Faculdade de Engenharia da Universidade do Porto

Mestrado Integrado em Engenharia Informática

Redes de Computadores

Ligação de Dados

Autores:
ei12135 - Paulo Faria
ei12161 - José Coutinho
ei12012 - Bruno Moreira
up201306843 - Pedro Moura

Relatório do
1º trabalho laboratorial

13 de Novembro de 2014

Sumário

O seguinte relatório foi realizado no âmbito do primeiro trabalho laboratorial da unidade curricular de Redes de Computadores e tem como tema a ligação de dados. Esta unidade curricular faz parte do curso Mestrado Integrado em Engenharia Informática da Faculdade de Engenharia Universidade do Porto.

Conteúdo

0.1 Introdução

O objetivo deste trabalho laboratorial é de implementar um protocolo de ligação de dados e de o testar posteriormente com uma aplicação simples de transferência de ficheiros.

Este relatório contém informação sobre o trabalho realizado, mais especificamente sobre:

- **Arquitetura**: blocos funcionais e interfaces.
- **Estrutura do Código**: APIs, principais estruturas de dados, principais funções e sua relação com a arquitetura.
- **Casos de uso principais**: identificação e sequências de chamada de funções.
- **Protocolo de ligação lógica**: identificação dos principais aspetos funcionais, descrição da estratégia de implementação com exemplos de código.
- **Validação**: descrição dos testes efectuados com apresentação quantificada dos resultados.
- **Elementos de valorização**: identificação dos elementos de valorização implementados e estratégia de implementação exemplos de código.
- **Conclusões**: síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

0.2 Arquitetura

O programa utiliza uma interface em modo de texto via terminal. Tanto o emissor como o receptor imprimem mensagens de controlo do fluxo e estado actual de transmissão do ficheiro.

O programa tem como interface os métodos *llopen*, *llwrite*, *llread* e *llclose*. Estes são utilizados no bloco principal do programa. Os métodos *llwrite* e *llread* estão mascarados pelo método *funcaoIO* de forma a dar ao programa principal uma vista mais simples.

Alguns blocos funcionais do programa, são por exemplo, a *maquinaEstados* que valida num controlo inicial a trama recebida e submete-a a outro método, o *alarm_handler* que é utilizado em todas as interrupções *SIGALRM* e contabiliza as tentativas de transmissão e timetouts.

0.3 Estrutura do código

API

- **int llopen(int porta, size_t flaggg)**: Abre a ligacao à porta /dev/ttyS'porta', inicializando o programa como transmissor/emissor (flaggg = 1) ou receptor (flaggg=0), retornando o descriptor no caso de sucesso ou -1 em caso de erro. Uma vez aberta a porta (em ambos os lados), o emissor começa por enviar uma trama de supervisão com campo de controlo SET e fica a espera de obter uma trama de supervisão com campo de controlo UA do receptor, de maneira a garantir que existe ligação.

- **int llclose(int fd)**: Termina a ligação "fd", retornando 0 em caso de sucesso ou -1 em caso de erro. O emissor envia uma trama de supervisão com campo de controlo DISC, de maneira a que este obtenha do receptor uma trama de supervisão com campo de controlo DISC. Uma vez obtida, o emissor envia outra trama de supervisão com campo de controlo UA e termina o programa. O receptor espera uma trama de supervisão com campo de controlo DISC do emissor, envia uma trama de supervisão com campo de controlo DISC ao emissor e espera que receba uma trama de supervisão com campo de controlo UA para terminar o programa.

- **int llwrite(int fd, char * buffer, int length)**: Envia para "fd"uma trama de informação composto pelos seus 6 bytes especificos mais o um pacote de dados resultante do algoritmo de sftuffing aplicado ao 'buffer' de tamanho 'length'. Retorna o tamanho da trama em caso de sucesso, ou -1 em caso de erro.

- **int llread(int fd, char * buffer)**:Recebe de "fd"uma trama de informação limitado por uma FLAG especificada pela camada de ligação de dados. Após obter a trama de informação, a função realiza o algoritmo oposto de stuffing ao pacote de dados contido da trama e assim obtem um pacote de dados 'buffer'. Retorna o tamanho de 'buffer' ou -1 em caso de erro.

Link Layer

- **int iFrame(unsigned char pkg[MTS], int size, unsigned char frame[MTS])**: insere o pacote na trama e procede ao stuffing.

- **int iFrameToPackage(unsigned char frame[MTS], int size, unsigned char pkg[MTS])**: procede ao unstuffing e separa o pacote da frame.

- **int suFrame(int emissor, int numSeq, int qualC, unsigned char frame[MTS])**: cria uma trama de supervisão com campo de controlo recebido por argumento.

Application Layer

- **void packageToControl(unsigned char pkg[MTS], int size, char * fileName, int * tamFile)**: despacota o pacote de controlo.

- **int controlToPackage(char * nomeFicheiro, int tamFicheiro, int modo, unsigned char pkg[MTS])**: cria um pacote de controlo com o modo inicio ou fim de transmissão, nome e tamanho do ficheiro.

- **int dataToPackage(unsigned char data[MFS+1], int numSeq, int size, unsigned char pkg[MTS])**: cria um pacote de dados com o número de sequência e tamanho especificados.

- **int packageToData(unsigned char pkg[MTS], int size, unsigned char data[MFS + 1])**: despacota o pacote de dados.

0.4 Casos de uso principais

Para melhorar a usabilidade do programa, assumimos que o computador **client** usa o */dev/ttyS4* e o **server** usa o */dev/ttyS0* e ainda que a string **send** determina que

esse computador está responsável pelo envio e que a string **get** determina que esse computador está responsável pela receção.

A sintaxe para iniciar o programa é: *./bin/main* ***computador funcao ficheiro***

Onde:

- **computador** : pode ser, ou **client** ou **server**
- **funcao** : pode ser, ou **send** ou **get**
- **ficheiro** : nome e extensão do ficheiro a transferir, este só deve ser introduzido se o computador utilizar a funcao **send**

Por exemplo:
O computador respetivo ao */dev/ttyS4* utiliza o comando *./bin/main client send pinguim.gif* e o computador respetivo ao */dev/ttyS0* utiliza o comando *./bin/main server get*.
Deve sempre existir um **send** e um **get** por transferência, por par de computadores.
A recepção, **get**, também permite três parâmetros opcionais ao programa, *baudrate*, *timeout* e *número de transmissões*.
Ambos os modos, **get** e **send**, estabelecem a ligação com o *llopen*.
Assim que entram na *funcaoIO* começam a ser tratados de forma diferente. O modo de envio, **send** começa a enviar as tramas de informação, primeiro com um pacote de controlo e depois com pacotes de dados. E o modo de leitura, **get**, irá receber essas tramas enviadas dando a resposta de acordo com a trama e estado actual no DFA.
Por fim, o modo de envio, **send**, desliga a ligação com o *llclose* e o modo de leitura, **get**, desliga-se assim que receber um pedido do modo de envio.

0.5 Protocolo de ligação lógica

As principais funções do protocolo são: *llopen*, *llread*, *llwrite* e *llclose*.

A função *llopen* estabelece a ligação à porta série, guarda os settings actuais da porta, altera os settings e trocam tramas de supervisão de maneira a garantir que ambos os computadores utilizam o mesmo protocolo.
Em mais detalhe, o emissor, **send**, envia uma trama de supervisão com o campo de controlo SET e o recetor, **get**, ao receber essa trama envia uma trama de supervisão com o campo de controlo UA.
O emissor, **send**, dispõe de um mecanismo de retransmissão, com um timeout de três segundos, que é activado sempre que o recetor não responde dentro desse tempo limite. Quando este é activado é reenviada a mesma trama.
Se isto é verificado com sucesso, então é dada a ligação como estabelecida e é retornado o descritor da porta série.

```
1 app.fileDescriptor = llopen(device,app.status);
```

A função *llclose* desliga a ligação à porta série, restaura os settings antigos da porta, altera os settings e trocam tramas de supervisão de maneira a garantir que ambos os computadores utilizam o mesmo protocolo.

Em mais detalhe, o emissor, **send**, envia uma trama de supervisão com o campo de controlo DISC e o recetor, **get**, ao receber essa trama envia uma trama de supervisão com o campo de controlo DISC. De seguida o emissor envia uma trama de supervisão com o campo de controlo UA e termina o programa. Assim que o recetor receber essa trama ele tambem desliga.

Se isto é verificado com sucesso, então é dada a ligação como desligada.

```
1 llclose(app.fileDescriptor);
```

Se isto é verificado com sucesso, então é dada a ligação como desligada.

A função llread chama a função leerSemAlarme() , e testa erros de bcc começando por processar uma mensagem de informação e consoante o retorno é enviado um RR ou REJ caso ocorra com sucesso ou insucesso respectivamente. Já a llwrite() envia para o ficheiro "fd"(porta série) uma trama criada apartir de um pacote (dados ou controlo) com tamanho 'tam', o qual é aplicado um algoritmo de stuffing, retornando o tamanho da trama em caso de sucesso; retorna -1 em caso de erro (timeout, numtrans max, ...).

0.6 Protocolo de aplicação

Cabe ao protocolo de aplicação a leitura/reconhecimento dos pacotes (receptor) , bem como o empacotamento desses dados que serão enviados(emissor). No receptor: Fica em espera até receber o pacote de controlo, como se ve aki no nosso codigo:

```
1 int tamFile = llread(app.fileDescriptor, PACOTE);
2
3 if (tamFile == -1)
4 {
5     printf("ERRO NA RECEPCAO DA TRAMA COM O PACOTE DE
    CONTROLO 0x02\n");
6     return(-1);
7 }
```

De seguida descompacta o pacote de controlo, obtendo o nome do ficheiro e o tamanho desse mesmo ficheiro iniciando a recepção das tramas de informação e consequente descompactação do pacote de dados até à recepção à aplicação do pacote de controlo que sinalize o fim da transferência. No emissor: Envia o pacote de controlo de inicio da transferência do ficheiro, abre o ficheiro a transmitir e então a aplicação lê a linha e cria um pacote de dados com o numero de sequência e tamanho especificados que será enviado para o receptor. Quando a totalidade do ficheiro é transmitida é então enviado o pacote de controlo de fim da transferência do ficheiro.

