
23 // =====
24 // PROGRAM VARIABLES
25 // =====
26
27 struct applicationLayer {
28     int fd; /*Descriptor correspondente a porta serie*/
29     int status; /*TRANSMITTER 0 | RECEIVER 1*/
30     unsigned char l2; /* info packet size = 12 * 256 + 11 */
31     unsigned char l1; /* defaults are L2 and L1 defined in
32                        AppProtocol.h */
33 };
34
35 struct applicationLayer app;
36
37 occurrences_Log_Ptr datalink_log;
38
39 bool conection_open = FALSE;
40
41 //pthread_t display_thread;
42 bool show_display;
43
44 //bool image_loaded = NO; //check with image bytes length nstead
45 unsigned int image_already_bytes; //num of image's bytes sent or
46 received
47 char* image_bytes;
48 unsigned int image_bytes_length;
49 char image_name[255]; //name is not path!!!
50 unsigned char image_name_length = 0;
```

```

48
49 //
=====

50 // PROGRAM FUNCS
51 //
=====

52
53 int connect()
54 {
55
56     if ((app.fd = llopen(app.status)) < 0) return -1;
57
58     return 0;
59 }
60
61
62 void setPacketSize(int packetSize) {
63     app.l2 = (unsigned char) (packetSize / 256);
64     // printf("l2: %u\n", app.l2);
65     app.l1 = (unsigned char) (packetSize % 256);
66     // printf("l1: %u\n", app.l1);
67 }
68
69 void config(char baud, char recon, char timeo, int packetSize)
70 {
71     int baudrate = -1;
72     switch (baud) {
73     case 'a':
74         baudrate = B0; break;
75     case 'b':
76         baudrate = B50; break;
77     case 'c':
78         baudrate = B75; break;
79     case 'd':
80         baudrate = B110; break;
81     case 'e':
82         baudrate = B134; break;
83     case 'f':
84         baudrate = B150; break;
85     case 'g':
86         baudrate = B200; break;
87     case 'h':
88         baudrate = B300; break;
89     case 'i':
90         baudrate = B600; break;
91     case 'j':
92         baudrate = B1200; break;
93     case 'k':
94         baudrate = B1800; break;
95     case 'l':
96         baudrate = B2400; break;
97     case 'm':
98         baudrate = B4800; break;
99     case 'n':
100         baudrate = B9600; break;
101     case 'o':
102         baudrate = B19200; break;
103     case 'p':
104         baudrate = B38400; break;
105     case 'q':
106         baudrate = B57600; break;
107     case 'r':
108         baudrate = B115200; break;
109     case 's':
110         baudrate = B230400; break;

```



```

111     case 't':
112         baudrate = B460800; break;
113     default: break;
114 }
115
116 int reconnect_tries = -1;
117 switch (recon) {
118     case 'a':
119         reconnect_tries = 1; break;
120     case 'b':
121         reconnect_tries = 3; break;
122     case 'c':
123         reconnect_tries = 5; break;
124     case 'd':
125         reconnect_tries = 7; break;
126     default: break;
127 }
128 /*
129 int timeout = -1;
130 switch (timeo) {
131     case 'a':
132         timeout = 2; break;
133     case 'b':
134         timeout = 3; break;
135     case 'c':
136         timeout = 5; break;
137     case 'd':
138         timeout = 8; break;
139     default: break;
140 }
141 */
142 set_basic_definitions(/*timeout,*/ reconnect_tries, 0, baudrate,
143                       packetSize);
144 setPacketSize(packetSize);
145
146 }
147
148 //return -1 if failed to send complete image, -2 if not even start was
149 //sent
150 int sendImage() {
151     //send
152     int ret = 0;
153     image_already_bytes = 0;
154     ret = sendFile(app.l2, app.l1, app.fd, image_name_length,
155                   image_name, image_bytes_length, image_bytes, &
156                   image_already_bytes);
157
158     //if(ret==-1) can_reconnect=YES;
159     return ret;
160 }
161 //return 0 if ok, -1 if image was not received, -2 start faled, -3 if
162 //connection failed on disk
163 int receiveImage() {
164     //receive
165     int ret = 0;
166     image_already_bytes = 0;
167     ret = receiveFile(app.fd, image_name, &image_bytes, &
168                      image_bytes_length, &image_already_bytes);
169
170     //if(ret==-1) can_reconnect=YES;
171     //receive disk(do this before saving image to avoid delays)

```

```

172     char* packet; int llread_result = 0;
173     llread_result = llread(app.fd, &packet);
174     if (llread_result != -2) //read returns -2 when receives disk
175     {
176         if (llread_result > 0) free(packet);
177         ret = -3;
178     }
179
180     return ret;
181 }
182
183
184 int reconnect()
185 {
186     if (image_already_bytes == 0 || image_already_bytes ==
187         image_bytes_length)
188     {
189         if (app.status) printf("\nNot possible:There is nothing to re-
190             send");
191         else printf("\nNot possible:There is no data already received
192             or all the data as already been received.");
193         return OK;
194     }
195     return OK;
196 }
197 //
198 // =====
199 // MAIN
200 // =====
201
202 int main(int argc, char** argv)
203 {
204     time_t t;
205     srand((unsigned) time(&t)); // seed for random numbers for random
206     error generator
207
208     if ((argc < 3) ||
209         ((strcmp("/dev/ttyS0", argv[1]) != 0) &&
210          (strcmp("/dev/ttyS4", argv[1]) != 0))
211         || ((strcmp("t", argv[2]) != 0) && (strcmp("r", argv[2]) != 0)))
212     {
213         printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS0 \
214             nAppStatus: t (=transmitter) or r (=receiver)\n");
215         exit(1);
216     }
217
218     if (strcmp("t", argv[2]) == 0) app.status = APP_STATUS_TRANSMITTER;
219     else app.status = APP_STATUS_RECEIVER;
220
221     app.l2 = L2; /* info packet size = L2 * 256 + L1 */
222     app.l1 = L1; /* defaults are L2 and L1 defined in AppProtocol.h
223     */
224
225     image_bytes_length = 0;
226     set_basic_definitions(3, argv[1], BAUDRATE, L2 * 256 + L1);
227
228     char anws = ' ';
229     while (anws != 'f'){
230         anws = main_menu(app.status);
231     }

```

```

229     switch (anws){
230
231     case 'a':
232         if (app.status == APP_STATUS_TRANSMITTER &&
            image_bytes_length <= 0)
233         {
234             printf("\nNO IMAGE SELECTED!");
235         }
236         else if (connect() == 0)
237         {
238             conection_open = TRUE;
239
240             if (
241                 (app.status == APP_STATUS_TRANSMITTER ?
242                  sendImage() : receiveImage()) == 0)
243             {
244                 show_display = NO;
245                 llclose(app.fd, 0); // normal close
246
247                 //save file if receiver
248                 if (app.status == APP_STATUS_RECEIVER){
249                     if (save2File(image_bytes, image_bytes_length,
250                                 image_name) != OK){
251                         free(image_bytes);
252                         image_bytes_length = 0;
253                         printf("\nImage was not saved sucessfully.\n"
254                               "n");
255                         return -1;
256                     }
257                     free(image_bytes);
258                     image_bytes_length = 0;
259                     printf("\nImage was saved sucessfully.\n");
260                 }
261                 else llclose(app.fd, 1); // hard close
262                 conection_open = FALSE;
263
264                 show_display = NO;
265             }
266             break;
267
268     case 'b':printf("\nNOT IMPLEMENTED");//reconnect();
269             break;
270
271     case 'c':select_config(config);
272             break;
273
274     case 'd':
275         if (app.status == APP_STATUS_TRANSMITTER)
276         {
277             if (image_bytes_length > 0) free(image_bytes);
278             image_bytes_length = selectNload_image(&image_bytes,
279                                                    image_name, &image_name_length);
280         }
281         else printf("\nNOT IMPLEMENTED");
282         break;
283
284     case 'e':
285         datalink_log = get_occurrences_log();
286         show_prog_stats(datalink_log->num_of_Is, datalink_log->
            total_num_of_timeouts, datalink_log->num_of_REJs, app.
            status);
287         break;
288
289     case 'f':printf("\nNow exiting...\n"); sleep(1);

```

```

290         break;
291
292         default: printf("\nNo valid command recognized."); sleep(1);
                break;
293     }
294 }
295
296 //devia de ser feito um exit handler com isto para caso haja uma
    terminacao inesperada
297 //if (conection_open) close_tio(app.fd); //nao podemos fazer isto
    porque AppProtocol nao conhece close_tio
298 if (image_bytes_length > 0) free(image_bytes);
299
300 return 0;
301 }

```

## A.2 AppProtocol.c

```

1
2
3 #include <sys/stat.h>
4 #include <stdio.h>
5 #include <string.h>
6 #include <stdlib.h>
7 #include <time.h>
8
9 #include "utilities.h"
10 #include "DataLinkProtocol.h"
11 #include "AppProtocol.h"
12 #include "FileFuncs.h"
13
14 //
    =====
15 //AUX DISPLAY
16 //
    =====
17 //will call show progress if packet send/received and if at least <
    UPDATE_DISPLAY_MIN_TIME_INTERVAL> seconds have elapsed
18 time_t start, end; //get elapsed time to avoid spamming interface
19 double timedif; //avoid interface spam
20 #define UPDATE_DISPLAY_MIN_TIME_INTERVAL 0.3f
21 int progress_icon_state = 0;
22 const int NUMBER_OF_BARS_IN_PROGRESS_BAR = 20;
23 const char progress_bar_character = '#';
24 char progress_icon = 0;
25 void show_progress(int appstatus, unsigned int image_already_bytes,
    unsigned int image_bytes_length)
26 {
27
28     switch (progress_icon_state){
29     case 0: progress_icon = '~'; break;
30     case 1: progress_icon = '\\'; break;
31     case 2: progress_icon = '|'; break;
32     case 3: progress_icon = '/'; break;
33     default: progress_icon = ' ';
34     }
35     progress_icon_state = (progress_icon_state + 1) % 4;
36
37     // - - - - -
38     system("clear");
39
40
41     /*if (data->estimate_recBytesPerSec > 1000)
42     printf("\n Rate:%d KB/sec", data->estimate_recBytesPerSec /

```

