Protocolo de Ligação de Dados

1 Trabalho Laboratorial

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Turma 1 - Grupo 4

Duarte Pinto Valente – up201504327
Nuno Miguel Outeiro Pereira – up201506265
Rui André Rebolo Fernandes Leixo – up201504818

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, s/n, 4200-465 Porto, Portugal

7 de Novembro de 2017

Conteúdo

1 Sumário

Este relatório tem como propósito suportar o primeiro projeto laboratorial da unidade curricular de Redes de Computadores, cujo foco foi a implementação de um protocolo de ligação de dados, sobre a porta série RS-232. Para deteção de erros nas tramas foi usado a estratégia ARQ Stop and Wait.

O trabalho é capaz de transferir o ficheiro sem erros, tanto quando são introduzidas interferências durante a transferência através de curtos-circuitos, tal como com a interrupção do funcionamento da porta de série, sendo que no primeiro caso, o pacote recebido é ignorado enquanto que no segundo o programa re-sincroniza a ligação entre o emissor e o recetor e retoma a transmissão.

2 Introdução

Este trabalho tem como principal objetivo a implementação de um protocolo de ligação de dados em C, em ambiente LINUX, onde nos foi requerida a transmissão de ficheiros , sendo esta era mediada por tramas de supervisão e tramas informação. A ligação, assíncrona, entre as duas máquinas foi efetuada através da porta de série RS-232, configurada em modo não canónico. Foram implementados os serviços llopen, llread, llwrite e llclose. O programa possui também um alarme caso o tempo de receção de uma trama exceda o máximo pré-estipulado, assim como uma condição de término do mesmo caso o número máximo de tentativas seja excedido para a mesma trama.

Este relatório vai ser subdividido em várias secções, de forma a que informação esteja mais facilmente acessível e seja mais percetível:

- Introdução: onde os principais objetivos do trabalho são apresentados
- Arquitetura: módulos e interfaces
- Estrutura do código: onde são apresentadas as funções de maior relevância, assim como estruturas de dados criadas de forma a facilitar a implementação de outras funcionalidades
- Casos de uso principais: identificação da principais sequências de chamada de funções
- Protocolo de ligação lógica, descrição da estratégia de implementação da camada de ligação de dados
- Protocolo de aplicação, descrição da estratégia de implementação da camada de aplicação
- Validação, onde serão alvo de atenção os testes realizados ao programa
- Eficiência do protocolo de ligação de dados, cálculos de eficiência através de estatísticas retiradas a partir de testes efetuados ao programa
- Conclusão considerações finais

3 Arquitetura

Estruturalmente, o presente trabalho encontra-se dividido por duas camadas, sendo elas a camada de ligação de dados, definida em *linkLayer* e a camada de aplicação, declarada em *appLayer*.
A camada de ligação de dados engloba todas as funções que lidam diretamente com a porta de série, incluindo a sua abertura e configuração, em *llopen*, e posterior encerramento, com *llclose*. É esta camada que regula o envio e receção de mensagens e comandos, bem como a sua interpretação, na função *receiveFrame*, de modo a manter o bom funcionamento do programa. As operações de *stuffing* e *destuffing* são também aqui implementadas.
A camada de aplicação é totalmente independente da camada de ligação, não tendo conhecimento dos procedimentos nela implementados. Esta secção é apenas responsável pela implementação de uma interface básica e pelo envio e receção do ficheiro. Para este efeito, recorre-se à criação de pacotes contendo partições de dados do ficheiro e de pacotes de controlo contendo o nome e tamanho do ficheiro a transmitir.
A aplicação admite dois modos de execução, recetor e emissor, pelo que o funcionamento de funções necessárias a ambos varia dependendo do modo escolhido no início do programa.

4 Estrutura do código

4.1 Camada de Ligação de Dados

Na camada da ligação de dados está definida uma estrutura de dados chamada LinkLayer com variadas informações desta camada, entre elas: prog (TRANSMITOR ou RECEIVER), o descritor de ficheiro da porta série, baudrate da ligação, número de sequência, o tamanho atual da trama, o número de octetos de informação efetiva lidos e a própria trama.

```
typedef struct {
char prog;
int fd; /* /dev/ttySx File descriptor*/
unsigned int baudrate;
unsigned int seqNum;
unsigned int frameSize;
unsigned int readBytes;
unsigned char* frame; /*Trama*/
} LinkLayer;
```

As principais funções implementadas nesta camada são: a função que estabelece a ligação, as função de leitura e escrita de um pacote, a função na qual foi implementado o término da conexão lógica bem como o *handler do SIGALRM*

```
int llopen(int port, char transmissor);
int llread(int fd, unsigned char * buffer);
int llwrite(int fd, unsigned char * buffer, unsigned int length);
int llclose(int fd);
void alarmHandler(int sigNum);
```

