

Redes de Computadores

1º Trabalho Laboratorial

11 de novembro de 2018

Beatriz Garrido up201504710

Diogo Santos up201505606

João Costa up201404935

*Índice*

**Sumário2**

**Introdução2**

**Arquitetura3**

**Estrutura do código3**

Camada de Ligação3

Camada de Aplicação 3

**Casos de uso principais4**

**Protocolo de ligação lógica4**

**Protocolo de aplicação**8

Emissor 8

Recetor 9

**Validação**9

**Elementos de valorização12**

**Conclusões13**

**Anexo I - Código fonte14**

appLayer.c14

dataLink.c18

dataLink.h29

**Anexo II – Estatísticas realizadas31**

*Sumário*

No âmbito da unidade curricular de Redes de Computadores, este relatório tem como objetivo a implementação do primeiro trabalho laboratorial, referente à transferência de dados. Este trabalho consiste no desenvolvimento de uma aplicação capaz de enviar ficheiros de um computador para outro através de uma porta série.

Dito isto, o trabalho foi realizado com sucesso, visto que os objetivos propostos foram concretizados e a aplicação foi desenvolvida na perfeição, sem qualquer perda de dados.

*Introdução*

O objetivo deste trabalho é implementar um protocolo de ligação de dados, de acordo com as especificações descritas no guião fornecido. Bem como testar o protocolo com uma aplicação simples de transferência de ficheiros, igualmente especificada.

Relativamente ao relatório, a sua função é explicar toda a lógica presente no trabalho, seguindo a seguinte estrutura:

* **Arquitetura:** exposição dos blocos funcionais e interfaces;
* **Estrutura do código:** identificação das APIs, principais estruturas de dados, principais funções e sua relação com a arquitetura;
* **Casos de uso principais:** identificação dos principais casos de uso e sequências de chamada de funções;
* **Protocolo de ligação lógica:** identificação dos principais aspetos funcionais e descrição da estratégia de implementação destes, com apresentação de extratos de código;
* **Protocolo de aplicação:** identificação dos principais aspetos funcionais e descrição da estratégia de implementação destes, com apresentação de extratos de código;
* **Validação:** descrição dos testes efetuados com apresentação quantificada dos resultados;
* **Elementos de valorização:** identificação dos pontos adicionais implementados e descrição da estratégia de implementação;
* **Conclusão:** síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados;

*Arquitetura*

Este trabalho está dividido em duas principais camadas lógicas, com a finalidade de uma melhor organização do protocolo e aperfeiçoamento da aplicação.

*Estrutura do código*

**Camada de Ligação**

A camada de ligação (ou *Data Link Layer*) proporciona os meios necessários para a transferência de dados entre camadas. Pode também fornecer códigos de deteção de erros, cajo estes ocorram na camada física.

Assim, esta categoria contém as funções necessárias para o estabelecimento de ligação (LLOPEN), escrita e leitura na porta série (LLWRITE e LLREAD respetivamente), fecho da ligação (LLCLOSE) e deteção de erros, como já referido anteriormente.

**Camada de Aplicação**

A camada de aplicação (ou *Application Layer*) é a que se encontra diretamente acima da camada acima mencionada, responsável pela comunicação entre o utilizador e a interface. É a partir desta que se inicia o processo de transferência de dados, sendo a que contém a função *main().*

*Casos de uso principais*

A aplicação desenvolvida pode ser corrida de duas maneiras diferentes:

**Como emissor**

A transmissão de um ficheiro para outro computador necessita de 2 parâmetros. Depois da referente a porta série (normalmente /dev/ttyS0) é necessário introduzir o tipo de comunicação – “*sender*” – e o nome do ficheiro que se deseja enviar.

De notar que, se após um determinado número de *timeouts*, nenhuma ligação for estabelecida por parte do recetor, é invocada a função de fecho de ligação.

**Como recetor**

A receção do ficheiro enviado apenas necessita do parâmetro referente ao tipo de comunicação – “*receiver*” – depois da introdução da porta série.

Ao contrário do emissor, esta comunicação não tem tempo limitado, visto que fica constantemente à espera que o emissor inicie a ligação.

*Protocolo de ligação lógica*

O protocolo de ligação lógica é implementado na camada de ligação (*data link Layer),* da qual depende a camada de aplicação (A*pplication Layer*).

Nesta camada são implementadas as seguintes funcionalidades:

* Estabelecer e terminar uma ligação através da porta série, bem como transmitir e receber dados/mensagens na mesma.
* Criar e enviar comandos (tramas I, SET e DISC) e respostas (UA, RR e REJ).
* Receber mensagens através da porta série.
* Fazer *stuff* e *destuff* de pacotes da camada de aplicação.
* Verificar se a informação recebida é igual à envida, através do cabeçalho em todas as tramas. Nas tramas I, além do cabeçalho usa-se uma variável de controlo: BCC2.

**LLOPEN**

**int** LLOPEN(**int** fd, **int** com\_type);

Função responsável por estabelecer a ligação através da porta série.

Quando esta função é chamada pelo emissor, este envia a trama SET e aguarda resposta do recetor, ou seja, a mensagem UA. Caso o recetor não receba uma resposta ao fim de 3 *timeouts,* cuja duração, até ser atingido, é definida através de um alarme,a ligação é fechada.

Quando esta função é chamada pelo recetor, a função fica à espera de receber a mensagem SET, sendo que, quando a recebe, envia como resposta a mensagem UA e a ligação é estabelecida. No caso do recetor não é implementado qualquer tipo de alarme, ficando este indefinidamente à espera de receber a mensagem SET.

**LLWRITE**

**int**LLWRITE(**int**fd, **char** \*buffer,**int**lenght);

Função responsável por tentar escrever na porta série, os dados através da função *write()* e fica a aguardar uma resposta. Caso não receba uma resposta ao fim de 3 *timeouts*, a ligação é fechada. Quando a resposta for recebida e, caso esta contenha a variável de controlo RR, a mensagem foi transmitida corretamente.