```

        1000);
43     else
44     printf("\n Rate:%d B/sec", data->estimate_recBytesPerSec);
45     */
46
47     if (appstatus) printf("Received ");
48     else printf("Sent ");
49
50     printf("\n %dKB of %dKB", (image_already_bytes) / 1000, (
        image_bytes_length) / 1000);
51
52     printf("\n-----");
53     printf("\n      PROGRESS %c", progress_icon);
54     printf("\n<");
55     int number_of_block_2_print = image_already_bytes / (
        image_bytes_length / NUMBER_OF_BARS_IN_PROGRESS_BAR);
56     int num_of_blanks_2_print = NUMBER_OF_BARS_IN_PROGRESS_BAR -
        number_of_block_2_print;
57     for (; number_of_block_2_print > 0; --number_of_block_2_print)
        printf("%c", progress_bar_character);
58     for (; num_of_blanks_2_print > 0; --num_of_blanks_2_print)
        printf(" ");
59     printf(">\n");
60 //}
61
62 //return 0;
63 }
64 //
65 //
=====
66 //MAIN FUNCS
67 //
=====
68
69 int getControlPacket(char control, unsigned int size, unsigned char
    nameSize, const char *name, char *controlPacket) {
70     char fileSize[32];
71     unsigned int n = 0;
72     while (size != 0) {
73         fileSize[n] = (char)size;
74         size >>= 8; // next byte
75         n++;
76     }
77     controlPacket[0] = ((control == START) ? CS : CE);
78     controlPacket[1] = TSIZE;
79     controlPacket[2] = n;
80     unsigned int i;
81     for (i = 0; i < n; i++) controlPacket[3 + i] = fileSize[i];
82     controlPacket[3 + n] = TNAME;
83     controlPacket[4 + n] = nameSize;
84     for (i = 0; i < nameSize; i++) controlPacket[5 + n + i] = name[i];
85     return 5 + n + nameSize; // 1 byte for C, 2 bytes for T and L, n
        bytes for size, 2 bytes for T and L, nameSize bytes for name
86 }
87
88 int getInfoPacket(unsigned char N, unsigned int infoSize, char *info,
    char *infoPacket) {
89     infoPacket[0] = CD;
90     infoPacket[1] = N;
91     infoPacket[2] = (unsigned char) (infoSize / 256);
92     infoPacket[3] = (unsigned char) (infoSize % 256);
93     unsigned int i;
94     for (i = 0; i < infoSize; i++) infoPacket[4 + i] = info[i];
95     return 4 + infoSize; // 1 byte for C, 1 byte for N, 2 bytes for
        L2 and L1, L2 * 256 + L1 bytes for info

```

```

96 }
97
98 int sendControlPacket(int fd, char *controlPacket, int
    sizeControlPacket) {
99     unsigned char try = 0;
100     while (try < MAX_TRY) {
101         if (llwrite(fd, controlPacket, sizeControlPacket) > 0) return
            OK;
102         try++;
103     }
104     return -1;
105 }
106
107 int sendInfoPacket(int fd, char *infoPacket, int sizeInfoPacket) {
108     unsigned char try = 0;
109     while (try < MAX_TRY) {
110         if (llwrite(fd, infoPacket, sizeInfoPacket) > 0) return OK;
111         try++;
112     }
113     return -1;
114 }
115
116
117
118 int sendFile(unsigned char l2, unsigned char l1, int fd, unsigned char
    fileNameSize, const char *fileName, unsigned int image_bytes_length,
    const char *image_bytes, unsigned int* out_already_sent_bytes) {
119     timedif = 0;
120     char controlPacket[MAX_CTRL_P];
121     int sizeControlPacket;
122     char infoPacket[MAX_INFO_P];
123     int sizeInfoPacket;
124     unsigned int infoSize;
125     unsigned int maxInfoSize = l2 * 256 + l1;
126     //printf("packet size: %u\n", maxInfoSize);
127     char info[maxInfoSize];
128     unsigned char N = 0;
129
130     //(unsigned int)image_bytes_length: this casting is not the ideal
        solution but works for now
131     sizeControlPacket = getControlPacket(START, (unsigned int)
        image_bytes_length, fileNameSize, fileName, controlPacket); //
        START control packet
132
133     if (sendControlPacket(fd, controlPacket, sizeControlPacket) != OK)
134         return -2;
135
136     //printf("\n--fileName;%s\nimlength:%l\n", fileName,
        image_bytes_length);
137     long i = 0;
138     while (1) {
139
140         time(&start); //avoid interface spam
141
142         infoSize = 0;
143         while (i < image_bytes_length && infoSize < maxInfoSize) {
144             info[infoSize] = image_bytes[i];
145             infoSize++;
146             i++;
147         }
148         if (infoSize == 0) break;
149         sizeInfoPacket = getInfoPacket(N, infoSize, info, infoPacket);
150         if (sendInfoPacket(fd, infoPacket, sizeInfoPacket) == OK) {
151             *out_already_sent_bytes += infoSize;
152             N++;
153
154             //interface stuff

```

```

155         time(&end); //avoid interface spam
156         timedif += difftime(end, start);
157         if (timedif >= UPDATE_DISPLAY_MIN_TIME_INTERVAL){
158             timedif = 0;
159             show_progress(1, *out_already_sent_bytes,
160                          image_bytes_length);
161         }
162     }
163     else {
164         return -1;
165     }
166 }
167 controlPacket[0] = CE; // END control packet
168 if (sendControlPacket(fd, controlPacket, sizeControlPacket) == OK)
169     return OK;
170
171 return -1;
172 }
173
174 int receivePacket(int fd, char **packet, int *sizePacket) {
175     *sizePacket = llread(fd, packet);
176     if (*sizePacket > 0) return OK;
177     //else if (*sizePacket == -2) return -2; /*received disk*/
178     else return -1;
179 }
180
181 #define DEBUG_RECEIVE_STEPS 1
182 int receiveFile(int fd, char* out_imagename, char** out_imagebuffer,
183                unsigned int* out_image_buffer_length, unsigned int*
184                out_already_received_imgbytes) {
185     timedif = 0;
186     char* packet;
187     int sizePacket;
188     unsigned int infoSize;
189     unsigned char N = 0;
190     char fileName[255];
191     unsigned char fileNameSize;
192     unsigned int fileSize;
193     if (receivePacket(fd, &packet, &sizePacket) == OK) {
194         if (packet[0] != CS) {
195             free(packet);
196             return -2;
197         }
198         if (packet[1] != TSIZE){
199             free(packet);
200             return -2;
201         }
202         unsigned char sizeFileSize = packet[2];
203         fileSize = 0;
204         unsigned int multiply = 1;
205         unsigned int i;
206         for (i = 0; i < sizeFileSize; i++) {
207             //printf("\n>>> %u\n", ((unsigned int)packet[3 + i]));
208             /*char to uint faz signal extend!!!*/
209             fileSize += (0x00ff & ((unsigned int)packet[3 + i])) *
210                 multiply;
211             multiply *= 256;
212         }
213         *out_already_received_imgbytes = 0;
214         *out_image_buffer_length = fileSize;
215         *out_imagebuffer = (char*)malloc(fileSize);
216         DEBUG_SECTION(DEBUG_RECEIVE_STEPS,
217             printf("\nfileSize: %d\n", fileSize));

```

```

218     if (packet[3 + sizeFileSize] != TNAME) {
219         free(packet); return -2;
220     }
221
222     fileNameSize = packet[4 + sizeFileSize];
223     for (i = 0; i < fileNameSize; i++)
224         fileName[i] = packet[5 + sizeFileSize + i];
225
226     memmove(out_imagename, fileName, (int) fileNameSize);
227     out_imagename[(int)((int)fileNameSize>255 ? 255 : fileNameSize)
228         ] = 0; //some extra precaution
229
230     DEBUG_SECTION(DEBUG_RECEIVE_STEPS,
231         printf("\nfile Name: %s\n", out_imagename));
232
233     free(packet);
234     time(&start); //avoid interface spam
235     while (receivePacket(fd, &packet, &sizePacket) == OK) {
236         if (packet[0] == CD) {
237             if ((unsigned char) packet[1]) == N) {
238                 infoSize = (((unsigned int) packet[2]) & 0x00ff) *
239                     256 + (((unsigned int) packet[3]) & 0x00ff);
240                 for (i = 0; i < infoSize; i++)
241                 {
242                     if (fileSize <= (*out_already_received_imgbytes)
243                         )
244                     {
245                         printf("\nERROR: receiveFile(...) received
246                             more data bytes than expected\n");
247                         free(packet);
248                         return -1;
249                     }
250                     (*out_imagebuffer)[(*
251                         out_already_received_imgbytes)] = packet[4 +
252                         i];
253                     (*out_already_received_imgbytes)++; //counts
254                     received bytes
255                 }
256                 N++;
257
258                 //interface stuff
259                 time(&end); //avoid interface spam
260                 timedif += difftime(end, start);
261                 if (timedif >= UPDATE_DISPLAY_MIN_TIME_INTERVAL){
262                     timedif = 0;
263                     show_progress(1, *out_already_received_imgbytes
264                         , *out_image_buffer_length);
265                 }
266             }
267             else {
268                 printf("\nERROR: receiveFile(...) not valid N\n");
269                 free(packet);
270                 return -1;
271             }
272         }
273         else if (packet[0] == CE) {
274             if (packet[1] != TSIZE) { printf("\nERROR: receiveFile
275                 (...) got CE with non valid TSIZE\n"); free(packet);
276                 return -1; }
277
278             if (packet[2] != sizeFileSize) { printf("\nERROR:
279                 receiveFile(...) got CE with non valid sizeFileSize
280                 \n"); free(packet); return -1; }

```