Outras funções também essenciais como as responsáveis pelo stuffing e destuffing, assim como pela receção da trama são também nesta camada implementadas.

```
int receiveFrame(LinkLayer* linkLayer);
int readData(LinkLayer* lk);
int bcc2Calc(unsigned char* buffer, int length);
int bcc2Check(LinkLayer* lk);
int destuffing(LinkLayer* lk);
int stuffing(unsigned char* buff, unsigned int* size);
```

4.2 Camada Aplicacional

No ficheiro da appLayer.h é possível encontrar a estrutura de dados que representa a camada aplicacional, AppLayer, sendo esta composta pelo descritor do ficheiro, descritor da porta de série, nome e tamanho do ficheiro a transmitir/receber, o pacote, assim como o tamanho do pacote num dado momento.

```
typedef struct{
  int fileFD;
  int serialPortFD;
  char* fileName;
  unsigned int fileSize;
  unsigned char* packet;
  unsigned int packetSize;
} AppLayer;
```

As funções implementadas nesta camada são as seguintes:

```
int sendControlPacket(AppLayer* appLayer, unsigned char control);
int receiveStartPacket(AppLayer* appLayer);
unsigned int receiveFile(AppLayer* appLayer);
unsigned int sendFile(AppLayer* appLayer);
void getFileSize(AppLayer* appLayer);
```

5 Casos de Uso Principais

A aplicação desenvolvida poderá ser executada em modo recetor ou em modo emissor. Independentemente do modo de execução, no início do programa, é solicitado a definição do baudrate e tamanho das tramas. Para executar o programa como recetor apenas necessário introduzir como parâmetros o número da porta de série e o modo de execução r. Caso se pretenda definir um nome para o ficheiro recebido, dever-se-à acrescentar o nome desejado após a definição do modo de execução, caso contrário, o ficheiro recebido ficará com o seu nome original. A função central, llread recebe a trama enviada pelo emissor, avalia-a, com recurso à função receiveFrame, e comunica ao emissor a resposta adequada, solicitando a próxima trama em caso de sucesso ou o reenvio caso tenham sido detetados erros na informação recebida. A execução como emissor necessita como parâmetros o número da porta de série, o modo de execução w e o nome do ficheiro a transmitir. Neste caso a principal função será llwrite, responsável por inserir os pacotes criados pela appLayer na trama, chamar bcc2Calc, para calcular o bcc2, efetuar o stuffing, enviar a trama e, por fim, receber a confirmação da receção da trama. No caso de terem ocurrido erros na transmissão, esta função é também responsável pelo reenvio da informação. As funções llopen e llclose são usadas, tanto no emissor como no recetor, embora com comportamento distinto, para efetuar a abertura e encerramento da porta de série.

6 Protocolo de Ligação Lógica

Esta é a camada responsável por estabelecer a ligação através da porta de série, efetuando-se a partir da função llopen. Esta função envia um comando SET e aguarda um comando UA como resposta do recetor, que serve para confirmar que recebeu um comando SET. Este comando UA tem de ser recebido pelo emissor dentro de um tempo limite estipulado pelo utilizador no início do programa, pois, caso contrário, um alarme será ativado. Após o aviso do alarme ter ocorrido, o comando SET será reenviado, repetindo todo o procedimento anterior. Este reenvio do comando SET tem também um número máximo de tentativas estipulado no programa e caso o número de tentativas esgote o programa irá terminar com erro. Caso o comando UA seja recebido com sucesso pelo emissor significa que a ligação foi estabelecida e que o programa deverá continuar a sua execução.

De seguida, a função llwrite é chamada para enviar a trama ao recetor e fica a aguardar uma resposta por parte deste para poder optar pela próxima ação. Se não for obtida uma resposta durante o período definido, o que se segue é semelhante ao caso anterior do estabelecimento da ligação sendo que neste caso a trama será reenviada até se ter atingido o numero máximo de tentativas, sendo que neste ponto, o programa termina com retorno de erro. Caso não se chegue a atingir este caso e de facto o emissor receba uma resposta, então uma ação será tomada conforme a resposta. Caso esta seja uma trama RR, então tudo correu conforme planeado e a trama foi recebida com sucesso, caso a resposta seja uma trama REJ, a trama deve ser reenviada.

Outra função importante para este processo é a llread que fica num ciclo a aguardar a receção da trama. Esta função vai chamar uma outra função denominada receiveFrame, onde se encontra a máquina de estados para analisar a trama que está a ser recebida, sendo que na eventualidade de serem encontrados bytes não esperados na trama, voltar ao início da máquina de estados e tentar encontrar o pedaço de informação certo. Existe também uma outra função chamada neste processo, readData, que vai ler o pedaço da trama que contém a informação do ficheiro, chamando também a função responsável por fazer o destuffing da informação. A seguir, seguem-se as verificações do BCC2 e se tudo corroborar a função envia como resposta um RR com o número da sequência seguinte e guarda a informação que recebeu. Caso este processo de verificação falhe, é enviado um REJ com o número da sequência da trama que acabou de ser rejeitada, para ser reenviada. Há ainda a possibilidade de a trama recebida conter o comando DISC, sinal de que a ligação deve ser fechada.