Caso, a resposta contenha a variável de controlo REJ, a mensagem não foi transmitida corretamente, o que implica a mensagem ser retransmitida e o valor de *timeouts* permitido restaurado para o valor inicial.

**LLREAD**

**int**LLREAD(**int**fd, **char** \*buffer);

Função responsável por tentar receber através da porta série os dados enviados.

Caso esta receba uma mensagem inválida, envia a variável de controlo REJ através da porta série. Caso a mensagem recebida for válida, a informação é guardada e é realizado o envio da variável de controlo RR através da porta série.

Caso receba a mensagem DISC, a ligação deve ser encerrada.

**LLCLOSE**

**int**LLCLOSE(**int**fd,**int**com\_type);

Função responsável por terminar a ligação através da porta série.

Caso seja o emissor a chamar esta função, a mensagem DISC é enviada pela porta série e este fica a aguardar pela receção da trama DISC. Depois de o receber, envia a mensagem UA.

Caso seja o recetor a chamar a função, aguarda pela receção da trama DISC e quando o receber reenvia-o e fica a aguardar pela receção da mensagem UA.

Quando estes eventos ocorrem, a ligação através da porta série é terminada.

**Stuffing**

**char\* stuffing(char\*** payload**, int\*** length**);**

Função responsável pelo mecanismo de *byte stuffing* da transparência de dados.

Caso os dados enviados contenham carateres 0x7E estes são substituídos por 0x7D 0x5E. Ou por 0x7D 0x5D, caso sejam carateres 0x7D.

Além disso, o cálculo de BCC2 é feito antes da operação de stuffing.

**Destuffing**

**char\* destuffing(char\*** msg**, int\*** length**);**

Função responsável pelo mecanismo de *byte destuffing* da transparência de dados.

Caso os dados recebidos contenham os carateres 0x7D 0x5E de forma consecutiva, estes são substituídos por 0x7E. Ou então, por 0x7D caso sejam recebidos os carateres 0x7D 0x5D.

Além disso, o cálculo de BCC2 é feito depois da operação de destuffing.

**Verify\_bcc2**

**char\* verify\_bcc2(char\*** control\_message**, int\* length);**

Função responsável pela comparação da variável de controlo BCC2 recebida com a enviada, que permite saber se os dados foram enviados sem erros, caso estas duas sejam iguais.

**Control\_frame**

**char\* control\_frame(char\*** filename**, FILE \***file**, int** start**, int\*** frame\_size**);**

Função responsável pela criação do pacote de controlo START/END.

**Get\_info**

**char\* get\_info(char\*** control**, int\*** file\_size**);**

Função responsável pela extração do nome e tamanho do ficheiro a ser transferido.

**Header**

**char\* header(char\*** buffer**, int\*** length**, short** sequence\_number**);**

Função responsável pela adição do campo de controlo, número de sequência, e tamanho da *payload* às tramas que vão ser enviados pelo emissor.

**Remove\_header**

**char\* remove\_header(char\*** buffer**, int\*** length**);**

Função responsável pela remoção do campo de controlo, número de sequência, e tamanho da *payload,* das tramas que vão ser recebidos pelo recetor.

*Protocolo de aplicação*

O protocolo de aplicação é implementado na camada de aplicação (“*Application Layer”)* que é a camada de mais alto nível, implementada neste projeto e é responsável pelas seguintes funcionalidades:

* Envio e receção de pacotes de controlo;
* Envio e receção de pacotes de dados;
* Envio e receção do ficheiro definido.

**Variáveis do Nível de Aplicação**

* **com\_type:** indica se a aplicação está a correr no modo de *sender* ou *receiver*.
* **Chunk\_syze:** indica o tamanho dos fragmentos de informação que vão ser enviados através da porta série de cada vez.

**Emissor**

O protocolo de aplicação implementado para o modo de emissor, consiste em:

* Abertura do ficheiro em modo de leitura de modo a ser possível usar a informação deste.
* Estabelecimento de comunicação através da função *LLOPEN().*
* Envio da trama de controlo inicial obtida através da função *control\_frame().*
* Envio de fragmentos do ficheiro com tamanho definido por Chunk\_syze, com o devido campo de controlo, número de sequência e tamanho obtidos na função *header()* e com transparência assegurada pela função *stuffing()*.
* Criação de uma barra de progresso que indica a percentagem de informação que está a ser enviada em relação à informação total.
* Envio da trama de controlo final obtida através da função *control\_frame().*
* Impressão no terminal do tempo de envio e do débito binário.

**Recetor**

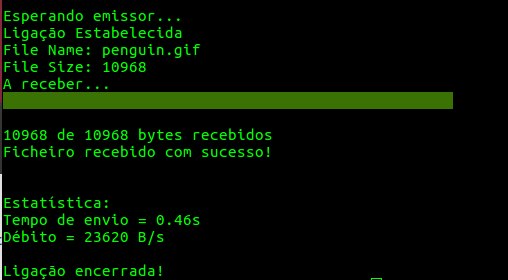
O protocolo de aplicação implementado para o modo de recetor, consiste em:

* Estabelecimento de comunicação através da função *LLOPEN()*.
* Leitura da informação do ficheiro através da função *LLREAD().*
* Verificação se a informação recebida é a mesma do que a enviada.
* Caso seja válida, envia RR e começa a escrita num ficheiro com o mesmo nome do ficheiro enviado.
* Caso não seja válida, envia RR e não realiza qualquer escrita no ficheiro.
* Ocorrência de *destuffing* e verificação do BCC2.
* Invocação da função *remove\_header(),* que remove o campo de controlo, número de sequência e tamanho da informação recebida.
* Criação de uma barra de progresso que indica a percentagem de informação que está a ser recebida em relação à informação total.
* Impressão no terminal o tempo de envio e o débito binário.

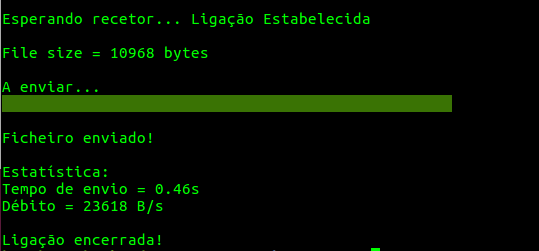
*Validação*

Nesta secção abordam-se os testes realizados relativamente à transferência de ficheiros e comentários acerca dos mesmos.