```

273         unsigned int finalFileSize = 0;
274         multiply = 1;
275         for (i = 0; i < sizeFileSize; i++) {
276             finalFileSize += (0x00ff & ((unsigned int)packet[3
                + i])) * multiply;
277             multiply *= 256;
278         }
279
280         if (finalFileSize != fileSize) { printf("\nERROR:
                receiveFile(...) got CE with non valid fileSize\n")
                ; free(packet); return -1; }
281         if (packet[3 + sizeFileSize] != TNAME) { printf("\
                nERROR: receiveFile(...) got CE with non valid
                TNAME\n"); free(packet); return -1; }
282         if (packet[4 + sizeFileSize] == fileNameSize) {
283             for (i = 0; i < fileNameSize; i++)
284                 if (packet[5 + sizeFileSize + i] != fileName[i]
                    ]) return -1;
285             free(packet);
286             return OK;
287         }
288         else {
289             printf("\nERROR: receiveFile(...) END file name !=
                START file name\n");
290             free(packet);
291             return -1;
292         }
293     }
294     else{
295         printf("\nERROR: receiveFile(...) end CE not received\n
                ");
296         free(packet);
297         return -1;
298     }
299
300     free(packet);
301
302     time(&start); //avoid interface spam
303 } //---(receieve packets loop) endwhile---
304 return -1;
305 }
306 printf("\nERROR:receiveFile(...) ???\n");
307 return -1;
308 }

```

### A.3 AppProtocol.h

```

1
2 #ifndef APPPROTOCOL
3 #define APPPROTOCOL
4
5 #define CD 0b00000000 // campo de controlo no caso de um pacote de
    dados
6 #define CS 0b00000001 // campo e controlo no caso de um pacote de start
7 #define CE 0b00000010 // campo e controlo no caso de um pacote de end
8 #define TSIZE 0b00000000 // type para pacote de controlo no caso de ser
    tamanho do ficheiro
9 #define TNAME 0b00000001 // type para pacote de controlo no caso de ser
    nome do ficheiro
10 #define START 1
11 #define END 2
12 #define L2 50
13 #define L1 100
14 #define MAX_CTRL_P 264 /* maximum size of control packet: 1 byte for C,
    2 bytes for T and L, 4 bytes for size, 2 bytes for T and L, 255
    bytes for name */

```

```

15 #define MAX_INFO_P 65540 /* maximum size of info packet: 1 byte for C,
    1 byte for N, 2 bytes for L2 and L1, 255 * 256 + 255 bytes for info
    */
16 #define MAX_TRY 1
17
18 /**
19 @return 0(OK) if no errors/problems. -1 if failed to send everything,
    -2 failed on sending start
20 */
21 int sendFile(unsigned char l2, unsigned char l1, int fd, unsigned char
    fileNameSize, const char *fileName, unsigned int image_bytes_length
    , const char *image_bytes, unsigned int* out_already_sent_bytes);
22
23 /**
24 @param out_imagename will get the name of the file to be used later.
    must be a 255 char array
25 @param out_imagebuffer will get the image bytes. Must be a dynamic
    array and be freed outside ths method
26 @return 0(OK) if no errors/problems. something else otherwise. -1
    failed to receive everything, -1 failed to recive start
27 */
28 int receiveFile(int fd, char* out_imagename, char** out_imagebuffer,
    unsigned int* out_image_buffer_length, unsigned int*
    out_already_received_imgbytes);
29
30 #endif /*APPPROTOCOL*/

```

## A.4 DataLinkProtocol.c

```

1
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/stat.h>
5 #include <fcntl.h>
6 #include <stdio.h>
7 #include <signal.h>
8 #include <unistd.h>
9
10 #include <termios.h>
11 #include <unistd.h>
12 #include <string.h>
13
14 #include "utilities.h"
15 #include "DataLinkProtocol.h"
16
17 // #define MAX_FRAME_SIZE 64
18 struct linkLayer{
19     char port[20]; /*Dispositivo /dev/ttySx, x = 0, 4*/
20     int baudRate; /*Velocidade de transmissao*/
21     unsigned int sequenceNumber; /*Numero de sequencia da trama: 0, 1*/
22     int timeout; /*Valor do temporizador: 1 s*/
23     int numTransmissions; /*Numero de tentativas em caso de
24                             falha*/
25     //int lframe_numdatabytes;
26     //char frame[MAX_FRAME_SIZE]; /*trama*/
27 };
28
29
30 typedef char message_type;
31 #define MESSAGE_SET 1
32 #define MESSAGE_DISC 2
33 #define MESSAGE_UA 3
34 #define MESSAGE_RR 4
35 #define MESSAGE_REJ 5
36 #define MESSAGE_I 6
37

```

```

38
39 //X=X=X=X    general debug stuff
40 #if (1)
41 //
=====
42
43 #define DEBUG_READ_BYTES    1
44 int total_read; /*variable used for debug purposes, increment when
    reading bytes*/
45
46 //send a bcc with 0 to tests the answer when errors occur
47 bool genObcc = FALSE;
48 #define DEBUG_REJ_WITHWRONG_BCCS    0
49
50 #define DEBUG_PRINT_SECTION_NUM 0
51 #endif
    //=====
=====
52
53 //X=X=X=X    basic definitions
54 #if (1)
55 //
=====
=====
56
57 // #define BAUDRATE B38400
58 // #define MODEMDEVICE "/dev/ttyS1"
59 // #define _POSIX_SOURCE 1 /* POSIX compliant source */
60
61 struct linkLayer link_layer_data; /*contains data related to the link
    layer*/
62
63 struct termios oldtio, newtio; /*termios old and new configuration*/
64 //int private_tio_fd;          /*termios file descriptor*/
65
66 struct occurrences_Log occ_log;
67 occurrences_Log_Ptr get_occurrences_log()
68 {
69     return &occ_log;
70 }
71
72 app_status_type app_status; //indicates i fits transmissor or receiver
73
74 int OPENED_TERMIOS = FALSE; /*indicates if termios is open. could be
    used in handlers if anything goes wrong*/
75
76 int NS = 0;
77 int NR = 0;
78 //int CONNECTION_TYPE; /*emissor or receptor*/
79
80 #endif
    //=====
=====
81
82 //X=X=X=X    ALARM related
83 #if (1)
    //=====
=====
84
85 // #define TIMEOUTS_ALLOWED 3
86 //int timeout_inseconds;
87 volatile int STOP = FALSE; //STOP s used to repeat a prcedure before
    the alarm handler rings/*volatile:used in cycles that could be
    interrupted from another thread or interrupt*/
88 volatile int timeouts_done;
89

```

```

90 void timeout_alarm_handler() // atende alarme
91 {
92     printf("\nalarme # %d\n", timeouts_done);
93     timeouts_done++;
94     STOP = TRUE;
95     /*update occurrences log*/++occ_log.total_num_of_timeouts;
96 }
97
98
99 void startAlarm() {
100     STOP = FALSE;
101     alarm(link_layer_data.timeout);
102 }
103
104 void stopAlarm() {
105     STOP = TRUE;
106     alarm(0);
107 }
108
109 #endif
    //=====

110
111 //X=X=X=X    tramas
112 #if (1)
113 //
    =====

114
115 #define FLAG 0b01111110 // FLAG no inicio e fim da trama
116 #define ATRANS 0b00000011 // A se for o emissor a enviar a trama e o
    receptor a responder
117 #define ARECEI 0b00000001 // A se for o receptor a enviar a trama e o
    emissor a responder
118 #define SET 0b00000111 // C se for uma trama de setup
119 #define DISC 0b00001011 // C se for uma trama de disconnect
120 #define UA 0b00000011 // C se for uma trama de unnumbered
    acknowledgement
121 #define RR0 0b00000001 // C se RR se S = 1, pede mensagem seguinte, com
    R = 0
122 #define RR1 0b00100001 // C se RR se S = 0, pede mensagem seguinte, com
    R = 1
123 #define REJ0 0b00000101 // C se REJ se R = 0, pede novamente mensagem
    com R = 0
124 #define REJ1 0b00100101 // C se REJ se R = 1, pede novamente mensagem
    com R = 1
125 #define IO 0b00000000 //C se I se S = 0
126 #define I1 0b00100000 //C se I se S = 1
127 #define ESCAPE 0b01111101 //usado no stuffing e destuffing
128
129 #define BCC_ON_SET 0b00000100//A TRANS ^ SET
130 #define BBC_ON_UA 0b00000000//A RECEI ^ UA
131 const char SET_MSG[5] = { FLAG, ATRANS, SET, BCC_ON_SET, FLAG };
132 const char UA_MSG[5] = { FLAG, ATRANS, UA, BBC_ON_UA, FLAG };
133
134 /* C se for uma trama de positive acknowledgment */
135 char getRR(int R) {
136     if (R == 0) return RR0;
137     else return RR1;
138 }
139
140 /* C se for uma trama de negative acknowledgment, R identifica a trama
    a que estamos a responder */
141 char getREJ(int R) {
142     if (R == 0) return REJ0;
143     else return REJ1;
144 }

```

```

145
146 /* C se for uma trama de I, S */
147 char getI(int S) {
148     if (S == 0) return I0;
149     else return I1;
150 }
151
152 /* A depende do sentido da trama original */
153 char getA(app_status_type status) {
154     if (status == APP_STATUS_TRANSMITTER) return ATRANS; // A se for o
        emissor a enviar a trama e o receptor a responder
155     else return ARECEI; // A se for o receptor a enviar a trama e o
        emissor a responder
156 }
157
158 /* C depende do tipo de trama e, se for positive ou negative
    acknowledgment, do numero da mensagem R (S se for I)*/
159 char getC(message_type message, int R) {
160     if (message == MESSAGE_SET) return SET;
161     else if (message == MESSAGE_DISC) return DISC;
162     else if (message == MESSAGE_UA) return UA;
163     else if (message == MESSAGE_I) return getI(R);
164     else if (message == MESSAGE_RR) return getRR(R);
165     else return getREJ(R);
166 }
167
168 /* BCC1 codigo de verificacao, depende de A e de C: BCC1 = A ^ C (A ou
    exclusivo C) */
169 char getBCC1(app_status_type status, message_type message, int R) {
170     return getA(status) ^ getC(message, R);
171 }
172
173 /* F | A | C | BCC1 | F */
174 void getMessage(app_status_type status, message_type message, int R,
    char* msg) {
175     msg[0] = FLAG;
176     msg[1] = getA(status);
177     msg[2] = getC(message, R);
178     msg[3] = getBCC1(status, message, R);
179 }
180
181 #endif
    //=====

182
183 //X=X=X=X    SET BASICS, OPEN AND CLOSE TERMIOS
184 #if (1)
185 //
    =====

186 bool port_name_was_set = NO;
187 void set_basic_definitions(int number_of_tries_when_failing, char* port
    , int baudrate, int packetSize)
188 {
189     DEBUG_SECTION(DEBUG_PRINT_SECTION_NUM,
190         printf("\n-section1-");
191     sleep(1);
192     );
193
194     signal(SIGALRM, timeout_alarm_handler);
195
196     int realbauds[20] = {0, 50, 75, 110, 134, 150, 200, 300, 600, 1200,
197         1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400,
198         460800};
199     int realBaudrate;
200     if (baudrate <= 15) realBaudrate = realbauds[baudrate];
201     else realBaudrate = realbauds[baudrate - 4081];

```