0.7 Validação

Condições de teste e respetivos resultados:

- Computador portátil com uso da ferramenta *SOCAT*
  - Transferência com sucesso quando ininterrupta.
- Computadores do laboratório com uso de cabo de série
  - Transferência com sucesso quando ininterrupta.

- Transferência com sucesso com interrupções, desligando e ligando o cabo de série.
- Transferência com sucesso com interrupções e introdução de ruído, com uso de uma chave na porta série.
- Transferência com sucesso para tamanhos superiores a 1 *tbyte*.
- Transferência com sucesso para *baudrates* diferentes.
- Transferência com sucesso para tipos de ficheiro diferente.

O segundo teste foi corroborado na avaliação do trabalho práctico com a presença do professor das práticas.

Para além disto foi averiguado, passando com sucesso, para um tamanho de ficheiro errado, se seria detetado o erro no fim do programa. Perante os testes realizados também foi possível verificar e contabilizar as tramas de rejeição enviadas.

0.8 Elementos de valorização

Foram implementados os seguintes elementos de valorização:

- **Implementação de REJ**: durante a verificação da consistência de uma trama de informação, assim que é verificado um erro nessa trama deve ser enviada uma trama de supervisão campo de controlo REJ para fazer um pedido de retransmissão da última trama enviada.

```
tamResposta = enviarTramaI(app.fileDescriptor, TRAMA,
    tamTrama+1, &r, TRAMA2, Crr, Crej);
```

- **Selecção de parâmetros pelo utilizador**: o utilizador pode configurar os parâmetros da aplicação.

```
if (argc >= 5) // baudrate
{
  int n = toInt(argv[4]);
  lnk.baudrate = (n == 0) ? 1 : n;
}
if (argc >= 6) // timeout
{
  int n = toInt(argv[5]);
  lnk.timeout = (n == 0) ? 1 : n;
}
if (argc >= 7) // num_trans
{
  int n = toInt(argv[6]);
    lnk.numTransmissions = (n == 0) ? 1 : n;
}
```

- **Registo de ocorrências**: é indicado ao utilizador o estado actual da transferência, utilizando o método *mostrarEstadoTransferencia* e ainda o método *estatistica* que fornece informação da transmissão e ficheiro.
- **Verificação da integridade dos dados pela Aplicação**: Conseguimos retomar a transmissão quer após a introdução de ruido, quer após a remoção e posterior inserção do cabo de série.

0.9 Conclusões

Em suma conseguimos realizar na sua totalidade e com sucesso este 1º trabalho.

Aprendemos a estabelecer um protocolo de comunicação, a estabelecer uma transmissão de dadados e a proteger os dados de erros de transmissão.

Poderiamos ter implementado a Geração aleatória de erros em tramas de Informação como elemento de valorização mas não foi possível no tempo útil.