Relativamente à transferência do ficheiro penguin.gif exigida no guião, esta é realizada pela aplicação com distinto sucesso. De seguida estão apresentados prints da realização do referido teste.

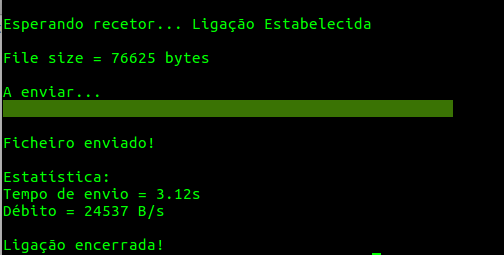


*Fig. 1 - Modo Recetor*

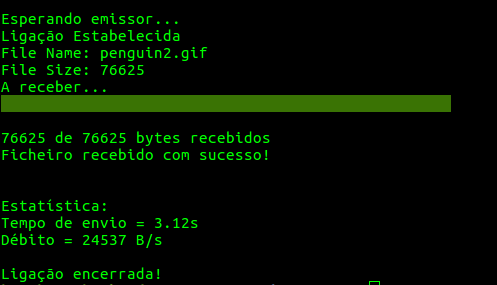
**

*Fig. 2 - Modo Emissor*

Além disso, também foi testado um ficheiro .gif de maior tamanho com sucesso idêntico ao anterior, como comprovado em seguida.



*Fig. 3 - Modo Emissor com um ficheiro maior*



*Fig. 4 - Modo Recetor com um ficheiro maior*

Por fim, foram realizados testes onde a ligação da porta série foi cortada, de modo a testar os *timeouts* e se a ligação continuava após o restabelecimento da conexão; tais testes foram verificados com sucesso. Também foi testada uma a ligação à massa de pinos do lado do emissor de modo a introduzir erros, o dito teste foi igualmente foi passado com sucesso.

Apesar de não serem apresentadas imagens, estes testes foram comprovados pelo docente na altura da apresentação do projeto.

*Elementos de valorização*

**Seleção do ficheiro a enviar**

O utilizador pode selecionar qualquer ficheiro que pretenda enviar bastando, para isso, possuir o mesmo na pasta do projeto e indicar o seu nome no terminal.

**Barra de progresso de envio/receção**

Apresentação de uma barra de progresso no envio/receção do ficheiro.

**Processamento das Tramas**

É feito o processamento das tramas, sendo que:

* Tramas são processadas através das funções *control\_frame()* e *header()*;
* Tramas com erros são verificadas através das variáveis de controlo REJ, RR, BCC e BCC2 e não são guardadas;

**Verificação da Integridade dos Dados**

A aplicação verifica se o tamanho do ficheiro recebido é igual ao tamanho do ficheiro enviado na appLayer.c

**if**(getFileSize(file) == received\_file\_size){

**printf**("ld de %d bytes recebidos\n",getFileSize(file),received\_file\_size);

**printf**("Ficheiro recebido com sucesso!\n\n");

}

**else**{

**printf**("%ld de %d bytes recebidos\n",getFileSize(file),received\_file\_size);

**printf**("Tamanho recebido diferente de tamanho original!\n\n");

}

*Conclusões*

Durante as últimas semanas quer através das aulas laboratoriais, quer através de tempo não letivo, o grupo desenvolveu esta aplicação que permite a transferência de dados entre dois computadores através de uma ligação física à porta série.

O trabalho foi bem entendido pelos elementos do grupo quer através da ajuda do docente da aula, que nos esclareceu diversos pontos acerca do mesmo, quer através do guião que nos serviu de guia.

Acerca da implementação do trabalho, os pontos pedidos no guião foram implementados, desde a divisão em camadas, o tratamento dos dados enviados através de tramas de controlo e o mecanismo de transparência de *byte stuffing.*

Em nota de conclusão, achamos que este trabalho contribuiu para a consolidação dos nossos conhecimentos acerca dos temas lecionados nas aulas teóricas e laboratoriais, permitindo-nos estar mais preparados para desafios futuros relacionados com este tema.

*Anexo I - Código fonte*

**appLayer.c**

1. /\*Non-Canonical Input Processing\*/
3. #define CHUNK\_SIZE 256                                              //Número de bytes do ficheiro a ser enviado de cada vez
4. #include "dataLink.h"
5. #include "utilities.h"
7. **int** main(**int** argc, **char**\*\* argv) {
8. fflush(NULL);
9. **int** fd = open(argv[1], O\_RDWR | O\_NOCTTY );
11. **struct** termios oldtio,newtio;

14. **if** (fd <0) {
15. perror(argv[1]);
16. exit(-1);
17. }
19. **if** ( tcgetattr(fd,&oldtio) == -1) { /\* save current port settings \*/
20. perror("tcgetattr");
21. exit(-1);
22. }
24. bzero(&newtio, **sizeof**(newtio));
25. newtio.c\_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
26. newtio.c\_iflag = IGNPAR;
27. newtio.c\_oflag = 0;
29. /\* set input mode (non-canonical, no echo,...) \*/
30. newtio.c\_lflag = 0;
32. newtio.c\_cc[VTIME]    = 1;   /\* inter-character timer unused \*/
33. newtio.c\_cc[VMIN]     = 0;   /\* blocking read until 5 chars received \*/


37. tcflush(fd, TCIOFLUSH);
39. **if** ( tcsetattr(fd,TCSANOW,&newtio) == -1) {
40. perror("tcsetattr");
41. exit(-1);
42. }

45. fflush(NULL);

48. /\*-------------------------------------------------------------
49. ---------------------------------------------------------------
50. ---------------------------------------------------------------
51. ----------------Início da Comunicação--------------------------
52. ---------------------------------------------------------------
53. ---------------------------------------------------------------
54. ---------------------------------------------------------------
55. \*/