```

200
201     int timeout_in_seconds = ((packetSize + 4 + 1) * 2 + 5) / (
        realBaudrate / 8) + 1;
202     printf("size: %d\n", packetSize);
203     printf("baudrate: %d\n", realBaudrate);
204     printf("timeout: %d\n", timeout_in_seconds);
205     link_layer_data.timeout = timeout_in_seconds;
206     link_layer_data.numTransmissions = number_of_tries_when_failing;
207     if (port_name_was_set == NO) { strcpy(link_layer_data.port, port);
        port_name_was_set = YES; }
208     link_layer_data.baudRate = baudrate;
209 }
210
211 int open_tio(int* tio_fd, int vtime, int vmin)
212 {
213     total_read = 0;
214
215     int private_tio_fd = open(link_layer_data.port, O_RDWR | O_NOCTTY);
216     if (private_tio_fd < 0) { perror(link_layer_data.port); exit(-1); }
217
218     if (tcgetattr(private_tio_fd, &oldtio) == -1) { /* save current
        port settings */
219         perror("tcgetattr");
220         exit(-1);
221     }
222
223     bzero(&newtio, sizeof(newtio));
224     newtio.c_cflag = link_layer_data.baudRate | CS8 | CLOCAL | CREAD;
225     newtio.c_iflag = IGNPAR;
226     newtio.c_oflag = OPOST;
227
228     /* set input mode (non-canonical, no echo,...) */
229     newtio.c_lflag = 0;
230     newtio.c_cc[VTIME] = vtime; /* link_layer_data.timeout * 10;  /*
        inter-character timer unused */
231     newtio.c_cc[VMIN] = vmin; /* blocking read until X chars received
        */
232
233
234     tcflush(private_tio_fd, TCIFLUSH);
235     if (tcsetattr(private_tio_fd, TCSANOW, &newtio) == -1) {
236         perror("tcsetattr in open_tio");
237         exit(-1);
238     }
239
240     if (private_tio_fd != 0) *tio_fd = private_tio_fd;
241
242     return OK;
243 }
244
245 int close_tio(int tio_fd)
246 {
247     printf("\n- - - - -");
248     printf("\nnow sleeping for 2 seconds before closing fd...\n");
249     sleep(2); /* just a precaution
250
251     printf("\n- - - - -");
252     printf("\ntotal: %d\nattemp to close fd...", total_read);
253     if (tcsetattr(/*private_*/tio_fd, TCSANOW, &oldtio) == -1) {
254         perror("\ntcsetattr in close_tio");
255         close(/*private_*/tio_fd);
256         exit(-1);
257     }
258     close(/*private_*/tio_fd);
259     printf("\nfd closed without problems.");

```

```

260     printf("\n- - - - -\n");
261     return OK;
262 }
263
264 #endif
    //=====
265
266 //X=X=X=X    PROTOCAL AUX FUNCS
267 #if (1)
268 //
    =====
269
270
271 //-----    GET MSGS STATE MACHINE - - - - -
    - - - - -
272 #if (1)
273
274 typedef int state_machine_state;
275 #define STATE_MACHINE_START      1
276 #define STATE_MACHINE_FLAG_RCV   2
277 #define STATE_MACHINE_A_RCV      3
278 #define STATE_MACHINE_C_RCV      4
279 #define STATE_MACHINE_BCC_RCV    5
280 #define STATE_MACHINE_STOP       6
281 #define STATE_MACHINE_ESCAPE     7
282
283 /*msgExpectedType SET; UA ; DISC ; RR ; I (DONT USE REJ IT WILL BE
    CONSIDERED WHEN USING RR)
284 !!! !!! !!! IMPORTANT: AFTER REACHING TERMINAL STATES THE STATE MUST BE
    RESETED FROM OUTSIDE !!! !!! !!!
285
286 appStatus      indica o tipo de estado da aplicacao (transmissor ou
    receptor)
287 adressStatus   indica o tipo de campo de endereco
288 state          o estado actual a atualizar
289 msgExpectedType o tipo de mensaem que se esta a espera de receber
    (antes de receber o C)
290 rcv            o que caracter que foi lido da porta serie
291
292 SO VAI TRANSITAR SE APANHAR DO TIPO ESPERADO,
293 EXCEPTO O READ Q PODE APANHAR SETS OU DISKS
294 E O WRITE PODE APANHAR REJ MSM ESPERANDO RRs
295
296 use received_C_type to confirm the type of packet received or check S/R
    for possible missing packet
297 */
298 message_type received_C_type = -1;//not sure if the most correct
    approach but simplifies the state machine a lot, indicates type of
    message received and is reused outside ethod when _STOP state is
    reached
299 char received_C=0;
300 #define DEBUG_LLO_STATE_MACHINE 0
301 #define DEBUG_STATE_MACHINE_GETC 0
302 int update_state_machine(app_status_type appStatus, app_status_type
    adressStatus, message_type msgExpectedType, char rcv,
    state_machine_state* state)
303 {
304     DEBUG_SECTION(DEBUG_LLO_STATE_MACHINE,
305         printf("\nappstatus:%d tramastatus(A):%d msgtype:%d state:%
            d rcv:" PRINTBYTETOBINARY, appStatus, adressStatus,
            msgExpectedType, *state, BYTETOBINARY(rcv));
306 );
307
308 if (*state < STATE_MACHINE_BCC_RCV)

```

```

309 {
310     if (rcv == FLAG)
311     {
312         *state = STATE_MACHINE_FLAG_RCV;
313         return OK;
314     }
315 }
316
317 switch (*state)
318 {
319     case STATE_MACHINE_START: return OK; /*flag checked before switch*/
320
321     case STATE_MACHINE_FLAG_RCV:
322         if (rcv == getA(adressStatus)) *state = STATE_MACHINE_A_RCV;
323         else *state = STATE_MACHINE_START;
324         return OK;
325
326     case STATE_MACHINE_A_RCV:
327         if (msgExpectedType == MESSAGE_I) //no READ c/ tramas I pra
328             apanhar possiveis sets ou discs
329         {
330             if (rcv == getC(MESSAGE_I, 0) || rcv == getC(MESSAGE_I, 1))
331             { *state = STATE_MACHINE_C_RCV; received_C_type =
332                 MESSAGE_I; }
333             /*catch set on read()*/ else if (appStatus ==
334                 APP_STATUS_RECEIVER && rcv == getC(MESSAGE_SET, 0)) { *
335                 state = STATE_MACHINE_C_RCV; received_C_type =
336                 MESSAGE_SET; }
337             else if (appStatus == APP_STATUS_RECEIVER && rcv == getC(
338                 MESSAGE_DISC, 0)) { *state = STATE_MACHINE_C_RCV;
339                 received_C_type = MESSAGE_DISC; }
340             else *state = STATE_MACHINE_START;
341         }
342         //WRITE RR/REJ
343         else if (msgExpectedType == MESSAGE_RR || msgExpectedType ==
344             MESSAGE_REJ)
345         {
346             if (rcv == getC(MESSAGE_RR, 0) || rcv == getC(MESSAGE_RR,
347                 1)) { *state = STATE_MACHINE_C_RCV; received_C_type =
348                 MESSAGE_RR; }
349             else if (rcv == getC(MESSAGE_REJ, 0) || rcv == getC(
350                 MESSAGE_REJ, 1)) { *state = STATE_MACHINE_C_RCV;
351                 received_C_type = MESSAGE_REJ; }
352             else *state = STATE_MACHINE_START;
353         }
354         //SET,GET or DISC
355         else if (rcv == getC(msgExpectedType, 0)) {
356             *state = STATE_MACHINE_C_RCV;
357             received_C_type = msgExpectedType;
358         }
359         else *state = STATE_MACHINE_START;
360         received_C=rcv;
361         return OK;
362
363     case STATE_MACHINE_C_RCV:
364         //note:C type already received so it is a valid msg type (if it
365         wasn't state would go back to START) unless we have an
366         error in which bcc will hopefully fail
367         //BBC1 is not generating REJ, only BBC2. should I change this
368         behaviour???
369         if (rcv == getBCC1(adressStatus, received_C_type,
370             /*will only work on RR and REJ*/(received_C>>5 & 0b00000001))
371             )
372             *state = STATE_MACHINE_BCC_RCV;
373         else *state = STATE_MACHINE_START;
374
375         DEBUG_SECTION(DEBUG_STATE_MACHINE_GETC,
376             char aux = getBCC1(adressStatus, received_C_type,(received_C>>5

```



```

        & 0b00000001));
360     printf("\n--" PRINTBYTETO_BINARY , BYTETO_BINARY(aux));
361     );
362
363     return OK;
364
365     case STATE_MACHINE_BCC_RCV://also goes 2 this state after
STATE_MACHINE_ESCAPE
366         if (rcv == FLAG) *state = STATE_MACHINE_STOP;
367         else if (received_C_type == MESSAGE_I)
368             {
369                 if (rcv == ESCAPE) *state = STATE_MACHINE_ESCAPE;
370             }
371         else *state = STATE_MACHINE_START;
372         return OK;
373
374     case STATE_MACHINE_ESCAPE://could be done outside but makes more
sense to be here
375         *state = STATE_MACHINE_BCC_RCV;
376         return OK;
377
378     default:
379         printf("\nWARNING(usm2):Not valid/expected state (%d) reached
in --> int state_machine(app_status_type status) => from =>
Protocol.c\n", *state);
380         return 1;
381     }
382
383     return 1;
384 }
385
386 #endif /*GET MSGS STATE MACHINE*/
387
388 //----- STTUFING AND DESTUFFING - - - - -
389 #if (1)
390 /*buf deve entrar ja com o dobro do tamanho dos caracteres que tem
391 pode ser dinamico ou nao
392 data_size deve ser o tamanho efectivo ocupado
393 assim poupasse processamento extra usando a solucao em 1) ou 2) e
evitasse possiveis erros de memoria
394 */
395 int apply_stuffing(char* buf, int /*bufSize*/data_size)
396 {
397     int newSize = data_size;
398     int i = 0;
399
400     /*
401     //1)count prev 2 avoid multiple reallocs
402     for (; i < bufSize; ++i)/
403         if (*buf[i] == FLAG || *buf[i] == ESCAPE)
404             ++newSize;
405     *buf = (char)realloc(*buf, newSize);
406     */
407
408     i = 0;
409     for (; i < newSize; ++i)
410         if (buf[i] == FLAG || buf[i] == ESCAPE)
411             {
412                 /*2)using multiple reallocs*/*buf = (char)realloc(*buf,++
newSize);
413                 ++newSize;
414                 memmove(buf + i + 1, buf + i, newSize - i);
415                 buf[i] = ESCAPE;
416                 buf[i + 1] ^= 0x20;
417                 i++;
418             }

```