0.10 Anexos

0.10.1 definicoes.h

Listing 1: ficheiro definicoes PONTO h.

```
#ifndef DEFINICOES_H
#define DEFINICOES_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define BAUDRATE B9600
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define TIMEOUT_LIMIT 3
#define NUM_TRANSMITIONS 15

//int BAUDRATE = B9600;
//unsigned int TIMEOUT_LIMIT = 3;
//unsigned int NUM_TRANSMITIONS = 15;

#define TRANSMITTER 1
#define RECEIVER 0

#define FALSE 0
#define TRUE 1

#define FLAG 0x7e
#define ESC 0x7d
#define Fstuffing 0x5e
#define ESCstuffing 0x5d

#define A1 0x03
#define A2 0x01

#define C_S_0 0x00
#define C_S_1 0x40

#define C_SET 0x03
#define C_UA 0x07
#define C_DISC 0x0b

#define C_RR_0 0x05
#define C_RR_1 0x85
#define C_REJ_0 0x01
#define C_REJ_1 0x81

#define MFS 500
#define MTS 1010

#define Cpkg_dados 0x01
#define Cpkg_inicio 0x02
#define Cpkg_fim 0x03

/*
    ********************************************************************************
```

```
  ESTRUTURAS DE DADOS E SUAS FUNCOES
*******************************************************************************
    */

/*  Application */
typedef struct _app
{
    int fileDescriptor;
    int status;
} AppLayer;

void setAppLayer(AppLayer * app, int f, int s)
{
    app->fileDescriptor = f;
    app->status = s;
}

void initAppLayer(AppLayer * app)
{
  app->fileDescriptor = 0;
  app->status = 0;
}

/*  Link */
typedef struct _link
{
    int baudrate; // velocidade de transmissao
    unsigned int timeout; // valor do temporizador (
    segundos): ex: 3
    unsigned int numTransmissions; //numero de tentativas
    em caso de falha
} LinkLayer;

void setLinkLayer(LinkLayer * link, int b, unsigned int t,
    unsigned int u)
{
   link->baudrate = b;
   link->timeout = t;
   link->numTransmissions = u;
}

void initLinkLayer(LinkLayer * link)
{
   link->baudrate = BAUDRATE;
   link->timeout = TIMEOUT_LIMIT;
   link->numTransmissions = NUM_TRANSMITIONS;
}

/*
    *******************************************************************************

  FUNCOES AUXILIARES AO TRATAMENTO DE DADOS
*******************************************************************************
    */
int tamanhoFicheiro(char * str)
{
  int r = 0;

  FILE * f;
```

```
104   f = fopen(str, "rb");
105
106   if (f == NULL)
107     return(-1);
108
109   fseek(f, 0L, SEEK_END);
110   r = ftell(f);
111
112   fclose(f);
113   return(r);
114 }
115
116 int numDigitos(int a)
117 {
118   int s = 0;
119   int v = a;
120   while(v > 0)
121   {
122     v /= 10;
123     s++;
124   }
125   return(s);
126 }
127
128 char * toString(int num) // contem caracteres latinos,
      terminando em '\0'
129 {
130   int dig = numDigitos(num);
131   char * r;
132   r = (char*) malloc(sizeof(char) * (dig+1) );
133
134   int i;
135   int v = num;
136
137   for(i = 0; i <= dig; i++)
138   {
139     r[i] = '\0';
140   }
141
142   for (i = (dig-1); i >= 0; i--)
143   {
144     int x = v % 10;
145     v /= 10;
146
147     r[i] = (char)x;
148   }
149
150   return(r);
151 }
152
153 int numCorrespondente(char n)
154 {
155   if (n == '0')
156     return(0);
157   if (n == '1')
158     return(1);
159   if (n == '2')
160     return(2);
161   if (n == '3')
```

```
162     return(3);
163   if (n == '4')
164     return(4);
165   if (n == '5')
166     return(5);
167   if (n == '6')
168     return(6);
169   if (n == '7')
170     return(7);
171   if (n == '8')
172     return(8);
173   if (n == '9')
174     return(9);
175 }
176
177 int toInt(char num[12])
178 {
179   int r = 0;
180   int i;
181   for (i = 0; num[i] != '\0'; i++)
182   {
183     r = (r * 10) + numCorrespondente(num[i]);
184   }
185
186   return(r);
187 }
188
189 void colocarFicheiro(FILE * ficheiro, unsigned char *
      content, int size) // ficheiro aberto em modo "w" ou "w
      +"
190 {
191   int k;
192   for (k = 0; k < size; k++)
193   {
194     fputc(content[k], ficheiro);
195   }
196 }
197
198 void verDados(unsigned char * str, int size)
199 {
200   int i ;
201   for (i = 0; i <= size; i++)
202   {
203     printf("0x%X ", str[i]);
204     if ( (i + 1) % 25 == 0)
205       printf("\n");
206   } printf("\n\n");
207 }
208
209 unsigned char xoor(unsigned char dados[MTS], int xmin, int
      xmax) // obtem bcc2 da trama entre xmin e xmax
210 {
211   //printf("INSIDE XOR FUNCTION: XOR[0]: 0x%X | XOR[SIZE]:
      0x%X\n", dados[xmin], dados[xmax]);
212   //printf("INSIDE XOR FUNCTION: XOR[0]: %d | XOR[SIZE]: %d
      \n", xmin, xmax);
213   unsigned char r = dados[xmin];
214   int i;
215   for (i = (xmin+1); i <= xmax; i++)
```

```
216   {
217     r = (r ^ dados[i]);
218   }
219   return(r);
220 }
221
222 void copiarPara(unsigned char src[MTS], unsigned char dst[
      MTS], int tam)
223 {
224   int i;
225   for(i = 0; i<=tam; i++)
226   {
227     dst[i] = src[i];
228   }
229 }
230
231 void limparFrame(unsigned char frame[MTS])
232 {
233   int i;
234   for (i = 0; i < MTS; i++)
235   {
236     frame[i] = '\0';
237   }
238 }
239
240 void limparPacote(unsigned char pkg[MTS])
241 {
242   int i;
243   for (i = 0; i < MTS; i++)
244   {
245     pkg[i] = '\0';
246   }
247 }
248
249 void limparDados(unsigned char data[MFS + 1])
250 {
251   int i;
252   for (i = 0; i < (MFS + 1); i++)
253   {
254     data[i] = '\0';
255   }
256 }
257
258 void limparNomeFicheiro(char file[30])
259 {
260   int i;
261   for (i = 0; i < 30; i++)
262   {
263     file[i] = '\0';
264   }
265 }
266
267 void mostrarEstadoTransferencia(float percentagem)
268 {
269   int numBarras = 10;
270   int numBarrasCheias = (int)(percentagem / (1.0 *
      numBarras));
271
272   printf(" [");
```

```
273
274   int i = 1;
275   for (i = 1; i <= numBarrasCheias; i++)
276     printf("%%");
277
278   for (; i <= numBarras; i++)
279   {
280     printf(" ");
281   } printf("]");
282
283   printf(" %.3f%%\n", percentagem);
284 }
285
286 /*
      ********************************************************************************

287   FUNCOES TRATAMENTO DE DADOS
288 ********************************************************************************
      */
289 int iFrameToPackage(unsigned char frame[MTS], int size,
      unsigned char pkg[MTS]);
290 int iFrame(unsigned char pkg[MTS], int size, unsigned char
      frame[MTS]);
291 void suFrameExtract(unsigned char frame[MTS]);
292 int suFrame(int emissor, int numSeq, int qualC, unsigned
      char frame[MTS]); // set, ua, disc, rr, rej
293
294 void packageToControl(unsigned char pkg[MTS], int size,
      char * fileName, int * tamFile);
295 int controlToPackage( char * nomeFicheiro, int tamFicheiro,
       int modo, unsigned char pkg[MTS]);
296 int dataToPackage(unsigned char data[MFS+1], int numSeq,
      int size, unsigned char pkg[MTS]);
297 int packageToData(unsigned char pkg[MTS], int size,
      unsigned char data[MFS + 1]);
298 /*=============================================================
299 *   FUNCOES AO NIVEL DO LINK LAYER
300 =============================================================*/
301 int iFrameToPackage(unsigned char frame[MTS], int size,
      unsigned char pkg[MTS])
302 {
303   //printf("-------------------------\nFRAME I UNBUILDER\n
      -------------------------\n");
304
305   //printf("Tamanho da trama: %d\n", size);
306   //verDados(frame, size);
307
308   limparPacote(pkg);
309
310   unsigned char N = frame[0];
311   unsigned char A = frame[1];
312   unsigned char C = frame[2];
313   unsigned char Bcc1 = (A ^ C);
314
315   //printf("[0x%X, 0x%X, 0x%X, 0x%X, ..., 0x%X, 0x%X]\n", N
      , A, C, Bcc1, frame[size-1], frame[size]);
316
317   // if (Bcc1 != frame[3]) printf("errou\n");
318
```

```
319   //printf("Tamanho do trama I: %d\n", size);
320
321   int limite = size - 2;
322   int i;
323   int j = 0;
324
325   //printf("[FIRST, LAST] = [0x%X, 0x%X]\n", frame[4],
        frame[limite]);
326
327   for (i = 4; i <= limite; i++)
328   {
329     if (frame[i] == ESC)
330     {
331       i++;
332       if (frame[i] == 0x5e) // flag
333       {
334         pkg[j] = 0x7e;
335         j++;
336       }
337       else if(frame[i] == 0x5d) // esc
338       {
339         pkg[j] = 0x7d;
340         j++;
341       }
342     }
343     else
344     {
345       pkg[j] = frame[i];
346       j++;
347     }
348
349     //pkg[i - 4] = frame[i];
350   }
351
352   //verDados(pkg, (*tamPacote));
353
354   return( j - 1 );
355 }
356
357 int iFrame(unsigned char pkg[MTS], int size, unsigned char
      frame[MTS])
358 {
359   limparFrame(frame);
360   //printf("--------------------------\nFRAME I BUILDER\n
        --------------------------\n");
361   //printf("Tamanho do pacote: %d\n", size);
362   //verDados(pkg, size);
363
364   int si = 0; //size of trama
365
366   frame[0] = FLAG;
367   frame[1] = 0x03;
368
369   if(pkg[0] == 0x01) // pacote de dados
370   {
371     int num = (int) pkg[1];
372     if ( num % 2 == 0) // se pkg.N par
373     {
374       frame[2] = 0x00;
```

```
375       }
376       else
377       {
378         frame[2] = 0x40;
379       }
380     }
381     else // pacote de controlo
382     {
383       frame[2] = 0x00;
384     }
385
386     // bcc1
387     frame[3] = (frame[1] ^ frame[2]);
388
389     int i;
390     int b = 4;
391
392     for (i = 0; i <= size; i++)
393     {
394       if (pkg[i] == ESC) // ESC
395       {
396         frame[b] = ESC;
397         b++;
398         frame[b] = 0x5d;
399         b++;
400       }
401       else
402       {
403         if (pkg[i] == FLAG) // FLAG
404         {
405           frame[b] = ESC;
406           b++;
407           frame[b] = 0x5e;
408           b++;
409         }
410         else
411         {
412           frame[b] = pkg[i];
413           b++;
414         }
415       }
416
417       // stuffing
418       // trama[i + 4] = pkg[i];
419     }
420     i = 5 + size;
421
422     // bcc2
423     frame[b] = xoor(frame, 4, b - 1); //xor(pkg, 0, (size));
424     //printf("RESULTDO DO XOR: 0x%X\n", frame[b]);
425     b++;
426
427     frame[b] = FLAG;
428     b++;
429
430     //printf("Tamanho trama I: %d\n", (*tamTrama));
431
432     //verDados(trama, *tamTrama);
433
```

```
434   //frameUnbuilderI(trama, tamTrama);
435
436   //printf("[0x%X, 0x%X, 0x%X, 0x%X, ..., 0x%X, 0x%X]\n",
        trama[0], trama[1], trama[2], trama[3], trama[(*
        tamTrama)-1], trama[(*tamTrama)]);
437
438   return(b - 1);
439 }
440
441 int suFrame(int emissor, int numSeq, int qualC, unsigned
      char frame[MTS])
442 {
443   //printf("-------------------------\nFRAME SU BUILDER\n
        -------------------------\n");
444   limparFrame(frame);
445
446   frame[0] = FLAG;
447
448   switch(qualC)
449   {
450     case 0: // SET -> comando
451       frame[2]  = 0x03;
452       frame[1] = (emissor == 1) ? 0x03 : 0x01;
453       break;
454     case 1: // DISC -> comando
455       frame[2]  = 0x0b;
456       frame[1] = (emissor == 1) ? 0x03 : 0x01;
457       break;
458     case 2: // UA -> resposta
459       frame[2]  = 0x07;
460       frame[1] = (emissor == 1) ? 0x01 : 0x03;
461       break;
462     case 3: // RR -> resposta
463       if ((numSeq % 2) == 0)
464       {
465         frame[2]  = 0x85;
466       }
467       else
468       {
469         frame[2]  = 0x05;
470       }
471       frame[1] = (emissor == 1) ? 0x01 : 0x03;
472       break;
473     case 4: // REJ -> resposta
474       if ((numSeq % 2) == 0)
475       {
476         frame[2]  = 0x81;
477       }
478       else
479       {
480         frame[2]  = 0x01;
481       }
482       frame[1] = (emissor == 1) ? 0x01 : 0x03;
483       break;
484   }
485   frame[3] = (frame[1] ^ frame[2]);
486   frame[4] = FLAG;
487
```

```
488   //printf("[0x%X, 0x%X, 0x%X, 0x%X, 0x%X]\n", trama[0],
        trama[1], trama[2], trama[3], trama[4]);
489
490   //verDados(trama, 5);
491
492   return(5);
493 }
494
495
496 void packageToControl(unsigned char pkg[MTS], int size,
        char * fileName, int * tamFile)
497 {
498   //printf("--------------------------\nCONTROL UNPACKET\n
        ----------------------------\n");
499
500   //printf("SIZE: %d\n", size);
501
502   unsigned char C = pkg[0];
503   unsigned char T1 = 0x00, T2 = 0x00;
504   int vec1 = 0, vec2 = 0;
505
506   int tamanhoTotal = 0;
507
508   (*tamFile) = 0;
509
510   unsigned char nomeFicheiro[255];
511
512   int i = 0, tf = 0, tn = 0;
513   int ref = 0;
514
515   T1 = pkg[1];
516   vec1 = (int) pkg[2];
517
518   ref = 3;
519   for (i = 0; i < vec1; i++)
520   {
521     int t = (int) pkg[i + ref];
522     (*tamFile) = (*tamFile) * 10 + t;
523   }
524
525   ref += vec1;
526   T2 = pkg[ref];
527   vec2 = (int) pkg[ref+1];
528   ref += 2;
529
530   for (i = 0; i < vec2; i++)
531   {
532     fileName[i] = (char)pkg[i + ref];
533   }
534   fileName[vec2] = '\0';
535
536   //printf("[0x%X, 0x%X, %d, ..., 0x%X, %d, ...]\n\n", C,
        T1, vec1, T2, vec2);
537   //printf("Tamanho do ficheiro: %d bytes\n", tamFicheiro);
538   //printf("Nome do ficheiro: %s\n", nomeFicheiro);
539
540   // armazenar tamanho e nome do ficheiro no lado do llread
        ()
541 }
```

```
542
543 int controlToPackage( char * nomeFicheiro, int tamFicheiro,
        int modo, unsigned char pkg[MTS])
544 {
545   //controlPacket(int modo, char * str_file, int tam_file)
546
547   //printf("==========\nCONTROL PACKET\n
        ----------\n");
548   int comprimentoFile = strlen(nomeFicheiro);
549   int comprimentoTamanho = numDigitos(tamFicheiro);
550
551   int i, j;
552   int total =
553     1 +
554     1 + 1 + comprimentoFile +
555     1 + 1 + comprimentoTamanho;
556
557   limparPacote(pkg);
558
559   i = 0;
560   if (modo == 2) // 0x01 -> inicio de transmissao
561   {
562     pkg[i] = 0x02;
563     i++;
564   }
565   else if (modo == 3)  // 0x02 -> fim de transmissao
566   {
567     pkg[i] = 0x03;
568     i++;
569   }
570
571   //printf("[0x%X, ", pkg[i-1]);
572
573   pkg[i] = 0x00; // tamanho do ficheiro
574   i++;
575
576   //printf("0x%X, ", pkg[i-1]);
577
578   pkg[i] = (char) comprimentoTamanho;
579   i++;
580
581   //printf("%d, ", comprimentoTamanho);
582
583   char * s = toString(tamFicheiro);
584
585   for (j = 0; j < comprimentoTamanho; j++)
586   {
587     pkg[i] = s[j];
588     i++;
589   }
590
591   //pkg[i - 3] = 0x00;
592
593   pkg[i] = 0x01; // nome do ficheiro
594   i++;
595
596   //printf("..., 0x%X, ", pkg[i-1]);
597
598   pkg[i] = (unsigned char) comprimentoFile;
```

```
599   i++;
600
601   for (j = 0; j < comprimentoFile; j++)
602   {
603     pkg[i] = nomeFicheiro[j];
604
605     i++;
606   }
607
608   //printf("%d, ...]\n", j);
609
610   //printf("Tamanho do pacote: %d\n\n", *tamPacote - 1);
611
612   return(i - 1);
613 }
614
615 int packageToData(unsigned char pkg[MTS], int size,
      unsigned char data[MFS + 1])
616 {
617   //printf("-------------------------\nDATA UNPACKET\n
      -------------------------\n");
618
619   //printf("Tamanho do pacote: %d\n", size);
620   //verDados(pkg, size);
621
622   limparDados(data);
623
624   unsigned char C = pkg[0];
625   unsigned char N = pkg[1];
626
627   unsigned char L2 = pkg[2];
628   unsigned char L1 = pkg[3];
629
630   int i = 4, j = 0;
631
632   int NN = N;
633   int LL2 = (int) L2;
634   int LL1 = (int) L1;
635
636   //printf("[C, N, L2, L1]: [0x%X, 0x%X, 0x%X, 0x%X]\n", C,
        NN, LL2, LL1);
637
638   int tamFragmento = LL2*MFS + LL1;
639   int tt = size - 4;
640
641   //printf("Tamanho do fragmento (aS): %d\n", tt);
642   //printf("Tamanho do fragmento teorico (LL2*MFS + LL1): %
      d\n", tamFragmento);
643
644   for (i = 4; i <= size; i++)
645   {
646     data[j] = pkg[i];
647     j++;
648   }
649
650   //printf("Tamanho do fragmento: %d\n\n", (*tamFrag));
651
652   //verDados(data, *tamFrag);
653
```

```
654   return(j - 1);
655 }
656
657 int dataToPackage(unsigned char data[MFS+1], int numSeq,
        int size, unsigned char pkg[MTS])
658 {
659   //printf("==========\nDATA PACKET\n
        ----------\n");
660   //printf("Tamanho do fragmento: %d\n", size);
661   //verDados(frag, size);
662
663   unsigned char C = Cpkg_dados;
664   unsigned char N = numSeq;
665
666   //verDados(frag, size+1);
667
668   int LL2 = size / MFS;
669   int LL1 = size - LL2 * MFS;
670   unsigned char L2 = (char)(LL2);
671   unsigned char L1 = (char)(LL1);
672
673   //printf("[C, N, L2, L1]: [0x%X, 0x%X, 0x%X, 0x%X]\n", C,
        N, L2, L1);
674
675   limparPacote(pkg);
676
677   int a, b = 4;
678
679   pkg[0] = C;
680   pkg[1] = N;
681   pkg[2] = L2;
682   pkg[3] = L1;
683
684   for (a = 0; a <= size; a++)
685   {
686     pkg[b] = data[a];
687     b++;
688   }
689
690   //printf("PKG[ultimo]: 0x%X\n\n", pkg[b-1]);
691
692   //printf("Tamanho do pacote: %d\n", *tamPacote);
693
694   //for(b = 0; b < *tamPacote; b++)
695   //{
696     //printf("0x%X ", pacote[b]);
697     //if (b % 10 == 0)
698       //printf("\n");
699   //} printf("\n\n");
700
701   //verDados(pkg, *tamPacote) ;
702
703   return(b - 1);
704 }
705
706 #endif
```