58. //Tipo de comunicação: Sender (1) or Receiver (0)
59. **int** com\_type = check\_arguments(argc, argv);
60. **struct** timeval start, stop;
61. **double** secs = 0;
62. **FILE** \*file;
64. printf("\n");
66. **if**(com\_type) {

69. **if**((file=openfile(argv[3], com\_type))==NULL)exit(1);
70. **if**(LLOPEN(fd , com\_type)==-1) exit(1);                                      //Estabelecimento da comunicação
71. gettimeofday(&start, NULL);                                                 //Inicia relógio
72. fflush(NULL);


76. **char** buffer[CHUNK\_SIZE];
77. **char** \*payload;
78. **char** \*stuffed;
79. **int** size;
80. **short** sequence\_number = 0;
81. **char** \*control = control\_frame( argv[3] , file , 1 , &size);                 //START frame
83. stuffed = stuffing (control, &size);

86. **if**(LLWRITE(fd, stuffed, size) == -1 )                                       //Send file info
87. exit(1);
89. printf("A enviar...\n");
91. **int** enviado=0;
92. **int** file\_size = getFileSize(file);
94. **while** ( (size = fread(buffer, **sizeof**(**char**), CHUNK\_SIZE, file)) > 0){     //Lê fragmento do ficheiro
95. enviado = enviado + size;
96. loading(enviado,file\_size);                                         //Barra de progresso

99. payload = header(buffer, &size, sequence\_number++);                 //Adiciona campo de controlo, número de sequência, tamanho da payload
100. stuffed = stuffing(payload, &size);                                 //Transparência e adiciona BCC2
102. **if**(LLWRITE(fd, stuffed, size)==-1)                                  //Envia o trama
103. exit(1);
105. }
107. control = control\_frame( argv[3] , file , 0 , &size);                       //END frame
108. stuffed = stuffing (control, &size);
110. **if**(LLWRITE(fd, stuffed, size) == -1 )                                       //Send END frame
111. exit(1);
113. gettimeofday(&stop, NULL);                                                  //Para o relógio
114. printf("\n\nFicheiro enviado!\n");
115. secs = (**double**)(stop.tv\_usec - start.tv\_usec) / 1000000 + (**double**)(stop.tv\_sec - start.tv\_sec);
116. printf("\nEstatística:\n");
117. printf("Tempo de envio = %.2fs\n",secs);
118. printf("Débito = %.0f B/s\n\n", file\_size/secs);
120. }
122. **if**(!com\_type){
124. **if**(LLOPEN(fd , com\_type)==-1) exit(1);                                      //Estabelecimento da comunicação
125. gettimeofday(&start, NULL);                                                 //Inicia o relógio
127. **char** stuffed [CHUNK\_SIZE+(**int**)CHUNK\_SIZE/2];
128. **char** \*payload;
129. **char** \*buffer;
130. **int** length;
131. **int** received\_file\_size=0;
132. **char** \*filename;
134. //Leitura da informação do ficheiro
135. **while**(1){
136. length = LLREAD(fd, stuffed);
138. buffer = verify\_bcc2(stuffed, &length);
139. **if**(buffer == NULL)
140. send\_REJ(fd);
141. **else**{
142. filename = get\_info(buffer, &received\_file\_size);
143. **if**(filename!=NULL){
144. printf("File Name: %s\nFile Size: %d\n", filename, received\_file\_size);
145. send\_RR(fd);
146. file=openfile(filename, com\_type);
147. **break**;
148. }
149. **else**
150. send\_REJ(fd);
151. }
152. }
154. //Receção do ficheiro
155. printf("A receber...\n");
156. **while**( (length = LLREAD(fd, stuffed) )> 0){
158. loading((**int**)getFileSize(file),received\_file\_size);
159. **if**(stuffed[0]==END){
161. buffer = verify\_bcc2(stuffed, &length);
162. **if**(buffer == NULL)
163. send\_REJ(fd);
164. **else**{
165. send\_RR(fd);
166. **break**;
167. }
168. }
169. **else**{
170. payload = verify\_bcc2(stuffed, &length);                                //Faz destuff e verifica o BCC2
172. **if**(payload == NULL)
173. send\_REJ(fd);
174. **else**{
175. buffer = remove\_header(payload, &length);                           //Remove Header
176. send\_RR(fd);
177. fwrite(buffer,1,length,file);
178. }
180. }
182. }
184. gettimeofday(&stop, NULL);
186. **if**(getFileSize(file) == received\_file\_size)
187. printf("\n\n%ld de %d bytes recebidos\nFicheiro recebido com sucesso!\n\n", getFileSize(file),received\_file\_size);
188. **else**
189. printf("\n\n%ld de %d bytes recebidos\nTamanho recebido diferente de tamanho original!\n\n", getFileSize(file),received\_file\_size);
190. secs = (**double**)(stop.tv\_usec - start.tv\_usec) / 1000000 + (**double**)(stop.tv\_sec - start.tv\_sec);
191. printf("\nEstatística:\n");
192. printf("Tempo de envio = %.2fs\n",secs);
193. **int** file\_size = getFileSize(file);
194. printf("Débito = %.0f B/s\n\n", received\_file\_size/secs);
195. fclose(file);
197. }

200. LLCLOSE(fd, com\_type);






208. **if** ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
209. perror("tcsetattr");
210. exit(-1);
211. }



216. close(fd);
217. **return** 0;
218. }

**dataLink.c**

1. #include "dataLink.h"
3. **int** flag=0;
4. **int** TIMEOUT=0;
5. **int** Nr=0;


9. **void** time\_out() {
10. printf("Timeout number: %d\n", TIMEOUT+1);
11. TIMEOUT++;
12. flag=1;
13. }
15. **int** LLOPEN(**int** fd, **int** com\_type) {
17. **if**(com\_type)
18. (**void**) signal(SIGALRM, time\_out);
19. **char** address, address2;
20. **char** c;
21. **int** state = 0;
23. **if**(!com\_type) {
24. printf("Esperando emissor...\n");
25. }
26. **else**
27. printf("Esperando recetor...\n");
28. **while**(TIMEOUT<=3) {
30. **if**(com\_type) {
32. send\_SET(fd);
33. alarm(3);
34. flag=0;
35. }