```

419
420     return newSize;
421 }
422
423 //buf nao precisa de ser array dinamico
424 #define DEBUG_DESTUFFING 0
425 int apply_destuffing(char* buf, int bufSize)
426 {
427     //int newSize = bufSize;
428     int i = 0;
429     for (; i < bufSize; ++i)
430         if (buf[i] == ESCAPE)
431         {
432             --bufSize;
433             memmove(buf + i, buf + i + 1, bufSize - i);
434             buf[i] ^= 0x20;
435
436             DEBUG_SECTION(DEBUG_DESTUFFING,
437                 printf("\n");
438                 printf("ZZ-");
439                 int a=0;
440                 for(;a<bufSize;++a)
441                     printf(PRINTBYTETOBINARY " ", BYTETOBINARY(buf[a])););
442             }
443
444     return bufSize;
445 }
446 #endif /*STTUFING AND DESTUFFING*/
447
448 //----- BCC2 GENERATION AND VALIDATION - - - - -
449 #if (1)
450 #define BCC2_EVEN 0
451 #define BCC2_ODD 1
452 #define DEBUG_GENBBC 0
453 char genBCC2(char* buf, int bufsize) {
454
455     /*send (almost always) invalid bcc*/if (gen0bcc) return 0;
456
457     char BCC2;
458     if (BCC2_EVEN) BCC2 = 0b11111111;
459     else          BCC2 = 0b00000000;
460
461     int i = 0;
462     for (; i < bufsize; i++)
463     {
464         BCC2 ^= buf[i];
465         DEBUG_SECTION(DEBUG_GENBBC, printf("\ngenbcc2debug(i:%d):"
466             PRINTBYTETOBINARY, i, BYTETOBINARY(BCC2))););
467     }
468     return BCC2;
469 }
470
471 #define DEBUG_VALIDATEBBC 0
472 char validateBCC2(char* buf, int bufsize) {
473     char valid = buf[0];
474     int i = 1;
475     for (; i < bufsize; i++)
476         valid ^= buf[i];
477
478     DEBUG_SECTION(DEBUG_VALIDATEBBC, printf("\nvalbcc2debug:"
479         PRINTBYTETOBINARY, BYTETOBINARY(valid))););
480
481     if (BCC2_EVEN) { if (valid == 0b11111111) return OK; }
482     else if (valid == 0b00000000) return OK;

```

```

483     return 1;
484 }
485 #endif /*BCC2 GENERATION AND VALIDATION */
486
487 //----- WRITE AND READ MSGS AUX - - - - -
488
489 #if (1)
490 void write_UorS(app_status_type adressStatus, message_type msg_type,
491     int SorR, int fd)
492 {
493     char msg[4];
494     getMessage(adressStatus, msg_type, SorR, msg);
495     if (write(fd, msg, 4) != 4)
496     {
497         perror("write_UorS()");
498     }
499     if (write(fd, msg, 1) != 1)
500     {
501         perror("write_UorS()");
502     }
503 }
504
505 #define CABE_PROB 100 // significa que em cada CABE_PROB cabecalhos
506                       // haveria um erro, em media. 0 se nao quisermos erros
507 #define DATA_PROB 100 // significa que em cada DATA_PROB campos
508                       // de dados haveria um erro, em media. 0 se nao quisermos erros
509
510 void randomErrorCabe(char* msg) {
511     int trama_byte;
512     if (CABE_PROB > 0) {
513         int cabe_error = rand() % CABE_PROB;
514         if (cabe_error == 0) {
515             trama_byte = rand() % 3 + 1; // escolher erro
516             // entre A, C e BCC1
517             msg[trama_byte]++; // o erro e
518             incrementar o byte escolhido
519         }
520     }
521 }
522
523 void randomErrorData(char* trama, int data_size) {
524     int trama_byte;
525     if (DATA_PROB > 0) {
526         int data_error = rand() % DATA_PROB;
527         if (data_error == 0) {
528             trama_byte = rand() % data_size; // escolher o byte
529             // de erro na zona dos dados da trama
530             trama[trama_byte]++; // o erro e
531             incrementar o byte escolhido
532         }
533     }
534 }
535
536 #define DEBUG_WRITE_I 0
537 //data must come in without stuffing
538 int write_I(int SorR, int fd, char* data, int data_size)
539 {
540     char msg[4];
541     getMessage(APP_STATUS_TRANSMITTER, MESSAGE_I, SorR, msg);
542     randomErrorCabe(msg);
543     if (write(fd, msg, 4) != 4)
544     {
545         perror("write_I()");
546         return -1;
547     }
548 }

```

```

542     }
543
544     DEBUG_SECTION(DEBUG_WRITE_I,
545         printf("\n");
546         int i=0;
547         for(;i<data_size;++i)
548     printf(PRINTBYTETOBINARY " ", BYTETOBINARY(data[i]));
549 );
550
551     char finalMessage2Send[(data_size+1)*2]; //char* finalMessage2Send =
552         (char*)malloc(data_size + 1);
553     memcpy(finalMessage2Send, data, data_size);
554     finalMessage2Send[data_size] = genBCC2(data, data_size);
555
556     randomErrorData(finalMessage2Send, data_size);
557
558     DEBUG_SECTION(DEBUG_REJ_WITHWRONG_BCCS,
559         gen0bcc = gen0bcc? FALSE: TRUE;);
560
561     data_size = apply_stuffing(finalMessage2Send, data_size + 1);
562
563     DEBUG_SECTION(DEBUG_WRITE_I,
564         printf("\nDEBUG WRITTE I SECT 2, data_size=%d",data_size)
565         ;
566         printf("\n");
567         int i=0;
568         for(;i<data_size;++i)
569     printf(PRINTBYTETOBINARY " ", BYTETOBINARY(finalMessage2Send[i]));
570 );
571
572     //set size to max available
573     if (write(fd, finalMessage2Send, data_size) != data_size)
574     {
575         perror("write_I()");
576         return -1;
577     }
578
579     if (write(fd, msg, 1) != 1)
580     {
581         perror("write_I()");
582         return -1;
583     }
584
585     //free(finalMessage2Send);
586
587     return 5 + data_size;
588 }
589 #endif /*WRITE AND READ MSGS AUX*/
590
591 // - - - LLOPEN AUXS - - - - -
592
593 #if (1)
594 #define DEBUG_LLOPEN_RECEIVER_OVERFLOW 0
595 int llopen_receiver(int fd)
596 {
597     state_machine_state state = STATE_MACHINE_START;
598     char buf[20]; //used to receive stuff
599     int res; //number of bytes read or sent
600     // - - - - -
601     //receive SET
602     while (state != STATE_MACHINE_STOP || received_C_type !=
603         MESSAGE_SET)
604     {
605         if ((res = read(fd, buf, 20)) < 0) perror("llopen_receiver:");
606
607         DEBUG_SECTION(DEBUG_LLOPEN_RECEIVER_OVERFLOW,
608             if (res > 5)

```

```

605         printf("\nWARNING:llopen_receiver received more than 5
           characters at 1 time\n");
606     );
607
608     int i = 0;
609     for (; i < res && (state != STATE_MACHINE_STOP ||
        received_C_type != MESSAGE_SET); ++i)
610         update_state_machine(APP_STATUS_RECEIVER,
            APP_STATUS_TRANSMITTER, MESSAGE_SET, buf[i], &state);
611
612     DEBUG_SECTION(DEBUG_LLOPEN_RECEIVER_OVERFLOW,
613         if (res > i)
614             printf("\nWARNING:llopen_receiver received more
                characters than expected\n");
615     );
616 }
617 // - - - - -
618 write_UorS(APP_STATUS_TRANSMITTER, MESSAGE_UA, 0, fd);
619
620 return OK;
621 }
622
623 #define DEBUG_LLOPEN_TRANSMITTER_OVERFLOW 0
624 //should send set and receive UA
625 int llopen_transmitter(int fd)
626 {
627     int res;
628     state_machine_state state = STATE_MACHINE_START;
629     timeouts_done = 0;
630     char buf[20]; //used to receive stuff
631     bool QUIT = NO;
632
633     DEBUG_SECTION(DEBUG_PRINT_SECTION_NUM,
634         printf("\n-section2-");
635     sleep(1);
636     );
637
638     while (QUIT == NO && timeouts_done < link_layer_data.
        numTransmissions)
639     {
640         write_UorS(APP_STATUS_TRANSMITTER, MESSAGE_SET, 0, fd);
641
642         startAlarm(); //ready alarm for possible timeout
643
644         DEBUG_SECTION(DEBUG_PRINT_SECTION_NUM,
645             printf("\n-section3-");
646         sleep(1);
647         );
648
649         while (STOP == FALSE)
650         {
651             //get response
652             if ((res = read(fd, buf, 20)) < 0) perror("
                llopen_transmitter:");
653
654             DEBUG_SECTION(DEBUG_PRINT_SECTION_NUM,
655                 printf("\n-section4-");
656             sleep(1);
657             );
658
659             DEBUG_SECTION(DEBUG_LLOPEN_TRANSMITTER_OVERFLOW,
660                 if (res > 5)
661                     printf("\nWARNING:llopen_transmitter received more
                        than 5 characters at 1 time\n");
662             );
663
664             //update state machine and stop loop if valid

```