0.10.2 programa.h

Listing 2: ficheiro programa PONTO h.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>

#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>

#include <math.h>

#include "definicoes.h"

/*************************************
  VARIAVEIS GLOBAIS
*************************************/
struct termios oldtermios, newtermios;

int eEmissor;
int device;

char nomeFicheiro[30];
int tamFicheiro = 0;
FILE * f; // ficheiro a ler e guardar dados

AppLayer app;
LinkLayer lnk;

int contador = 0; //vezes que sooa o alarme
int num_trans = 0; // numero de transmissoes

unsigned int STOP = FALSE; // condicao de paragem nas
    tentativas de envio e leitura
unsigned int KEEP = 0; // condicao de manter a trama ou nao
    , segundo o bcc1
unsigned int flag = 1;

unsigned char TRAMA[MTS];
unsigned char TRAMA2[MTS];
unsigned char PACOTE[MTS];
unsigned char DADOS[MFS+1];

int tamTotal = 0;
int numTramasI_IO = 0, numTramasI_re = 0, numTimeouts = 0,
    numIO_REJ = 0;

/*************************************
  FUNCOES GLOBAIS
*************************************/
          /* ARRANQUE */

void setEmissorDevice(char * arg1, char * arg2)
{
```

```
53    if (strcmp(arg1, "server") == 0) // servidor pc local
54    {
55      device = 0; // aparelho destino
56    }
57    else // cliente pc local
58    {
59      device = 4;
60    }
61
62    printf("PROGRAMA PRINCIPAL :: ");
63    if (strcmp(arg2, "send") == 0) // pc local a querer ser o
        emissor
64    {
65      eEmissor = TRANSMITTER;
66
67      if (device == 4)
68        printf("CLIENTE :: ");
69      else
70        printf("SERVIDOR :: ");
71
72      printf("EMISSOR\n");
73    }
74    else // pc local a querer ser o receptor
75    {
76      eEmissor = RECEIVER;
77
78      if (device == 4)
79        printf("CLIENTE :: ");
80      else
81        printf("SERVIDOR :: ");
82
83      printf("RECEPTOR\n");
84    }
85    printf("==========\n"
        );
86  }
87
88  void set_newsettings(int fd, float vtime, int vmin) // mete
        settings novos na port
89  {
90      if ( tcgetattr(fd,&oldtermios) == -1) // save current
       port settings
91      {
92          perror("tcgetattr");
93          exit(1);
94      }
95
96      bzero(&newtermios, sizeof(newtermios));
97      newtermios.c_cflag = lnk.baudrate | CS8 | CLOCAL |
       CREAD;
98      newtermios.c_iflag = IGNPAR;
99      newtermios.c_oflag = 0; //OPOST;
100
101     /* set input mode (non-canonical, no echo,...) */
102     newtermios.c_lflag = 0;
103
104     //newtermios[VTIME]    = 0.1;   /* inter-character
       timer unused */
```

```
105     newtermios.c_cc[VTIME]    = vtime;//3   /* inter-
        character timer unused */
106     newtermios.c_cc[VMIN]     = vmin; //5  /* blocking read
         until 5 chars received */
107
108     /*
109      VTIME e VMIN devem ser alterados de forma a proteger
        com um temporizador a
110      leitura do(s) pr ximo(s) caracter(es)
111      */
112
113     tcflush(fd, TCIOFLUSH);
114
115     if ( tcsetattr(fd,TCSANOW,&newtermios) == -1) // set
        new port settings
116     {
117         perror("tcsetattr");
118         exit(2);
119     }
120 }
121
122 void set_oldsettings(int fd) // rep e os settings antigos
        da port
123 {
124     if ( tcsetattr(fd,TCSANOW,&oldtermios) == -1)
125     {
126         perror("tcsetattr");
127         exit(1);
128     }
129 }
130
131         /* LEITURA E ESCRITA DE DADOS */
132
133 int suuDFA(unsigned char trama[MTS], unsigned char
        Aesperado, unsigned char Cesperado)
134 {
135   int estado = 0;
136
137   int i;
138   for (i = 0; i < 5; i++)
139   {
140     switch(estado)
141     {
142       case 0:
143         if (trama[i] == FLAG)
144           estado = 1;
145         break;
146       case 1:
147         if (trama[i] == FLAG)
148           estado = 1;
149         else if (trama[i] == Aesperado)
150           estado = 2;
151         else
152           estado = 0;
153         break;
154       case 2:
155         if (trama[i] == FLAG)
156           estado = 1;
157         else if (trama[i] == Cesperado)
```

```
158             estado = 3;
159           else
160             estado = 0;
161           break;
162         case 3:
163           if (trama[i] == FLAG)
164             estado = 1;
165           else if (trama[i] == (Aesperado ^ Cesperado))
166             estado = 4;
167           else
168             estado = 0;
169           break;
170         case 4:
171           if (trama[i] == FLAG)
172             estado = 5;
173           else
174             estado = 0;
175       }
176     }
177
178     if (estado == 5)
179     {
180       return(0); // perfeito
181     }
182     else
183     {
184       return(-1);
185     }
186 }
187
188 int iiDFA(unsigned char trama[MTS], int size, unsigned char
        Aesperado, unsigned char Cesperado)
189 {
190     int estado = 0;
191
192     int i;
193     for (i = 0; i < 5; i++)
194     {
195       switch(estado)
196       {
197         case 0:
198           if (trama[i] == FLAG)
199             estado = 1;
200           break;
201         case 1:
202           if (trama[i] == FLAG)
203             estado = 1;
204           else if (trama[i] == Aesperado)
205             estado = 2;
206           else
207             estado = 0;
208           break;
209         case 2:
210           if (trama[i] == FLAG)
211             estado = 1;
212           else if (trama[i] == Cesperado)
213             estado = 3;
214           else
215             estado = 0;
```