38. **while**(state != 5 && flag==0 ) {
40. read(fd, &c, 1);

43. **switch** (state) {
45. **case** 0:                                 //FLAG
46. **if**(c == FLAG) {
47. state = 1;
48. }//else stay in same state
49. **break**;
50. **case** 1:                                 //A\_R/A\_T
51. **if**(com\_type) {
52. address=A\_R;
53. }
54. **else** {
55. address=A\_T;
56. }
57. **if**(c == address) {
58. state = 2;
59. } **else** **if**(c == FLAG) {
60. state = 1;
62. } **else**
63. state=0;
64. **break**;
66. **case** 2:                                 //UA/SET
67. **if**(com\_type) {
68. address2=UA;
69. }
70. **else** {
71. address2=SET;
72. }
73. **if**(c == address2) {
74. state = 3;
75. } **else** **if**(c == FLAG) {
76. state = 1;
77. } **else** {
78. state = 0;
79. }
80. **break**;
81. **case** 3:                                 //BCC
82. **if** (c == (address^address2)) {
83. state = 4;
84. } **else** {
85. state = 0;
86. }
87. **break**;
88. **case** 4:                                 //FLAG
89. **if** (c == FLAG) {
90. state = 5;
91. **if** (!com\_type) {
92. send\_UA(fd, com\_type);
93. }
94. printf("Ligação Estabelecida\n");
95. **return** 1;
96. } **else** {
97. state = 0;
98. }
99. **break**;
100. }
101. }
102. }
104. printf("TIMEOUT - Ligação Não Estabelecida\n");
105. **return** -1;
106. }
108. **int** LLWRITE(**int** fd, **char** \*buffer, **int** length) {
110. fflush(NULL);
111. TIMEOUT = 0;
112. **char** \*trama = malloc((length+5)\***sizeof**(**char**));
113. **char** controlo;
114. **char** c;
115. **int** i, written, state = 0;
117. **if**(Nr==0) {
118. Nr=1;
119. controlo = RR0;
120. }
121. **else** **if**(Nr==1) {
122. Nr=0;
123. controlo = RR1;
124. }

127. //Prepara bytes iniciais
128. trama[0] = FLAG;
129. trama[1] = A\_T;
130. trama[2] = controlo;
131. trama[3] = trama[1]^trama[2];
132. trama[length+4] = FLAG ;
134. **for**(i = 4 ; i < length + 4 ; i++) {
135. trama[i]=buffer[i-4];
136. }


140. //ESPERAR PELO ACK
141. (**void**) signal(SIGALRM, time\_out);
142. TIMEOUT = 0;
143. **while**(TIMEOUT<=3) {
145. written = write(fd, trama, length + 5);
146. written = written-5;

149. alarm(3);
150. flag=0;
151. state=0;
152. **while**(state != 5 && flag==0 ) {

155. read(fd, &c, 1);

158. **switch** (state) {
159. **case** 0:                                     //FLAG
160. **if**(c == FLAG) {
161. state = 1;
162. }//else stay in same state
163. **break**;
164. **case** 1:                                     //A\_T/A\_R
166. **if**(c == A\_T) {
167. state = 2;
168. } **else** **if**(c == FLAG) {
169. state = 1;
171. } **else**
172. state=0;//else go back to beggining
173. **break**;
175. **case** 2:                                     //RR/REJ
177. **if**(c == RR0 || c == RR1) {
179. state = 3;
180. }
181. **else** **if**( c == REJ0 || c == REJ1){
183. state = 0;
184. flag = 1;
186. } **else** **if**(c == FLAG) {                  //FLAG
187. state = 1;
188. } **else** {
189. state = 0;
190. }
191. **break**;
192. **case** 3:                                     //BCC
193. **if** (c == (A\_T^controlo)) {
194. state = 4;
195. } **else** {
196. state = 0;
197. }
198. **break**;
199. **case** 4:                                     //FLAG
200. **if** (c == FLAG) {
201. state = 5;
202. **return** written;
204. } **else** {
205. state = 0;
206. }
207. **break**;
208. }
209. }
210. }
211. free(trama);
212. printf("TIMEOUT - Escrita não Realizada\n");
213. **return** -1;
215. }
217. **int** LLREAD(**int** fd, **char** \*buffer) {
219. **int** length=0;
220. **int** random;
221. **int** state = 0;
222. **char** c;
223. **char** controlo;
225. **if**(Nr==0) {
227. controlo = RR0;

230. }
231. **else** **if**(Nr==1) {
233. controlo = RR1;
234. }

237. **while**(state != 5) {
239. read(fd, &c, 1);
241. **switch** (state) {
242. **case** 0:                                 //FLAG
243. **if**(c == FLAG) {
244. state = 1;
245. }
246. **break**;
247. **case** 1:                                 //A
249. **if**(c == A\_T) {
250. state = 2;
251. } **else** **if**(c == FLAG) {
252. state = 1;
254. } **else**
255. state=0;
256. **break**;
258. **case** 2:                                 //RR
260. **if**(c == controlo) {
261. state = 3;
262. } **else** **if**(c == FLAG) {
263. state = 1;
264. } **else** {
265. state = 0;
266. }
267. **break**;
268. **case** 3:                                 //BCC
270. **if**(BCC\_ERROR\_PROBABILITY != 0){     //Introdução manual de erro de leitura
271. random = rand();
272. random = (random % (100/BCC\_ERROR\_PROBABILITY)) + 1;
274. **if**(random == 1)
275. c = !c;
276. }
278. **if** (c == ((A\_T^controlo))) {
279. state = 4;
280. } **else** {
281. printf("Erro no BCC\n");
282. state = 0;//else go back to beggining
283. }
284. **break**;
285. **case** 4:                                 //FLAG
286. **if**( c == FLAG){
287. state = 5;
288. }
289. **else**{
290. buffer[length] = c;
291. length++;
292. }