```

665         int i = 0;
666         for (; i < res && (state != STATE_MACHINE_STOP ||
        received_C_type != MESSAGE_UA); ++i)
667             update_state_machine(APP_STATUS_TRANSMITTER,
        APP_STATUS_TRANSMITTER, MESSAGE_UA, buf[i], &state);
668
669         if (state == STATE_MACHINE_STOP && received_C_type ==
        MESSAGE_UA)
670         {
671             stopAlarm(); QUIT = YES; break;
672         }
673
674         DEBUG_SECTION(DEBUG_LLOPEN_TRANSMITTER_OVERFLOW,
675             if (res > i)
676                 printf("\nWARNING:llopen_transmitter received more
        characters than expected\n");
677     );
678 }
679
680 }
681
682 //END AND RETURN
683 //if did not receive confirmation UA
684 if (state != STATE_MACHINE_STOP || received_C_type != MESSAGE_UA)
685 {
686     printf("\nllopen: No confirmation of the reception was received
        !");
687     if (timeouts_done >= link_layer_data.numTransmissions)
688         printf("\nllopen: Max tries (timeouts) reached.\n");
689     return -1;
690 }
691 //or else
692 printf("\nllopen: Connection established.\n");
693 return OK;
694 }
695
696 #endif /*LLOPEN AUXS*/
697
698 // - - - LLCLOSE AUXS - - - - -
699 #if (1)
700
701 int llclose_receiver(int fd)
702 {
703     int res;
704     state_machine_state state = STATE_MACHINE_START;
705     timeouts_done = 0;
706     char buf[20]; //used to receive stuff
707     bool QUIT = NO;
708
709     while (QUIT == NO && timeouts_done < link_layer_data.
        numTransmissions)
710     {
711         //done in read when disc is received
712         //write_UorS(APP_STATUS_RECEIVER, MESSAGE_DISC, 0, fd);
713
714         startAlarm(); //ready alarm for possible timeout
715
716         while (STOP == FALSE)
717         {
718             //get response
719             if ((res = read(fd, buf, 20)) < 0) perror("llclose_receiver
        :");
720
721             //update state machine and stop loop if valid
722             int i = 0;
723             for (; i < res && (state != STATE_MACHINE_STOP ||

```

```

724         received_C_type != MESSAGE_UA); ++i)
725             update_state_machine(APP_STATUS_TRANSMITTER,
726                                 APP_STATUS_RECEIVER, MESSAGE_UA, buf[i], &state);
727
728         //quit if received confirmation from transmitter
729         if (state == STATE_MACHINE_STOP && received_C_type ==
730             MESSAGE_UA)
731         {
732             stopAlarm(); QUIT = YES; break;
733         }
734     }
735 }
736
737 //END AND RETURN
738 //if did not receive confirmation UA
739 if (state != STATE_MACHINE_STOP || received_C_type != MESSAGE_UA)
740 {
741     printf("\nllclose: No confirmation of the reception was
742           received!");
743     if (timeouts_done >= link_layer_data.numTransmissions)
744         printf("\nllclose: Max tries (timeouts) reached.\n");
745     return -1;
746 }
747 //or else
748 printf("\nllclose: Connection was properly closed.\n");
749 return OK;
750 }
751
752 int llclose_transmitter(int fd)
753 {
754     int res;
755     state_machine_state state = STATE_MACHINE_START;
756     timeouts_done = 0;
757     char buf[20]; //used to receive stuff
758     bool QUIT = NO;
759
760     while (QUIT == NO && timeouts_done < link_layer_data.
761           numTransmissions)
762     {
763         write_UorS(APP_STATUS_TRANSMITTER, MESSAGE_DISC, 0, fd);
764
765         startAlarm(); //ready alarm for possible timeout
766
767         while (STOP == FALSE)
768         {
769             //get response
770             if ((res = read(fd, buf, 20)) < 0) perror("llclose_receiver
771               :");
772
773             //update state machine and stop loop if valid
774             int i = 0;
775             for (; i < res && (state != STATE_MACHINE_STOP ||
776                 received_C_type != MESSAGE_DISC); ++i)
777                 update_state_machine(APP_STATUS_TRANSMITTER,
778                                     APP_STATUS_RECEIVER, MESSAGE_DISC, buf[i], &state);
779
780             if (state == STATE_MACHINE_STOP && received_C_type ==
781                 MESSAGE_DISC)
782             {
783                 stopAlarm(); QUIT = YES; break;
784             }
785         }
786     }
787 }
788
789 //END AND RETURN
790 //if did not receive confirmation UA
791 if (state != STATE_MACHINE_STOP || received_C_type != MESSAGE_DISC)

```

```

782     {
783         printf("\nllclose: No confirmation of the reception was
              received!");
784         if (timeouts_done >= link_layer_data.numTransmissions)
785             printf("\nllclose: Max tries (timeouts) reached.\n");
786         return -1;
787     }
788     //or else
789     write_UorS(APP_STATUS_RECEIVER, MESSAGE_UA, 0, fd);
790     printf("\nllclose: Connection was properly closed.\n");
791     return OK;
792 }
793
794 #endif /*LLCLOSE AUXS*/
795
796 #endif
    //=====
797
798 //X=X=X=X    PROTOCOL MAIN FUNCS
799 #if (1)
800 //
    =====
801
802 int llopen(app_status_type status)
803 {
804     int fd;
805
806     if (open_tio(&fd, 0, 0) != OK) {
807         printf("\nERROR: Could not open terminal\n");
808         return -1;
809     }
810
811     occ_log.num_of_Is = 0;
812     occ_log.total_num_of_timeouts = 0;
813     occ_log.num_of_REJs = 0;
814     if (status == APP_STATUS_TRANSMITTER)
815     {
816         app_status = APP_STATUS_TRANSMITTER;
817         if( llopen_transmitter(fd) != OK) {
818             close_tio(fd);
819             return -1;
820         };
821     }
822     else if (status == APP_STATUS_RECEIVER)
823     {
824         app_status = APP_STATUS_RECEIVER;
825         if( llopen_receiver(fd) != OK) {
826             close_tio(fd);
827             return -1;
828         };
829     }
830     else
831     {
832         printf("\nWARNING(llo):invalid app_status found on llopen().\n"
              );
833         close_tio(fd);
834         return 1;
835     }
836     return fd;
837 }
838
839 // - - - - -
    - - - - -
840 #define DEBUG_LLREAD_WARN_UNEXPECTED_MSG 1
841 #define LLREAD_AUXREADBUFFER_SIZE 131085

```



```

842 #define LLREAD_AUXDATABUFFER_SIZE 1310850 //valor maximo necessario
    para 1 mensagem com dados: 131085 = (255 * 256 + 255 + 4 + 1) * 2 +
    5
843 // = (L2 * 256 + L1 + 4bytes_overhead_app_protocol + 1
    byte_bcc2_data_link_protocol) * 2_stuffing + 5
    bytes_overhead_data_link_protocol
844 //buffer must be dynamic!
845 int llread(int fd, char** buffer)
846 {
847     STOP = FALSE; /*doesn't do anything right now. could define timeout
        later 4 receiver*/
848     state_machine_state state = STATE_MACHINE_START;
849     state_machine_state prevstate = 0;
850     char auxReadBuf[LLREAD_AUXREADBUFFER_SIZE]; //aux buffer used to
        receive stuff on read
851     char auxReceiveDataBuf[LLREAD_AUXDATABUFFER_SIZE]; //aux buffer used
        to store data
852     int auxReceiveDataBuf_length = 0;
853     int res; //number of bytes read or sent
854
855     //devia usar 1 timeout pro read tb nao?
856
857     //bool RECEIVE = YES; //used to cycle
858     // - - - - -
859     //receive stuff
860     timeouts_done = 0;
861     while (timeouts_done < link_layer_data.numTransmissions)
862     {
863         startAlarm();
864         //STOP = FALSE;
865         while (STOP == FALSE) {
866             /*clear buf*/memset(auxReadBuf, 0,
                LLREAD_AUXREADBUFFER_SIZE);
867             res = read(fd, auxReadBuf, LLREAD_AUXREADBUFFER_SIZE);
868
869             //UPDATE STATE MACHINE - UPDATE STATE MACHINE - UPDATE
                STATE MACHINE
870             int i = 0;
871             //int lastStart = 0;
872             for (; i < res; ++i)
873             {
874                 prevstate = state;
875                 update_state_machine(APP_STATUS_RECEIVER,
                    APP_STATUS_TRANSMITTER, MESSAGE_I, auxReadBuf[i], &
                    state);
876                 //if (state == STATE_MACHINE_START;) lastStart = i;
877
878                 //receive data to auxReceiveDataBuf
879                 if (
880                     (prevstate == STATE_MACHINE_BCC_RCV || prevstate ==
                        STATE_MACHINE_ESCAPE)
881                     && (state == STATE_MACHINE_BCC_RCV || state ==
                        STATE_MACHINE_ESCAPE)
882                     && received_C_type == MESSAGE_I)
883                 {
884                     auxReceiveDataBuf[auxReceiveDataBuf_length] =
                        auxReadBuf[i];
885                     ++auxReceiveDataBuf_length;
886                 }
887
888                 if (state == STATE_MACHINE_STOP){
889
890                     if (received_C_type == MESSAGE_I) //RECEIVED DATA;
                        SEND RR or REJ; destuff and retrieve DATA
891                     {
892                         if (auxReceiveDataBuf_length == 0)
893                         {

```