```
216         break;
217       case 3:
218         if (trama[i] == FLAG)
219           estado = 1;
220         else if (trama[i] == (Aesperado ^ Cesperado))
221           estado = 4;
222         else
223           estado = 0;
224         break;
225     }
226   }
227
228   unsigned char bcc2 = xoor(trama, 4, size-2);
229
230   //printf("bcc2: 0x%X\n", bcc2);
231   //printf("BCC2: 0x%X\n", trama[size - 1]);
232   if (estado == 4) // bcc1 correcto
233     if (bcc2 == trama[size - 1]) // bcc2 correcto
234       return(0);
235     else
236       return(-2);
237   else
238     return(-1);
239 }
240
241 int maquinaEstados(unsigned char trama[MTS], int size,
      unsigned char Aesperado, unsigned char Cesperado)
242 {
243   if (size < 4)
244   {
245     printf("Tamanho inferior a 5!\n");
246     //tcflush(app.fileDescriptor, TCIOFLUSH);
247     return(-3);
248   }
249
250   if (trama[0] == trama[4]) // trama SU
251     return(suuDFA(trama, Aesperado, Cesperado));
252   else
253     return(iiDFA(trama, size, Aesperado, Cesperado));
254 }
255
256 void ctrlC_handler() // handler para o sinal SIGALRM
257 {
258   printf("\nCTRL C ACTIVADO\n");
259   set_oldsettings(app.fileDescriptor);
260   close(app.fileDescriptor);
261   exit(1);
262 }
263
264 void alarm_handler() // handler para o sinal SIGALRM
265 {
266   //STOP = FALSE;
267   contador++;
268   numTimeouts++;
269
270   flag = 1;
271   KEEP = 1;
272
273   if (contador != (lnk.numTransmissions))
```

```
274   {
275     printf("RETRANSMISSAO DA TRAMA PELA %do VEZ\n",
      contador);
276   }
277 }
278
279 // le constantemente, ate obter uma trama, utilizada no dfa
      e verifica se se aceita ou nao
280
281 int leerComAlarme(int fd, unsigned char trama[MTS])
282 {
283   tcflush(fd, TCIFLUSH);
284
285   STOP = FALSE;
286
287   unsigned char buf[5];
288   unsigned char LETRA = 0x00;
289   int tamT = 0;
290   int res = 0;
291   int aux = 0;
292
293   //printf("Entrou no lerComAlarme()\n");
294
295   //printf("%d %d\n", STOP, flag);
296
297   limparFrame(trama);
298   while(STOP == FALSE && !flag)
299   {
300     //printf("ola\n");
301
302     strcpy(buf, "");
303     res = read(fd, buf, 1);
304     //printf("cenas\n");
305
306     if(res == -1)
307     {
308       return(-1);
309     }
310     LETRA = buf[0];
311
312     if (tamT == 0) // nao ha conteudo no buf
313     {
314       if (LETRA == FLAG)
315       {
316         trama[tamT] = LETRA;
317         tamT++;
318       }
319     }
320     else
321     {
322       trama[tamT] = LETRA;
323       tamT++;
324
325       if (LETRA == FLAG)
326       {
327         //printf("STOP = TRUE\n");
328         if (tamT >= 4)
329           STOP = TRUE;
330         else
```

```
331         {
332           aux = 0;
333           for (aux = 1; aux <= tamT; aux++)
334           {
335             trama[aux] = '\0';
336             tamT = 1;
337           }
338         }
339       }
340     }
341     //printf("adeus\n");
342   }
343   //printf("tera sido adeus?\n");
344
345   if (flag == 1)
346   {
347     //printf("TIMEOUT\n");
348     return(-1);
349   }
350
351   // uma vez lida a trama, verificar erros
352   return(tamT);
353 }
354
355 int leerSemAlarme(int fd, unsigned char trama[MTS])
356 {
357   tcflush(fd, TCIOFLUSH);
358
359   STOP = FALSE;
360
361   unsigned char buf[5];
362   unsigned char LETRA;
363   int tamT = 0;
364   int res = 0;
365   int aux = 0;
366
367   limparFrame(trama);
368
369   while(STOP == FALSE)
370   {
371     strcpy(buf, "");
372     res = read(fd, buf, 1);
373     LETRA = buf[0];
374
375     if (tamT == 0) // nao ha conteudo no buf
376     {
377
378       if (LETRA == FLAG)
379       {
380         //printf("ARRANCOU\n");
381         trama[tamT] = LETRA;
382         tamT++;
383       }
384     }
385     else
386     {
387       //printf("0x%X ", LETRA);
388
389       //printf("%d ", tamT);
```

```
390         //if (tamT % 10 == 0) { printf("\n"); getchar(); }
391
392         trama[tamT] = LETRA;
393         tamT++;
394         if (LETRA == FLAG)
395         {
396           //printf("STOP = TRUE\n");
397           if (tamT >= 4)
398             STOP = TRUE;
399           else
400           {
401             for (aux = 1; aux <= tamT; aux++)
402             {
403               trama[aux] = '\0';
404               tamT = 1;
405             }
406           }
407         }
408       }
409     }
410     return(tamT);
411 }
412
413 int enviarr(int fd, unsigned char frame[MTS], int size)
414 {
415     tcflush(fd, TCOFLUSH);
416     int res = write(fd, frame, size);
417     //tcflush(fd, TCOFLUSH);
418
419     return(res);
420 }
421
422 int enviarTramaI(int fd, unsigned char frame[MTS], int size
      , int * rr_rej,
423     unsigned char frameLido[MTS], unsigned char C_RResperado,
        unsigned char C_REJesperado)
424 {
425     (void) signal(SIGALRM, alarm_handler);  // instala rotina
        que atende interrupcao
426
427      int estadoEnviar = 0;
428      int estadoReceber = 0;
429      flag = 1;
430    KEEP = 0;
431
432     //printf("0x%X 0x%X\n", Aesperado, Cesperado);
433     int estadoDFA_RR = -2;
434     int estadoDFA_REJ = -2;
435     int estadoDFA = 0;
436
437     numTramasI_re--;
438      while ( contador < lnk.numTransmissions )
439      {
440
441       numTramasI_re++;
442       //aqui
443       if ( flag == 1 )
444       {
445       //printf("Alarme activado\n");
```

```
446       alarm(lnk.timeout);
447
448       estadoEnviar = enviarr(fd, frame, size);
449
450       KEEP = 0;
451
452       //printf("Vai entrar no while do KEEP\n");
453
454       flag = 0;
455       while(KEEP == 0) // -1
456       {
457         KEEP = 0;
458         //printf("Ola %d %d\n", flag, KEEP);
459
460         estadoDFA = -2;
461         limparFrame(frameLido);
462         estadoReceber = leerComAlarme(fd, frameLido);
463         //printf("Ola %d %d\n", flag, KEEP);
464
465         //verDados(frameLido, estadoReceber);
466         //printf("Estado Receber: %d\n", estadoReceber);
467
468         if ( estadoReceber >= 0 ) // se leu alguma coisa
469         {
470           //printf("%d\n", estadoReceber);
471           //printf("ESTADO >= 0\n");
472             //printf("0x%X 0x%X\n", frameLido[0], frameLido
    [4]);
473
474           estadoDFA_RR = maquinaEstados(frameLido,
    estadoReceber, A1, C_RResperado);
475           estadoDFA_REJ = maquinaEstados(frameLido,
    estadoReceber, A1, C_REJesperado);
476
477           if (estadoDFA_RR == 0)
478           {
479             KEEP = 1;
480             alarm(0);
481             *rr_rej = 0; // rr
482             estadoDFA = 0;
483           }
484           else
485           {
486             if (estadoDFA_REJ == 0)
487             {
488               KEEP = 1;
489               alarm(0);
490               *rr_rej = 1; // rej
491               estadoDFA = 0;
492             }
493             else
494             {
495               KEEP = 0; // ignorar por serem -1
496             }
497           }
498         }
499         else
500         {
501           if (flag == 1) //timeout limit
```

```
502             {
503               //printf("aqui\n");
504               break;
505             }
506             else
507             {
508               printf("entras aqui ? \n"); //apagar esta linha
      antes de entregar o trabalho
509               KEEP = 0;
510             }
511           }
512         } // fim do ciclo while
513
514       if (estadoDFA == 0) break;
515
516       //printf("Vai sair fora do ciclo WHILE do KEEP\n");
517       }
518     }
519
520     if (contador == (lnk.numTransmissions))
521     {
522       printf("NUMERO DE RETRANSMISSOES COM TEMPORIZADOR
      EXCEDIDO\n");
523       *rr_rej = -1;
524       return -2; //-1 corresponde ao timeout limit
525       }
526
527     return(estadoReceber); // tamanho da trama SU de
      resposta
528 }
529
530 int enviarrComAlarme(int fd, unsigned char frame[MTS], int
      size,
531   unsigned char frameLido[MTS], unsigned char Aesperado,
      unsigned char Cesperado)
532 {
533   (void) signal(SIGALRM, alarm_handler);  // instala rotina
       que atende interrupcao
534     contador = 0;
535
536     int estadoEnviar = 0;
537     int estadoReceber = 0;
538     flag = 1;
539   KEEP = 0;
540
541   //printf("0x%X 0x%X\n", Aesperado, Cesperado);
542   int estadoDFA = -2;
543
544     while ( contador < lnk.numTransmissions )
545     {
546       //aqui
547       if ( flag == 1 )
548       {
549       //printf("Alarme activado\n");
550         alarm(lnk.timeout);
551
552         estadoEnviar = enviarr(fd, frame, size);
553
554         KEEP = 0;
```