295. **break**;
296. }
297. }
298. **if**(TRANSMISSION\_DELAY)
299. delay(TRANSMISSION\_DELAY\*2);
300. **return** length;


304. }
306. **int** LLCLOSE(**int** fd, **int** com\_type) {
308. **char** address;
309. **int** state = 0;
310. **char** c;
311. **int** receiveUA = 0;

314. **if**(com\_type)
315. send\_DISC(fd, com\_type);



320. **while**(TIMEOUT<=3 || !com\_type) {
322. **if**(com\_type) {
323. alarm(3);
324. flag=0;
325. }

328. **while**(state != 5 && flag==0 ) {
330. read(fd, &c, 1);
332. **switch** (state) {
333. **case** 0:                                 //FLAG
334. **if**(c == FLAG) {
335. state = 1;
336. }
337. **break**;
338. **case** 1:                                 //A
339. **if**(com\_type) {
340. address=A\_R;
341. }
342. **else** **if**(!com\_type){
343. address=A\_T;
344. }
345. **if**(c == address) {
346. state = 2;
347. }**else** **if**(c==FLAG){
348. state = 1;
350. } **else**
351. state=0;
352. **break**;
354. **case** 2:
356. **if**(c == DISC) {                     //DISC
357. state = 3;
358. } **else** **if**(c == FLAG) {
359. state = 1;
360. }
361. **else** **if**(c == UA && receiveUA == 1){
362. state = 3;
363. } **else** {
364. state = 0;
365. }
366. **break**;
367. **case** 3:                                 //BCC
369. **if** (c == (address^DISC)) {
370. state = 4;
371. }
372. **else** **if**((receiveUA == 1) && (c == (UA^address))){
374. state = 4;
375. }
376. **else** {
377. state = 0;
378. }
380. **break**;
381. **case** 4:                                 //FLAG
383. **if** (c == FLAG) {
384. state = 5;
385. **if** (!com\_type && receiveUA == 0) {
386. send\_DISC(fd, com\_type);
388. state = 0;
389. receiveUA = 1;
390. }
391. **else** **if**(com\_type){
392. send\_UA(fd, com\_type);
394. printf("Ligação encerrada!\n");
395. **return** 1;
396. }
397. **else** **if**(receiveUA == 1){
399. printf("Ligação encerrada!\n");
400. **return** 1;
401. }
403. } **else** {
404. **return** -1;
405. }
406. **break**;
407. }
408. }
409. }


413. **return** -1;
414. }
416. **void** send\_UA(**int** fd, **int** com\_type) {
418. **char** trama\_UA[5];
419. **char** address;
420. trama\_UA[0]=FLAG;
421. **if**(com\_type){
422. address = A\_T;
423. trama\_UA[1]=address;
424. }
425. **else**{
426. address = A\_R;
427. trama\_UA[1] = address;
428. }
429. trama\_UA[2]=UA;
430. trama\_UA[3]=address^UA;
431. trama\_UA[4]=FLAG;
433. write(fd, trama\_UA, 5);
434. fflush(NULL);
436. }
438. **void** send\_SET(**int** fd) {
440. **char** trama[5];
441. trama[0]=FLAG;
442. trama[1]=A\_T;
443. trama[2]=SET;
444. trama[3]=A\_T^SET;
445. trama[4]=FLAG;
447. write(fd, trama, 5);
448. fflush(NULL);
450. }
452. **void** send\_DISC(**int** fd, **int** com\_type) {
454. **char** trama[5];
455. **char** address;
456. trama[0]=FLAG;
457. **if**(com\_type){
458. address = A\_T;
459. trama[1] = address;
460. }
461. **else**{
462. address = A\_R;
463. trama[1] = address;
464. }
465. trama[2]=DISC;
466. trama[3]=address^DISC;
467. trama[4]=FLAG;
469. write(fd, trama, 5);
470. fflush(NULL);
472. }
474. **void** send\_RR(**int** fd){
476. **char** controlo;
477. **if**(Nr==0) {
479. Nr = 1;
480. controlo = RR0;
481. }
482. **else** **if**(Nr==1) {
484. Nr = 0;
485. controlo = RR1;
486. }
488. **char** trama[5];
489. trama[0]=FLAG;
490. trama[1]=A\_T;
491. trama[2]=controlo;
492. trama[3]=(A\_T^controlo);
493. trama[4]=FLAG;
495. write(fd, trama, 5);
496. fflush(NULL);

499. }
500. **void** send\_REJ(**int** fd){
501. **char** controlo;
502. **if**(Nr==0) {
504. controlo = REJ0;
505. }
506. **else** **if**(Nr==1) {
508. controlo = REJ1;
509. }
511. **char** trama[5];
512. trama[0]=FLAG;
513. trama[1]=A\_T;
514. trama[2]=controlo;
515. trama[3]=(A\_T^controlo);
516. trama[4]=FLAG;
518. write(fd, trama, 5);
519. fflush(NULL);
521. }
523. **char**\* verify\_bcc2(**char**\* control\_message, **int**\* length){
525. **char**\* destuffed\_message = destuffing(control\_message, length);
527. **int** i;
528. **int** random;
529. **char** control\_bcc2 = 0x00;
530. **for**(i=0; i<\*length-1; i++){
531. control\_bcc2 ^= destuffed\_message[i];
533. }
534. **if**(BCC2\_ERROR\_PROBABILITY != 0){
535. random = rand();
536. random = (random % (100/BCC2\_ERROR\_PROBABILITY)) + 1;
538. **if**(random == 1)
539. control\_bcc2 = !control\_bcc2;
540. }
541. **if**(control\_bcc2 != destuffed\_message[\*length-1]){
542. \*length = -1;
543. **return** NULL;
544. }
545. \*length = \*length-1;
546. **char**\* data\_message = (**char**\*) malloc(\*length);
547. **for**(i=0; i<\*length; i++){
548. data\_message[i] = destuffed\_message[i];
549. }
550. free(destuffed\_message);

553. **return** data\_message;
554. }
556. **char**\* destuffing(**char**\* msg, **int**\* length){