```

894         printf("\nllread:empty I message received!
895                (? ? ?)");
896         stopAlarm();
897         return -1;
898     }
899     auxReceiveDataBuf_length = apply_destuffing(
900         auxReceiveDataBuf, auxReceiveDataBuf_length)
901     ;
902     if (validateBCC2(auxReceiveDataBuf,
903         auxReceiveDataBuf_length)==OK)
904     {
905         NR = received_C == IO? 1:0;
906         --auxReceiveDataBuf_length;//leave BCC2 out
907         of data
908         write_UorS(APP_STATUS_TRANSMITTER,
909             MESSAGE_RR, NR, fd);
910         //note: the original pointer must be
911         updated so a pointer to the pointer must
912         be used
913         if ((*buffer = (char*)malloc(
914             auxReceiveDataBuf_length)) == NULL)
915         {
916             perror("llread:");
917             stopAlarm();
918             return -1;
919         }
920         //memset(auxReadBuf, 0,
921             LLREAD_AUXREADBUFFER_SIZE);
922         memcpy(*buffer, auxReceiveDataBuf,
923             auxReceiveDataBuf_length);
924         /*update Occurrences_Log*/++occ_log.
925             num_of_Is;
926         stopAlarm();
927         return auxReceiveDataBuf_length;
928     }
929     else
930     {
931         write_UorS(APP_STATUS_TRANSMITTER,
932             MESSAGE_REJ, (received_C == IO ? 0 : 1),
933             fd);
934         state = STATE_MACHINE_START;
935         /*update Occurrences_Log*/++occ_log.
936             num_of_REJs;
937         auxReceiveDataBuf_length = 0;
938     }
939 }
940 //RECEIVED DISC ; SEND DISC BACK
941 else if (received_C_type == MESSAGE_DISC)
942 {
943     write_UorS(APP_STATUS_RECEIVER, MESSAGE_DISC,
944         0, fd);
945     stopAlarm();
946     return -2;//should notify the upper layer
947 }
948 //RECEIVED SET ; SEND UA BACK
949 else if (received_C_type == MESSAGE_SET)
950 {
951     write_UorS(APP_STATUS_TRANSMITTER, MESSAGE_UA,
952         0, fd);
953     state = STATE_MACHINE_START;
954     auxReceiveDataBuf_length = 0;
955 }
956 else

```

```

944         {
945             DEBUG_SECTION(DEBUG_LLREAD_WARN_UNEXPECTED_MSG,
                printf("\nllread:received unexpected msg of
                    type %d", received_C_type)););
946             state = STATE_MACHINE_START;
947             auxReceiveDataBuf_length = 0;
948             //return OK;
949         }
950     }
951 }
952 }
953 }
954 }
955 //-----
956
957 return -1;
958 }
959
960 /*
961 manda I
962 espera RR ou REJ
963 se RR sai
964 se REJ repete de inicio
965 */
966 #define DEBUG_LLWRITER_BADRR_R 0
967 #define DEBUG_LLWRITER_REJ 0
968 int llwrite(int fd, char * buffer, int length)
969 {
970     int res;
971     state_machine_state state = STATE_MACHINE_START;
972     timeouts_done = 0;
973     char buf[20]; //used to receive stuff
974     //bool QUIT = FALSE;
975
976     while ((*QUIT == NO &&*/ timeouts_done < link_layer_data.
        numTransmissions)
977     {
978         int num_of_writen_bytes = write_I(NS, fd, buffer, length);
979
980         startAlarm(); //ready alarm for possible timeout
981
982         while (STOP == FALSE)
983         {
984             //get respose
985             if ((res = read(fd, buf, 20)) < 0) perror("llwriter:");
986
987             //update state machine and stop loop if valid
988             int i = 0;
989             for (; i < res && (state != STATE_MACHINE_STOP); ++i)
990                 update_state_machine(APP_STATUS_TRANSMITTER,
                    APP_STATUS_TRANSMITTER, MESSAGE_RR, buf[i], &state);
991
992             if (state == STATE_MACHINE_STOP)
993             {
994                 if (received_C_type == MESSAGE_RR)
995                 {
996                     //check NR 2 c if it's ok to send next I
997                     if (received_C != (NS ? RR0 : RR1))
998                     {
999                         state = STATE_MACHINE_START;
1000                         stopAlarm();
1001                         //update statistics
1002
1003                         DEBUG_SECTION(DEBUG_LLWRITER_BADRR_R,
1004                             printf("\nllwriter debug: received a RR which C had
                                not expected R")););
1005

```



```

1069 }
1070
1071 #endif
    //=====

```

## A.5 DataLinkProtocol.h

```

1
2 #ifndef TYPEDEF_BOOLEAN_DECLARED_
3 #define TYPEDEF_BOOLEAN_DECLARED_
4 typedef char bool; //in case utilities is not included first...
5 #endif /* TYPEDEF_BOOLEAN_DECLARED_*/
6
7 #ifndef DATALINKPROTOCOL
8 #define DATALINKPROTOCOL
9
10 //
    =====
11 //basic definitions
12 //
    =====
13
14 //should be relative to a single transmission
15 struct occurrences_Log{
16     unsigned long num_of_Is; //sent if host(only counts when
        confirmation is received) | received by client
17     unsigned long total_num_of_timeouts;
18     unsigned long num_of_REJs; //received if host | sent if client
19 };
20
21 typedef struct occurrences_Log* occurrences_Log_Ptr;
22
23 typedef char app_status_type; //in case utilities is not included first
    ...
24 #define APP_STATUS_TRANSMITTER 0
25 #define APP_STATUS_RECEIVER 1
26
27 void set_basic_definitions(/*int timeout_in_seconds*/int
    number_of_tries_when_failing, char* port, int boudrate, int
    packetSize);
28
29 //self explanatory
30 occurrences_Log_Ptr get_occurrences_log();
31
32 //
    =====
33
34 //PROTOCOL MAIN FUNCS
35 //
    =====
36
37 int llopen(app_status_type status);
38
39 int llwrite(int fd, char * buffer, int length);
40
41 int llread(int fd, char** buffer);
42
43 int llclose(int fd, int hard);
44
45 #endif /* DATALINKPROTOCOL */

```

## A.6 FileFuncs.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "utilities.h"
4  #include "FileFuncs.h"
5
6  /*dest_buf must be freed outside!*/
7  long getFileBytes(char* filename, char** dest_buf)
8  {
9      FILE *pFile;
10
11      /*adapted from cplusplus.com examples*/
12
13      pFile = fopen(filename, "rb");//read,binary
14      if (pFile == NULL) { fputs("File error", stderr); exit(1); }
15
16      // obtain file size:
17      fseek(pFile, 0, SEEK_END);
18      long lSize = ftell(pFile);
19      rewind(pFile);
20
21      // allocate memory to contain the whole file:
22      *dest_buf = (char*)malloc(sizeof(char)*lSize);
23      if (*dest_buf == NULL) { perror("\ngetFileBytes(1):"); return -1; }
24
25      printf("\nCCC\n");
26
27      // copy the file into the buffer:
28      size_t result = fread(*dest_buf, 1, lSize, pFile);
29      if (result != lSize) {
30          perror("\ngetFileBytes(2):"); return -1;
31      }
32
33      //close file
34      if (fclose(pFile) != OK)
35      {
36          perror("\nsave2File:");
37          return -1;
38      }
39
40      return lSize;
41  }
42
43
44  int save2File(char* data, int data_size, const char* filename)
45  {
46      FILE *pFile;
47
48      //write: Create an empty file for output operations.
49      //If a file with the same name already exists, its contents are
50      //discarded and the file is treated as a new empty file.
51      if ((pFile = fopen(filename, "wb")) == NULL) //write,binary
52      {
53          perror("\nsave2File:");
54          return -1;
55      }
56
57      fwrite(data, sizeof(char), data_size, pFile); // write 10 bytes to
58      //our buffer
59
60      if (fclose(pFile) != OK)
61      {
62          perror("\nsave2File:");
63          return -1;
64      }
65  }
```

```

63
64     return OK;
65 }

```

## A.7 FileFuncs.h

```

1  #ifndef FILEFUNCS
2  #define FILEFUNCS
3
4  //copies file data to *dest_buf array. dest_buf must be dynamic and not
   initialized
5  //returns the length of the dest_buf
6  long getFileBytes(char* filename, char** dest_buf);
7
8  int save2File(char* data, int data_size, const char* filename);
9
10 #endif /*FILEFUNCS*/

```

## A.8 user\_interface.c

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <string.h>
5  #include "utilities.h"
6  #include "user_interface.h"
7  #include "FileFuncs.h"
8
9  //should have a start bigger than zero
10 int getIntPositiveRange(int start, int end) {
11     int num;
12     char get[50];
13     int done = NO;
14
15     while (!done) {
16         //printf("(%d,%d)", start, end);
17
18         gets(get);
19         //fgets(get, 50, stdin);
20         ///*clean buf*/char c; while ((c = getchar()) != '\n' && c !=
           EOF);
21         num = atoi(get);
22
23         if (num != 0 && start <= num && num <= end)
24             done = YES;
25         else printf("Invalid input/range, please select again:\n==>");
26     }
27
28     return num;
29 }
30
31 char getAnswer(int numOfChoices)
32 {
33     char get[20];
34     do {
35         gets(get);
36         //fgets(get, 2, stdin);
37         ///*clean buf*/char c; while ((c = getchar()) != '\n' && c !=
           EOF);
38         if (get[0] < 'a' || get[0] >= 'a' + numOfChoices || get[1] !=
           0)
39             printf("\nOption not recognized, please select again:");
40     } while (get[0] < 'a' || get[0] >= 'a' + numOfChoices || get[1] !=
           0);
41 }

```

```

42
43     return get[0];
44 }
45
46 char main_menu(bool receiver)
47 {
48     system("clear");
49
50     printf("\nWELCOME TO THE APP! PLEASE SELECT ONE OF THE FOLLOWING
51           OPERATONS");
52
53     if (receiver)
54         printf("\na) Establish Conection and receive picture");
55     else
56         printf("\na) Establish Conection and send picture");
57
58     printf("\nb) Try to re-establish lost conection");
59     printf("\nc) Change configurations");
60
61     if (receiver != 0) printf("\nd) Choose path to save picture");
62     else printf("\nd) Choose picture to send");
63
64     printf("\ne) Show Ocurrrences Log");
65
66     printf("\nf) Quit\n\n==>");
67
68     return getAnswer(6);
69 }
70
71 int select_config(void(*apply_options) (char, char, char, int))
72 {
73     char boudOpt, reconnectOpt, timetoutOpt;
74     int packetSize;
75
76     system("clear");
77
78     printf("Select baudrate:");
79     printf("\na)B0 - Hang Up      b)B50");
80     printf("\nc)B75                d)B110");
81     printf("\ne)B134                  f)B150");
82     printf("\ng)B200                  h)B300");
83     printf("\ni)B600                   j)B1200");
84     printf("\nk)B1800                 l)B2400");
85     printf("\nm)B4800                 n)B9600");
86     printf("\no)B19200                p)B38400 - Default");
87     printf("\nq)B57600                r)B115200");
88     printf("\ns)B230400               t)B460800\n\n==>");
89     boudOpt = getAnswer(20);
90
91     printf("\nSelect max Reconnect Tries: \na)1 \nb)3 \nc)5 \nd)7\n");
92     reconnectOpt = getAnswer(4);
93
94     /* printf("\nSelect timeout interval: \na)2 secs \nb)3 secs \nc)5 secs
95        \nd)8 secs \n==>");
96     timetoutOpt = getAnswer(4);
97 */
98
99     printf("\nInput packet size (number of file bytes per packet 1 -
100           65535):\n");
101     packetSize = getIntPositiveRange(1, 65535);
102
103     apply_options(boudOpt, reconnectOpt, timetoutOpt, packetSize);
104
105     return 0;
106 }

```