```
555
556       //printf("Vai entrar no while do KEEP\n");
557
558       flag = 0;
559       while(KEEP == 0)
560       {
561         KEEP = 0;
562         //printf("Ola %d %d\n", flag, KEEP);
563
564       estadoDFA = -2;
565       limparFrame(frameLido);
566         estadoReceber = leerComAlarme(fd, frameLido);
567         //printf("Ola %d %d\n", flag, KEEP);
568
569         //verDados(frameLido, estadoReceber);
570         //printf("Estado Receber: %d\n", estadoReceber);
571
572         if ( estadoReceber >= 0 ) // se leu alguma coisa
573         {
574           //printf("%d\n", estadoReceber);
575           //printf("ESTADO >= 0\n");
576             //printf("0x%X 0x%X\n", frameLido[0], frameLido
    [4]);
577
578             estadoDFA = maquinaEstados(frameLido,
    estadoReceber, Aesperado, Cesperado);
579
580             if (estadoDFA == 0)
581             {
582               KEEP = 1;
583               alarm(0);
584             //printf("1\n");
585             }
586             else if (estadoDFA == -1)
587             {
588
589               KEEP = 0;
590             //printf("2\n");
591             }
592         }
593         else
594         {
595           if (flag == 1) //timeout limit
596           {
597             //printf("aqui\n");
598             break;
599           }
600           else
601           {
602             printf("entras aqui ? \n");
603             KEEP = 0;
604           }
605         }
606       } // fim do ciclo while
607
608     if (estadoDFA == 0) break;
609
610     //printf("Vai sair fora do ciclo WHILE do KEEP\n");
611     }
```

```
612     }
613
614     if (contador == (lnk.numTransmissions))
615     {
616      printf("NUMERO DE RETRANSMISSOES COM TEMPORIZADOR
        EXCEDIDO\n");
617      return -2; //-1 corresponde ao timeout limit
618      }
619
620     return(estadoReceber); // tamanho da trama SU de
        resposta
621 }
622
623 int enviarrSemAlarme(int fd, unsigned char frame[MTS], int
        size)
624 {
625   return(enviarr(fd, frame, size));
626 }
627
628 /*************************************
629   FUNCOES DA APLICACAO A UTILIZAR
630 *************************************/
631 int llopen(int porta, size_t flag);
632 int llwrite(int fd,  char * buffer, int length);
633 int llread(int fd,  char * buffer);
634 int llclose(int fd);
635 int funcaoIO();
636 void estatistica();
```

0.10.3 programa.c

Listing 3: ficheiro programa PONTO c.

```
#include "programa.h"

int main(int argc, char ** argv)
{
  system("clear");
  initAppLayer(&app);
  initLinkLayer(&lnk);

  if (argc >= 1) // ha argumentos
  {
    if (strcmp(argv[2], "send") == 0) // servidor deve ter 3 argumentos
    {
      if (argc < 4)
      {
        printf("MAIN :: SERVIDOR DEVE TER MINIMO 3 ARGUMENTOS\n");
        exit(1);
      }
      else
      {
        if (strcmp(argv[1], "server") != 0 && strcmp(argv[1], "client") != 0)
        {
          printf("MAIN :: %s NAO EXISTE\n", argv[2]);
          exit(1);
        }

        //nomeFicheiro = argv[3];
        strcpy(nomeFicheiro, argv[3]);
        tamFicheiro = tamanhoFicheiro(nomeFicheiro);

        if (tamFicheiro == -1)
        {
          printf("FICHEIRO %s NAO EXISTE\n", nomeFicheiro);
          return(-1);
        }

        if (argc >= 5) // baudrate
        {
          int n = toInt(argv[4]);
          lnk.baudrate = (n == 0) ? 1 : n;
        }
        if (argc >= 6) // timeout
        {
          int n = toInt(argv[5]);
          lnk.timeout = (n == 0) ? 1 : n;
        }
        if (argc >= 7) // num_trans
        {
          int n = toInt(argv[6]);
          lnk.numTransmissions = (n == 0) ? 1 : n;
        }
      }
    }
```

```
53      else if (strcmp(argv[2], "get") == 0)// cliente deve
        ter 2 argumentos
54      {
55        if (argc < 3)
56        {
57          printf("MAIN :: SERVIDOR DEVE TER MINIMO 2
          ARGUMENTOS\n");
58          exit(1);
59        }
60        else
61        {
62          if (strcmp(argv[1], "server") != 0 && strcmp(argv
          [1], "client") != 0)
63          {
64            printf("MAIN :: %s NAO EXISTE\n", argv[2]);
65            exit(1);
66          }
67
68          if (argc >= 4) // baudrate
69          {
70            int n = toInt(argv[3]);
71            lnk.baudrate = (n == 0) ? 1 : n;
72          }
73          if (argc >= 5) // timeout
74          {
75            int n = toInt(argv[4]);
76            lnk.timeout = (n == 0) ? 1 : n;
77          }
78          if (argc >= 6) // num_trans
79          {
80            int n = toInt(argv[5]);
81            lnk.numTransmissions = (n == 0) ? 1 : n;
82          }
83        }
84      }
85    }
86    else
87    {
88      printf("MAIN :: FALTAM ARGUMENTOS AO PROGRAMA PRINCIPAL
        \n");
89      exit(1);
90    }
91
92
93    /*************************************
94          INICIO DO PROGRAMA
95    *************************************/
96
97     setEmissorDevice(argv[1], argv[2]);
98    //printf("%d %u %u\n", lnk.baudrate, lnk.timeout, lnk.
        numTransmissions);
99    //printf("App.STATUS: %d | eEmissor: %d | DISPOSITIVO: %d
        \n", app.status, eEmissor, device);
100   //return(0);
101
102   //initLinkLayer(&lnk, device);
103
104   app.status = (eEmissor == 1) ? TRANSMITTER : RECEIVER ;
105     app.fileDescriptor = llopen(device,app.status);
```

```
106
107   if (app.fileDescriptor == -1)
108   {
109     return(-1);
110   }
111   else
112   {
113     printf(" :: LIGADO COM SUCESSO\n");
114   }
115   (void) signal(SIGINT, ctrlC_handler);  // instala rotina
       que atende interrupcao
116
117   /*************************************
118         TRATAMENTO DE DADOS
119   *************************************/
120   if (funcaoIO() < 0) return(-1);
121
122   /*************************************
123         TERMINAR O PROGRAMA
124   *************************************/
125   //sleep(1);
126
127   llclose(app.fileDescriptor);
128
129   printf(" :: DESLIGADO COM SUCESSO\n");
130
131   estatistica();
132
133   exit(0);
134 }
135
136 int funcaoIO()
137 {
138   printf("\nINICIANDO TRANSMISSAO :: ");
139
140   if (app.status == TRANSMITTER)
141   {
142     //printf(" :: EMISSOR :: ");
143
144     // envio do pacote de controlo de inicio da
       transferencia do ficheiro
145     sleep(1);
146     int tamPacote = 0;
147
148     limparPacote(PACOTE);
149     tamPacote = controlToPackage(nomeFicheiro, tamFicheiro,
       Cpkg_inicio, PACOTE);
150
151     int estEnvio = 0;
152     estEnvio = llwrite(app.fileDescriptor, PACOTE,
       tamPacote);
153
154     if (estEnvio < 0)
155     {
156       printf("ERRO , ENVIO DO PACOTE DE CONTROLO com 0x02\n
       ");
157       return(-1);
158     }
159
```

```
160     printf("SUCESSO , ENVIO DO PACOTE DE CONTROLO INICIAL\n
    ");
161
162     // enviar a informacao
163     int linhaActual = 0, i = 0;
164     f = fopen(nomeFicheiro, "rb");
165
166     unsigned char string[MFS];
167     strcpy(string, "");
168
169     int tamDados = 0;
170
171     int res = 0;
172
173     int aux = fgetc(f);
174     while(aux != EOF)
175     {
176       string[i] = (unsigned char)aux;
177       i++;
178
179       if (i == MFS) //string[i] ja contem os MFS caracteres
180       {
181         //printf("ola\n");
182         limparPacote(PACOTE);
183         tamPacote = dataToPackage(string, linhaActual, MFS,
    PACOTE);
184         //printf("Tamanho do pacote: %d\n", tamPacote);
185         //printf("ola 1.5\n");
186         res = llwrite(app.fileDescriptor, PACOTE, tamPacote
    );
187         if (res < 0)
188         {
189           printf("ERRO NO ENVIO DA TRAMA DE INFORMACAO
    NUMERO %d\n", linhaActual);
190           return(-1);
191         }
192
193         tamTotal += MFS;
194         numTramasI_IO++;
195
196         i = 0;
197         linhaActual++;
198         //printf("Linha %d\n", linhaActual);
199         printf(" :: DADOS #%2.d\n", linhaActual);
200         strcpy(string, "");
201       }
202       aux = fgetc(f);
203     }
204
205     limparPacote(PACOTE);
206     tamPacote = dataToPackage(string, linhaActual, i,
    PACOTE);
207     res = llwrite(app.fileDescriptor, PACOTE, tamPacote);
208
209     //printf("Resultdo llwrite(): %d\n", res);
210
211     tamTotal += i;
212
213     linhaActual++;
```