559. **char**\* str = (**char**\*) malloc(\*length);
560. **int** i;
561. **int** new\_length = 0;


565. **for**(i=0; i<\*length; i++){
566. new\_length++;
568. **if**(msg[i] == 0x7d){
569. **if**(msg[i+1] == 0x5e){
570. str[new\_length-1] = FLAG;
571. i++;
572. }
573. **else** **if**(msg[i+1] == 0x5d){
574. str[new\_length -1] = 0x7d;
575. i++;
576. }
577. }
578. **else**{
579. str[new\_length-1] = msg[i];
580. }
582. }
583. \*length = new\_length;
585. **return** str;
586. }
588. **char**\* stuffing(**char**\* payload, **int**\* length){
590. **char**\* str;
591. **char**\* msg\_aux;
592. **int** i;
593. **int** j;

596. str = (**char** \*) malloc(\*length);
597. msg\_aux = (**char** \*) malloc(\*length);
598. **char** BCC2=0x00;
600. **for**(i=0,j=0; i < \*length; i++, j++){
601. BCC2^=payload[i];
602. msg\_aux[i] = payload[i];
603. }

606. msg\_aux = (**char**\*) realloc(msg\_aux,(\*length)+1);
607. msg\_aux[\*length/\*index = length-1\*/]=BCC2;
609. **for**(i=0,j=0; i < \*length+1; i++, j++){

612. **if**(msg\_aux[i] ==  FLAG){
613. str[j] = 0x7d;
614. str[j+1] = 0x5e;
615. j++;
616. }
617. **else** **if**(msg\_aux[i] == 0x7d){
618. str[j] = 0x7d;
619. str[j+1]= 0x5d;
620. j++;
621. }
622. **else**{
623. str[j] = msg\_aux[i];
624. }
625. }



630. \*length=j;
632. **return** str;
633. }
635. **char**\* control\_frame(**char**\* filename, **FILE** \*file, **int** start, **int**\* frame\_size){

638. **int** file\_name\_size  = strlen(filename);
639. **int** file\_size = getFileSize(file);                          //Get file size
640. **if**(start)
641. printf("\nFile size = %d bytes\n\n",file\_size);
642. **int** i = 0;
643. **char** file\_size\_in\_string[30];
644. sprintf(file\_size\_in\_string, "%d", file\_size);
646. \*frame\_size = 5 + file\_name\_size + strlen(file\_size\_in\_string);
647. **char** \*control\_frame = malloc(\*frame\_size);
649. **if**(start)
650. control\_frame[i++] = START;
651. **else**
652. control\_frame[i++] = END;

655. control\_frame[i++] = 0x00;
656. control\_frame[i++] = (**char**)strlen(file\_size\_in\_string);

659. **for**(; i < strlen(file\_size\_in\_string)+3 ; i++){
661. control\_frame[i] = file\_size\_in\_string[i-3];
662. }

665. control\_frame[i++] = 0x01;
666. control\_frame[i++] = (**char**) file\_name\_size;
668. **int** j;
669. **for**( j=i ; i<file\_name\_size+j ; i++ ){
671. control\_frame[i] = filename[i-j];
672. }
674. **return** control\_frame;
675. }
677. **char**\* get\_info(**char**\* control, **int**\* file\_size){
679. **if**(control[0]!=0x01)**return** NULL;
680. **int** pos = 4+control[2];
681. **int** filename\_size = control[4+control[2]];
683. **char** \*buffer = malloc(100);
685. **char**\* size = malloc(control[2]);
686. **int** i;
688. **for**(i=0 ; i < filename\_size ; i++ ){
689. buffer[i] = control[pos+1+i];
690. }

693. **for**(i=0 ; i < control[2] ; i++ )
694. size[i] = control[i+3];

697. \*file\_size = atoi(size);
698. **return** buffer;
699. }
701. **char**\* header(**char**\* buffer, **int**\* length, **short** sequence\_number){
703. **char**\* str = malloc((\*length)+4);
705. str[0] = 0x00;
706. str[1] = (**char**)sequence\_number;
707. str[2] = (**char**)(\*length)/256;
708. str[3] = (**char**)(\*length)%256;
710. **int** i;
711. **for**(i = 0 ; i < \*length ; i++ ){
712. str[i+4] = buffer[i];
713. }
715. \*length = \*length + 4;
716. **return** str;
718. }
720. **char**\* remove\_header(**char**\* buffer, **int**\* length){
722. **char**\* str = malloc(2 \* (\*length));
723. **int** i;


727. **for**(i = 0 ; i < \*length - 4 ; i++)
728. str[i] = buffer[i+4];
730. \*length = \*length-4;
731. **return** str;

734. }

**dataLink.h**

1. #include <sys/types.h>
2. #include <sys/stat.h>
3. #include <fcntl.h>
4. #include <termios.h>
5. #include <stdio.h>
6. #include <signal.h>
7. #include <unistd.h>
8. #include <stdbool.h>
9. #include <stdlib.h>
10. #include <string.h>
11. #include <sys/time.h>
12. #include "utilities.h"
14. //--------------------------------------------------------------------------------------------------------------------------------------
15. #define BCC\_ERROR\_PROBABILITY   0       //Probabilidade de erro na leitura de BCC ----------Valores entre 0-50  ------------------------
16. #define BCC2\_ERROR\_PROBABILITY  0       //Probabilidade de erro na leitura de BCC2  --------Valores entre 0-50 -------------------------
17. #define TRANSMISSION\_DELAY      5       //Tempo simulado de propagação dos pacotes (ms) ------------------------------------------------
18. //--------------------------------------------------------------------------------------------------------------------------------------
20. #define TRANSMITTER 1
21. #define RECEIVER 0