```

106
107
108
109 void show_prog_stats(unsigned long num_of_Is,
110 unsigned long total_num_of_timeouts,
111 unsigned long num_of_REJs, int appstatus)
112 {
113     system("clear");
114
115     printf("- - - Ocurrences log - - -");
116
117     if (appstatus/*receiver*/)
118         printf("\nNumber of Is received: %lu", num_of_Is);
119     else /*transmitter*/ printf("\nNumber of I's sent (RR confirmed): %
120         lu", num_of_Is);
121
122     printf("\nTotal number of timeouts: %lu", total_num_of_timeouts);
123
124     if (appstatus/*receiver*/)
125         printf("\nTotak number of REJs sent: %lu", num_of_REJs);
126     else /*transmitter*/printf("\nTotal number of REJs received: %lu",
127         num_of_REJs);
128
129     printf("\n- - - - -\nPress Enter key to return
130         to pevious menu...\n");
131
132     char get[20];
133
134     gets(get);
135     //fgets(get, 2, stdin);
136     ///*clean buf*/char c; while ((c = getchar()) != '\n' && c != EOF);
137 }
138
139 unsigned int selectNload_image(char** image_buffer, char*
140     out_image_name, unsigned char* out_image_name_length)
141 {
142     system("clear");
143     printf("Please input image file path (relative or not):\n>");
144
145     char get_path[100];
146
147     gets(get_path);
148     //fgets(get_path, 100, stdin);
149     ///*clean buf*/char c; while ((c = getchar()) != '\n' && c != EOF);
150
151     char* image_path;
152     while (TRUE){
153         image_path = realpath(get_path, NULL);
154         if (image_path != NULL) break;
155         printf("\nInvalid path/file. Please choose another.\n");
156
157         gets(get_path);
158         //fgets(get_path, 100, stdin);
159         ///*clean buf*/char c; while ((c = getchar()) != '\n' && c !=
160             EOF);
161     }
162
163     long imageSize;
164     if ((imageSize = getFileBytes(image_path, image_buffer)) < 0)
165     {
166         printf("\nFailed to load image.\n");
167         return -1;
168     }
169
170     //get name from path

```

```

168     image_path = strrchr(get_path, '/');
169     if (image_path==NULL)
170     {
171         strcpy(out_image_name, get_path);
172     }
173     else {
174         //(image_path - get_path + 1)
175         strcpy(out_image_name, image_path+1);
176     }
177
178     *out_image_name_length = strlen(out_image_name);
179
180     printf("\nImage sucessfully loaded.<%s , %ld bytes>\n",
181           out_image_name, imageSize);
182     sleep(2);
183     return (unsigned int) imageSize;
184 }

```

## A.9 user\_interface.h

```

1  #ifndef USER_INTERFACE
2  #define USER_INTERFACE
3
4  char main_menu(bool receiver);
5
6  /*@brief display menu configuration
7   * @param apply_options must be a pointer to a func that receives a
8   *       pointer to an array of 4 chars. The chosen options will be sent to
9   *       ths method that should apply them.
10 */
11 int select_config(void(*apply_options) (char, char, char, int));
12
13 //void* show_progress(void* args);
14
15 void show_prog_stats(unsigned long num_of_Is,
16                     unsigned long total_num_of_timeouts,
17                     unsigned long num_of_REJs, int appstatus);
18
19 unsigned int selectNload_image(char** image_buffer, char*
20                               out_image_name, unsigned char* out_image_name_length);
21
22 #endif /*USER_INTERFACE*/

```

## A.10 Utilities.h

```

1  #ifndef UTILITIES
2  #define UTILITIES
3
4  // section: should be a definition created by the programmer that must
5  // be equal to zero to avoid running the debug code.
6
7  #define DEBUG_SECTION(SECT, CODE) {\
8  if (SECT != 0)\
9  {\
10 CODE\
11 }\
12 }
13
14 #ifndef TYPEDEF_BOOLEAN_DECLARED_
15 #define TYPEDEF_BOOLEAN_DECLARED_
16 typedef int bool;
17 #endif /* TYPEDEF_BOOLEAN_DECLARED_*/

```

```

17
18 #define TRUE 1
19 #define YES 1
20 #define FALSE 0
21 #define NO 0
22 #define OK 0
23
24 #define PRINTBYTETOBINARY "%d%d%d%d%d%d%d%d"
25 #define BYTETOBINARY(byte)\
26 (byte & 0x80 ? 1 : 0),\
27 (byte & 0x40 ? 1 : 0),\
28 (byte & 0x20 ? 1 : 0),\
29 (byte & 0x10 ? 1 : 0),\
30 (byte & 0x08 ? 1 : 0),\
31 (byte & 0x04 ? 1 : 0),\
32 (byte & 0x02 ? 1 : 0),\
33 (byte & 0x01 ? 1 : 0)
34
35 #endif /* UTILITIES */

```

## B Tipos de Tramas Usadas

As tramas que utilizamos podem ser de três tipos:

- Informação (I) - transportam dados;
- Supervisão (S) - são usadas para iniciar e terminar a transmissão, assim como para responder a tramas do tipo I;
- Não Numeradas (U) - são usadas para responder a tramas de início e fim de transmissão.

Todas as tramas são delimitadas pelas *Flags* F - 01111110. Além disso, independentemente do tipo da trama, o cabeçalho é sempre o mesmo conjunto de 3 bytes:

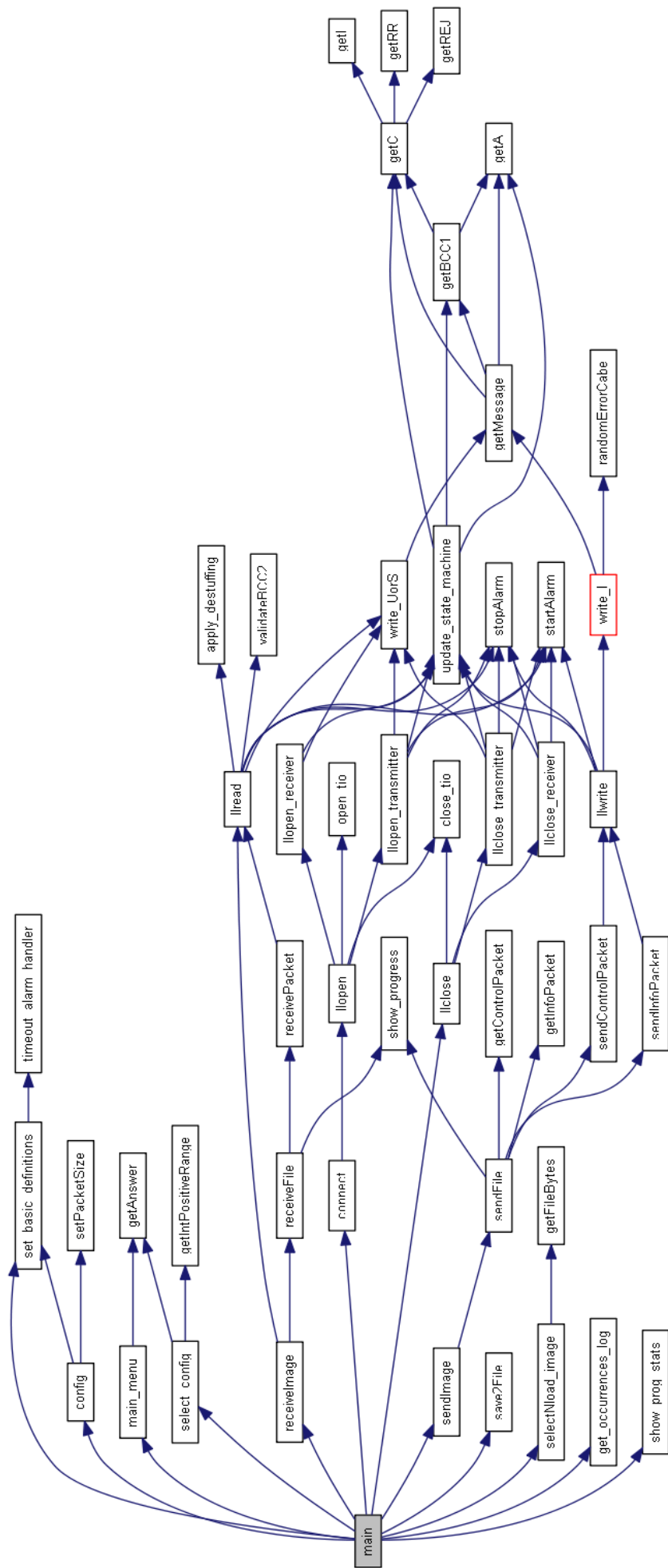
1. A - Campo de Endereço - 00000011 em comandos enviados pelo Emissor e respostas enviadas pelo Receptor, 00000001 na situação inversa;
2. C - Campo de Controle:
  - tramas I - 00S00000, onde S é o bit que identifica a trama;
  - tramas SET (*set up*) - 00000111;
  - tramas DISC (*disconnect*)- 00001011;
  - tramas UA (*unnumbered acknowledgment*) - 00000011;
  - tramas RR (*positive acknowledgment*) - 00R00001, onde R identifica a trama;
  - tramas REJ (*negative acknowledgment*) - 00R00101, onde R identifica a trama;
3. BCC1 (*Block Check Character*) - Campo de Proteção do Cabeçalho - é obtido realizando a disjunção exclusiva bit a bit de A e C.

Se se tratar de uma trama I, após o cabeçalho temos:

- $D_1, D_2, \dots, D_N$  - bytes de dados;
- BCC2 - Campo de Proteção dos Dados - calculado de forma que exista um número par de 1s em cada bit dos dados, incluindo o BCC2.

As restantes tramas apenas têm o respetivo cabeçalho delimitado por flags.

## C Diagrama de chamadas a funções (Fluxo)



## D Diagrama de Módulos