```
214     printf(" :: DADOS #%2.d\n", linhaActual);
215     numTramasI_IO++;
216 
217     fclose(f);
218 
219     sleep(1);
220 
221     // envio do pacote de controlo de fim da transferencia
        do ficheiro
222     limparPacote(PACOTE);
223     tamPacote = controlToPackage(nomeFicheiro, tamFicheiro,
         Cpkg_fim, PACOTE);
224 
225     estEnvio = 0;
226     estEnvio = llwrite(app.fileDescriptor, PACOTE,
        tamPacote);
227 
228     if (estEnvio < 0)
229     {
230       printf("ERRO , ENVIO DO PACOTE DE CONTROLO 0x03\n");
231       return(-1);
232     }
233     printf("SUCESSO , ENVIO DO PACOTE DE CONTROLO FINAL\n")
        ;
234   }
235   else
236   {
237     printf(" :: RECEPTOR :: ");
238 
239     limparPacote(PACOTE);
240     int tamFile = llread(app.fileDescriptor, PACOTE);
241 
242     //printf("Resultado do llread(): %d\n\n", tamFile);
243 
244     if (tamFile == -1)
245     {
246       printf("ERRO NA RECEPCAO DA TRAMA COM O PACOTE DE
        CONTROLO 0x02\n");
247       return(-1);
248     }
249 
250     printf("\nSUCESSO , RECEPCAO DO PACOTE DE CONTROLO");
251 
252     printf("\n");
253 
254 
255 
256     //verDados(informacaoFile, tamFile);
257     //if (nomeFicheiro == NULL)
258     //nomeFicheiro = (char *)malloc(sizeof(char) * 255);
259     limparNomeFicheiro(nomeFicheiro);
260     tamFicheiro = 0;
261 
262 
263     packageToControl(PACOTE, (tamFile), nomeFicheiro, &
        tamFicheiro);
264 
265     //printf("Nome ficheiro: %s\n", nomeFicheiro);
266     //printf("Tamanho do ficheiro: %d\n", tamFicheiro);
```

```
267
268     //if (tamanhoFicheiro(nomeFicheiro) == -1) printf("
    FICHEIRO %s NAO EXISTE\n", nomeFicheiro);
269
270
271     //nomeFicheiro = "galinha.gif";
272     //strcpy(nomeFicheiro, "galinha.gif");
273
274     // armazenar a informacao
275     f = fopen(nomeFicheiro, "wb+");
276     int tamPacote = 0;
277     int tamDados = 0;
278     float qqq = 0.0;
279
280     int res = 0;
281     int linha = 0;
282     while (tamTotal < tamFicheiro)
283     {
284       limparPacote(PACOTE);
285       limparDados(DADOS);
286       //setLinkLayer(&lnk, (lnk.sequenceNumber+1)%2);
287       tamPacote = llread(app.fileDescriptor, PACOTE);
288
289       if (tamPacote == -1)
290       {
291         printf("ERRO NA RECEPCAO DA TRAMA DE INFORMACAO
    NUMERO %d\n", linha+1);
292         return(-1);
293       }
294
295       linha++;
296       printf(" :: DADOS #%2.d :: ", linha);
297       numTramasI_IO++;
298
299       //verDados(PACOTE, tamPacote);
300       //printf("Tamanho llread(): %d\n", tamPacote);
301
302       tamDados = packageToData(PACOTE, tamPacote, DADOS);
303       //printf("Tamanho dados: %d\n", tamDados);
304
305       //verDados(data, tamDados);
306
307       colocarFicheiro(f, DADOS, tamDados);
308       //printf("Tamanho lido: %d\n", tamDados);
309       //if (pkg != NULL)
310       //free(pkg);
311
312       //if (data != NULL)
313       //free(data);
314       tamTotal += tamDados;
315
316       qqq = (100.0 * (1.0 * tamTotal) / (1.0*tamFicheiro));
317
318       //printf("---> TAMANHO ACTUAL FICHEIRO: %d %d [ ",
    tamActual, tamActual%MFS);
319       //system("clear");
320       //printf("\t\t\t\t\t[ %.3f%% ]\n", qqq);
321       mostrarEstadoTransferencia(qqq);
322     }
```

```
323
324     fclose(f);
325     //setLinkLayer(&lnk, 1);
326
327     // envio do pacote de controlo de inicio da
        transferencia do ficheiro
328     //printf("\n\n\nULTIMO PACOTE DE CONTROLO\n");
329     limparPacote(PACOTE);
330     tamFile = llread(app.fileDescriptor, PACOTE);
331
332     if (tamFile == -1)
333     {
334       printf("ERRO NA RECEPCAO DA TRAMA COM O PACOTE DE
        CONTROLO 0x03\n");
335       return(-1);
336     }
337
338     //char * fff = (unsigned char*) malloc( sizeof(unsigned
        char) * MFS );
339     char fff[30];
340     limparNomeFicheiro(fff);
341
342     unsigned int ttt = 0;
343
344     //nomeFicheiro = "pinguim.gif";
345     //strcpy(nomeFicheiro, "pinguim.gif");
346
347     packageToControl(PACOTE, (tamFile), fff, &ttt);
348
349     //printf("%s %d\n", fff, ttt);
350     int rrr = strcmp(fff, nomeFicheiro);
351
352     if (! (rrr == 0 && ttt == tamFicheiro) )
353     {
354       printf("ERRO NO PACOTE DE CONTROLO DE FIM DE
        TRANSFERENCIA\n");
355       return(-1);
356     }
357     printf("SUCESSO , RECEPCAO DO PACOTE DE CONTROLO FINAL\
        n");
358   }
359
360 }
361
362 void estatistica()
363 {
364   printf("\n\n ==========[ INFORMACAO DA TRANSMISSAO E
        FICHEIRO ]==========\n\n");
365
366   if (app.status == TRANSMITTER)
367   {
368
369     printf("NOME: %s\n", nomeFicheiro);
370     printf("TAMANHO ORIGINAL: %d\n", tamFicheiro);
371     //printf("TAMANHO RECEBIDO: %d\n", tamTotal);
372     printf("\n   TRAMAS I ENVIADAS: %d\n", numTramasI_IO);
373     printf("\n   TRAMAS I RENVIADAS: %d\n", numTramasI_re);
374     printf("\n   TRAMAS I REJEITADAS: %d\n", numIO_REJ);
375     printf("\n   TIMEOUTS: %d\n", numTimeouts);
```

```
376   }
377   else
378   {
379     //strcpy(nomeFicheiro, "galinha.gif");
380     printf("NOME: %s\n", nomeFicheiro);
381     printf("TAMANHO ORIGINAL: %d\n", tamFicheiro);
382     printf("TAMANHO RECEBIDO: %d\n", tamTotal);
383     printf("TAMANHO CRIADO: %d\n", tamanhoFicheiro(
      nomeFicheiro));
384     //strcpy(nomeFicheiro, "pinguim.gif");
385     printf("\n   TRAMAS I RECEBIDAS: %d\n", numTramasI_IO);
386     printf("\n   TRAMAS I REJEITADAS: %d\n", numIO_REJ);
387     printf("\n   TIMEOUTS: %d\n", numTimeouts);
388
389
390   }
391
392   printf("\t
      ------------------------------------------------------------\
      n");
393 }
394
395 /*
      ********************************************************************************

396 *                                DEFINICAO DAS FUNCOES A
      IMPLEMENTAR
397 ********************************************************************************
      */
398 int llopen(int porta, size_t flaggg)
399 {
400   printf("\nESTABELECENDO LIGACAO");
401
402   int r = 3; // diferente de 0 - read, 1 - write, 2 - erros
403   char portaa[11] = "/dev/ttyS0";
404
405   if (porta == 4)
406   {
407     portaa[9] = '4';
408   }
409   else if (porta != 0)
410   {
411     printf(" :: PORTA %d INVALIDA\n", porta);
412     return(-1);
413   }
414
415   r = open(portaa, O_RDWR | O_NOCTTY);
416   if (r == -1)
417   {
418     printf(" :: FALHOU A LIGACAO\n");
419     return(-1);
420   }
421
422   if (flaggg == TRANSMITTER) // llopen do TRANSMITTER
423   {
424     sleep(1);
425     //app.status = 1;
426     printf(" :: EMISSOR :: ");
427
```

```
428     set_newsettings(r, 0.1, 0.0); // 0.1 0
429
430     limparFrame(TRAMA);
431     int tamSU = suFrame(1, 0, 0, TRAMA); // int numSeq, int
         qualC
432
433     //verDados(TRAMA, tamSU);
434     limparFrame(TRAMA2);
435     int tamUA = enviarrComAlarme(r, TRAMA, tamSU, TRAMA2,
        A1, C_UA);
436
437     //verDados(TRAMA2, tamUA);
438
439     if (tamUA < 0)
440     {
441       printf("TIMEOUT LIMIT\n");
442       return(-1);
443     }
444
445     // trama UA e 'tramaUA' com tamanho 'estado'
446
447     //printf("Tamanho UA: %d\n", tamUA);
448
449     if (tamUA != 5)
450     {
451       printf("TAMANHO UA DIFERENTE DE 5. TAMANHO = %d\n",
      tamUA);
452       return(-1);
453     }
454   }
455   else if (flaggg == RECEIVER) // llopen do RECEIVER
456   {
457     //app.status = 0;
458     printf(" :: RECEPTOR :: ");
459
460     set_newsettings(r, 0.1, 1.0); // ------> 0.1 1.0
461
462     num_trans = 0;
463
464     int tamSET = 0;
465     int estadoDFA = 0;
466
467     do
468     {
469       limparFrame(TRAMA);
470       tamSET = leerSemAlarme(r, TRAMA);
471
472       //verDados(TRAMA, tamSET);
473       //TRAMA[1] = 0x45;
474       estadoDFA = maquinaEstados(TRAMA, tamSET, A1, C_SET);
475       num_trans++;
476
477       //printf("Contador: %d\n", contador);
478     }
479     while(num_trans <= lnk.numTransmissions && estadoDFA !=
         0);
480
481     if (num_trans > lnk.numTransmissions)
482     {
```