24. #define BAUDRATE B38400
25. #define MODEMDEVICE "/dev/ttyS1"
26. #define \_POSIX\_SOURCE 1 /\* POSIX compliant source \*/
27. #define FALSE 0
28. #define TRUE 1
30. #define OPEN 0
31. #define READ 1
32. #define WRITE 2
34. #define FLAG 0x7E
35. #define C\_T 0x03
36. #define C\_R 0x07
37. #define A\_T 0x03
38. #define A\_R 0x01
39. #define BCC\_T 0x00
40. #define BCC\_R 0x04
42. #define UA 0x07
43. #define SET 0x03
44. #define DISC 0x0A
45. #define START 0x01
46. #define END 0x02
48. #define RR0 0x05
49. #define RR1 0x25
50. #define REJ0 0x01
51. #define REJ1 0x21

54. #define TRAMA\_S 0
55. #define TRAMA\_I 1
57. #define ERROR\_BCC1 1
58. #define ERROR\_BCC2 2
60. **void** time\_out();


64. //-------------------------------------------------Data Link Layer --------------------------
66. **int** LLWRITE(**int** fd, **char** \*buffer, **int** length);                                  //Envia o trama. Retorna nr de chars escritos. -1 em caso de erro
68. **int** LLOPEN(**int** fd, **int** com\_type);                                               //Estabelece a comunicação. Retorna 1 ou -1(erro)
70. **int** LLREAD(**int** fd, **char** \*buffer);                                               //Retorna número de chars de Data lidos
72. **int** LLCLOSE(**int** fd, **int** com\_type);                                              //Encerra a comunicação. Retorna 1 ou -1 (erro)
74. //Funções de stuffing, destuffing e a função que verifica o BCC2
75. **char**\* stuffing(**char**\* payload, **int**\* length);                                     //Faz stuffing e adiciona BCC2
76. **char**\* destuffing(**char**\* msg, **int**\* length);                                       //Faz destuffing e retira BCC2
77. **char**\* verify\_bcc2(**char**\* control\_message, **int**\* length);                          //Verifica BCC2
78. **char**\* control\_frame(**char**\* filename, **FILE** \*file, **int** start, **int**\* frame\_size);    //Cria pacote de controlo START/END
79. **char**\* get\_info(**char**\* control, **int**\* file\_size);                                  //Extrai nome de ficheiro e respetivo tamanho em bytes
80. **char**\* header(**char**\* buffer, **int**\* length, **short** sequence\_number);                 //Adiciona byte de controlo, número de sequência e tamanho do campo de dados
81. **char**\* remove\_header(**char**\* buffer, **int**\* length);                                 //Remove Header
83. **FILE** \*openfile(**char**\* filename, **int** com\_type);
84. **void** send\_SET(**int** fd);
85. **void** send\_UA(**int** fd, **int** com\_type);
86. **void** send\_DISC(**int** fd, **int** com\_type);
87. **void** send\_REJ(**int** fd);
88. **void** send\_RR(**int** fd);

*Anexo II – Estatísticas realizadas*

|  |  |
| --- | --- |
| Tamanho do ficheiro (bytes) | Baudrate |
| 10968 | 38400 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Chunk Size | tempo (s) | R (bytes/s) | R (bit/s) | S (R/C) |
| 20 | 22,16 | 495 | 3960 | 0,103 |
| 40 | 11,14 | 984,5 | 7876 | 0,205 |
| 60 | 7,45 | 1472,2 | 11777,6 | 0,307 |
| 80 | 5,64 | 1945,3 | 15562,4 | 0,405 |
| 100 | 4,51 | 2431,3 | 19450,4 | 0,507 |
| 120 | 3,79 | 2893,6 | 23148,8 | 0,603 |
| 140 | 3,27 | 3359,2 | 26873,6 | 0,700 |
| 160 | 2,86 | 3830,4 | 30643,2 | 0,798 |
| 180 | 2,55 | 4307,2 | 34457,6 | 0,897 |
| 200 | 2,31 | 4756,1 | 38048,8 | 0,991 |

*Fig. 5 - Tabelas da eficiência variando o tamanho das tramas.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Baudrate (C) | Tempo (s) | R (bytes/s) | R (bit/s) | S (R/C) |
| 600 |  |  | 0 | 0,000 |
| 1200 |  |  | 0 | 0,000 |
| 1800 |  |  | 0 | 0,000 |
| 2400 |  |  | 0 | 0,000 |
| 4800 |  |  | 0 | 0,000 |
| 9600 |  |  | 0 | 0,000 |
| 19200 |  |  |  |  |
| 38400 |  |  |  |  |

|  |  |
| --- | --- |
| Tamanho do ficheiro (bytes) | Chunk Size |
| 10968 | 100 |

*Fig. 6 – Tabelas da eficiência variando o Baudrate.*

|  |  |  |
| --- | --- | --- |
| Tamanho do ficheiro (bytes) | Baudrate | Chunk Size |
| 10968 | 38400 | 100 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| prob erro(bcc1+bcc2)% | tempo (s) | R (bytes/s) | R (bit/s) | S (R/C) |
| 0+0 | 4,52 | 2428,8 | 19430,4 | 0,506 |
| 2+0 | 7,61 | 1440,7 | 11525,6 | 0,300 |
| 0+2 | 4,55 | 2410,3 | 19282,4 | 0,502 |
| 2+2 | 4,63 | 2367,7 | 18941,6 | 0,493 |
| 4+2 | 10,84 | 1011,9 | 8095,2 | 0,211 |
| 2+4 | 4,75 | 2307,4 | 18459,2 | 0,481 |
| 4+4 | 10,92 | 1004,8 | 8038,4 | 0,209 |
| 6+4 | 10,92 | 1004,7 | 8037,6 | 0,209 |
| 4+6 | 11 | 997,5 | 7980 | 0,208 |
| 6+6 | 11 | 997,5 | 7980 | 0,208 |
| 8+6 | 29,5 | 371,8 | 2974,4 | 0,077 |
| 6+8 | 11,12 | 989,5 | 7916 | 0,206 |
| 8+8 | 29,74 | 368,8 | 2950,4 | 0,077 |
| 10+8 | 29,78 | 368,3 | 2946,4 | 0,077 |
| 8+10 | 29,46 | 372,3 | 2978,4 | 0,078 |
| 10+10 | 33 | 332,4 | 2659,2 | 0,069 |

*Fig. 7 – Tabelas da eficiência variando a % de erros*