```
483         printf("NUMERO DE RETRANSMISSOES EXCEDIDAS\n");
484         return(-1);
485       }
486
487       //verDados(TRAMA, tamSET);
488       //printf("Tamanho SET: %d\n", tamSET);
489
490       if (estadoDFA < 0)
491       {
492         return(-1);
493       }
494
495       if (tamSET < 0)
496       {
497         printf("ERRO NA OBTENCAO DA TRAMA SET\n");
498         return(-1);
499       }
500
501       if (tamSET == 5)
502       {
503         printf("SET RECEBIDO , ENVIANDO UA\n");
504
505         limparFrame(TRAMA2);
506         int tamUA = suFrame(0, 1, 2, TRAMA2);
507
508         //verDados(TRAMA2, tamUA);
509
510         enviarr(r, TRAMA2, tamUA);
511       }
512       else // invalido
513       {
514         printf("SET INVALIDO , TAMANHO DIFERENTE %d != 5\n",
      tamSET);
515         return(-1);
516       }
517     }
518     else
519     {
520       printf(" :: FLAG INVALIDA\n");
521       return(-1);
522     }
523
524     return(r);
525   }
526
527   int llclose(int fd)
528   {
529     printf("\nDESLIGANDO LIGACAO");
530
531     int r = 0;
532
533     if (app.status == TRANSMITTER) // codigo do Emissor
534     {
535       printf(" :: EMISSOR :: ");
536
537       //printf("Codigo do emissor do llclose()\n");
538       limparFrame(TRAMA);
539       int tamDisc = suFrame(app.status, 0, 1, TRAMA); //
     tramaSU(int eEmissor, int qualCtrama, int 0 ou 1)
```

```
540
541     //verDados(tramaSET, 4);
542     printf("ENVIANDO DISC\n");
543     limparFrame(TRAMA2);
544     int tamDISC2 = enviarrComAlarme(app.fileDescriptor,
        TRAMA, 5, TRAMA2, A2, C_DISC);
545
546     //printf("TAMANHO DO TAMDISC2: %d\n", tamDISC2);
547
548     if (tamDISC2 < 0)
549     {
550       printf("TIMEOUT LIMIT\n");
551       return(-1);
552     }
553     printf(":: DISC RECEBIDO , ENVIANDO UA\n");
554
555     limparFrame(TRAMA);
556     int tamUA = suFrame(app.status, 0, 2, TRAMA); //
        tramaSU(int eEmissor, int qualCtrama, int 0 ou 1)
557     int res = enviarrSemAlarme(app.fileDescriptor, TRAMA,
        5);
558
559     //printf("TAMANHO DO TAMDISC2: %d\n", tamDISC2);
560   }
561   else // codigo do Receptor
562   {
563     printf(" :: RECEPTOR :: ");
564
565     num_trans = 0;
566
567     int tamDISC = 0;
568     int estadoDFA = 0;
569     //limparFrame(TRAMA);
570     do
571     {
572       limparFrame(TRAMA);
573       tamDISC = leerSemAlarme(app.fileDescriptor, TRAMA);
574
575       //TRAMA[1] = 0x45;
576       estadoDFA = maquinaEstados(TRAMA, tamDISC, A1, C_DISC
        );
577       num_trans++;
578       //printf("Contador: %d\n", contador);
579     }
580     while(num_trans <= lnk.numTransmissions && estadoDFA !=
        0);
581
582     if (num_trans > lnk.numTransmissions)
583     {
584       printf("NUMERO DE RETRANSMISSOES EXCEDIDAS\n");
585       return(-1);
586     }
587
588     if(estadoDFA != 0)
589     {
590       return(-1);
591     }
592
593     //printf("TAMANHO DO TAMDISC: %d\n", tamDISC);
```

```
594
595     //printf("Tamanho SET: %d\n", tamSET);
596
597     if (tamDISC < 0)
598     {
599       printf("ERRO NA OBTENCAO DA TRAMA DISC\n");
600       return(-1);
601     }
602
603     if (tamDISC != 5)
604     {
605       printf("DISC INVALIDO , TAMANHO DIFERENTE %d != 5\n"
      , tamDISC);
606       return(-1);
607     }
608
609     printf("DISC RECEBIDO , A ENVIAR DISC\n");
610     limparFrame(TRAMA);
611     int tamDISC2 = suFrame(app.status, 0, 1, TRAMA);
612
613     limparFrame(TRAMA2);
614     int tamUA = enviarrComAlarme(app.fileDescriptor, TRAMA,
       5, TRAMA2, A2, C_UA);
615
616     //printf("TAMANHO DO TAMUA: %d\n", tamUA);
617
618     if (tamUA < 0)
619     {
620       printf("ERRO NA OBTENCAO DA TRAMA UA\n");
621       return(-1);
622     }
623
624     if (tamUA != 5)
625     {
626       printf("TAMANHO UA DIFERENTE DE 5. TAMANHO = %d\n",
      tamUA);
627       return(-1);
628     }
629
630     printf(" :: UA RECEBIDO\n");
631   }
632
633   sleep(1);
634   set_oldsettings(fd);
635   close(fd);
636
637   return(0);
638 }
639
640 int llwrite(int fd,  char * buffer, int length) // (file,
      pck, tamPKG)
641 {
642   int r = 0;
643
644   limparFrame(TRAMA);
645   int tamTrama = iFrame(buffer, length, TRAMA);
646   int tamResposta = 0;
647   num_trans = 0;
648   contador = 0;
```

```
649
650   unsigned char Cs = TRAMA[2];
651   unsigned char Crr = 0x00, Crej = 0x00;
652
653   do
654   {
655     sleep(1);
656     if (Cs == C_S_0)
657     {
658       Crr = C_RR_1;
659       Crej = C_REJ_1;
660       r = 0;
661       //printf("llwrite() :: Ns = 0!!\n");
662     }
663     else if (Cs == C_S_1)
664     {
665       Crr = C_RR_0;
666       Crej = C_REJ_0;
667       r = 1;
668       //printf("llwrite() :: Ns = 1!!\n");
669     }
670     else
671     {
672       printf("Erro na trama: Caracter do C corresponde ao 0
      x%X\n", Cs);
673       return(-1);
674     }
675     if (contador > 0)
676       printf("RETRANSMISSAO DA TRAMA PELA %do VEZ\n",
      contador);
677
678     limparFrame(TRAMA2);
679     r = 0; //verificar se rr ou rej ou ignorar
680
681     tamResposta = enviarTramaI(app.fileDescriptor, TRAMA,
      tamTrama+1, &r, TRAMA2, Crr, Crej);
682
683     //printf("tamResposta: %d\n", tamResposta);
684
685     if (tamResposta < 0) // -> r = -1
686       return(tamResposta);
687
688     if (r == 1) //rej
689     {
690       //num_trans++;
691       numIO_REJ++;
692       contador++;
693     }
694   }
695   while( contador <= lnk.numTransmissions && r == 1); //rej
       ou contador
696
697   if (contador > lnk.numTransmissions)
698   {
699     printf("NUMERO DE RETRANSMISSOES EXCEDIDAS\n");
700     return(-1);
701   }
702
703   return(tamTrama);
```

```
704 }
705
706 int llread(int fd,  char * buffer)
707 {
708
709   int tamPacote = 0, tamTrama = 0, tamTrama2 = 0, res = 0;
710   int estadoDFA = 0, numSeq = 0;
711
712   num_trans = 0;
713   contador = 0;
714
715   unsigned char Cs, Cr;
716
717   do
718   {
719     tamTrama = 0;
720     numSeq = 0;
721
722     limparFrame(TRAMA);
723     tamTrama = leerSemAlarme(app.fileDescriptor, TRAMA);
724
725     limparFrame(TRAMA2);
726     copiarPara(TRAMA, TRAMA2, tamTrama); //copiar
      temporariamente para TRAMA2 para testar erros de bcc
727
728     //verDados(TRAMA, tamTrama);
729
730     //descomentar codigo abaixo para testar erro bcc1 /
      bcc2
731     //if (num_trans < 3)
732       //TRAMA2[7] = 0x23;
733
734     //printf("bife\n");
735     //printf("%d\n", tamTrama);
736     //printf("0x%X\n", TRAMA[0]);
737
738     Cs = TRAMA[2];
739     if (Cs == C_S_0)
740     {
741       Cr = C_RR_1;
742       numSeq = 0;
743     }
744     else if (Cs == C_S_1)
745     {
746       Cr = C_RR_0;
747       numSeq = 1;
748     }
749
750     //printf("0x%X 0x%X\n", A1, Cs);
751     estadoDFA = maquinaEstados(TRAMA2, tamTrama-1, A1, Cs);
752     //printf("Estado DFA llread(): %d\n", estadoDFA);
753
754
755     //limparPacote(PACOTE);
756     tamPacote = iFrameToPackage(TRAMA2, tamTrama-1, buffer)
      ; // <-- alterei aqui
757
758     limparFrame(TRAMA2);
759     if (estadoDFA == 0) // rr
```

```
760     {
761       //printf("llread() :: RR!!\n");
762       tamTrama2 = suFrame(app.status, numSeq, 3, TRAMA2);
      //rr
763     }
764     else if (estadoDFA == -2) // rej
765     {
766       //printf("llread() :: REJ!!\n");
767       tamTrama2 = suFrame(app.status, numSeq, 4, TRAMA2);
      //bcc2 erro -> rej
768       //num_trans++;
769       contador++;
770
771       numIO_REJ++;
772       numTramasI_re++;
773
774       //int numTramasI_IO = 0, numTramasI_re = 0,
      numTimeouts = 0, numIO_REJ = 0;
775     }
776     else
777     {
778       //num_trans++; // bcc1 erro
779       contador++;
780     }
781
782     res = enviarrSemAlarme(app.fileDescriptor, TRAMA2, 5);
783   }
784   while(contador < lnk.numTransmissions && estadoDFA != 0);
785
786   if (contador >= lnk.numTransmissions)
787   {
788     printf(" :: NUMERO DE RETRANSMISSOES EXCEDIDAS\n");
789     return(-1);
790   }
791
792   return(tamPacote);
793 }
```