O comando DISC, referido aqui anteriormente, é gerado pela função llclose que tem como objetivo terminar a ligação, enviando para tal efeito, o comando DISC ao recetor, sendo que para a ligação ser terminada com sucesso, é também necessário que o recetor envie um UA e este seja recebido pelo emissor pois, de outra forma, tal como nas funções anteriores um alarme vai controlar o tempo decorrido e o número de tentativas para poder terminar com sucesso.

7 Protocolo de Aplicação

A camada de aplicação é a camada de mais alto nível do programa, sendo portanto responsável pelo envio e receção de pacotes de dados e de pacotes de controlo, servindo-se para isso dos serviços da camada inferior. A receção dos dados é efetuada na função receiveFile onde são lidos pacotes de dados até ser encontrado o pacote de controlo end sinalizando o fim da transferência.

```
unsigned int receiveFile(AppLayer* appLayer){
  unsigned int readBytes = 0;
  while(1){
          appLayer->packetSize =  llread(appLayer->serialPortFD, appLayer->packet);

    if(appLayer->packetSize != 0){
      if(appLayer->packet[0] == END_PACKET)
        break;

      readBytes += appLayer->packetSize-4;
      write(appLayer->fileFD, appLayer->packet + 4, appLayer->packetSize-4);
    }
  }
  return readBytes;
}
```

O envio e criação dos pacotes de dados são efetuados pela função sendFile que irá ler os dados do ficheiro que se procura transmitir e inseri-los no pacote que será seguidamente enviado. Após o envio do pacote de controlo a função retorna o número de bytes de dados transmitidos.

```
unsigned int sendFile(AppLayer* appLayer);
```

A função sendControlPacket é responsável pela criação e envio de pacotes de controlo. Cada pacote contém o tamanho e nome do ficheiro a transmitir.

```
int sendControlPacket(AppLayer* appLayer, unsigned char control);
```

A validação da chegada de um pacote start ao recetor faz-se na função receiveStartPacket que, para além de receber o pacote de controlo, também lê o nome e tamanho do ficheiro a ser recebido.

```
int receiveStartPacket(AppLayer* appLayer);
```

8 Validação

Para efeito de compreensão e verificação do correto funcionamento do programa desenvolvido foi efetuada a transmissão de um ficheiro sob diferentes condições e de seguida, testado com ficheiros diferentes. Inicialmente, foi transferido o ficheiro "pinguim.gif"de forma a perceber se a aplicação estava a funcionar corretamente.
O teste seguinte baseou-se em efetuar a mesma transferência mas causando uma interrupção no decorrer da mesma, através do botão presente na placa, restabelecendo de seguida a transferência.
Por fim, foi transferido sendo introduzidos erros na transmissão por meio de curto-circuitos causados pelo fio de cobre ligado ao pino Rx do Recetor.
O programa passou com sucesso em todos os testes, ultrapassando as dificuldades, tal pode ser comprovado quer pela abertura do ficheiro recebido assim como pelo número de bytes do mesmo, igual ao do original.

9 Eficiência do Protocolo da Ligação de Dados

No protocolo de ligação foi implementado um mecanismo de Automatic Repeat ReQuest para lidar com perdas de pacotes ou pacotes com erros nos dados. Assim, após ser enviada uma trama de informação o emissor espera por uma mensagem de confirmação do recetor, solicitando a próxima trama, ou um pedido de retransmissão caso tenham sido encontrados erros.
A eficiência do protocolo foi testada para diferentes valores de baudrate e tamanho de trama e as medições resultantes permitiram a elaboração dos seguintes gráficos.

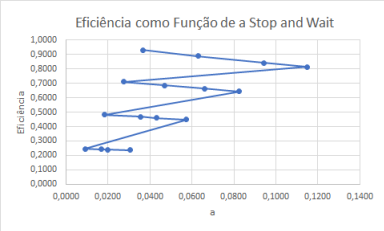

Figura 1: Eficiência do protocolo com frame 512 bytes e baud 19200

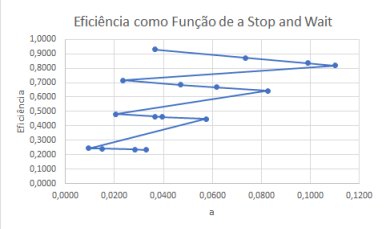

Figura 2: Eficiência do protocolo com frame 1024 bytes e baud 38400

Os resultados não foram os esperados, pois os valores de a não foram muito expressivos, logo não obtivemos um gráfico semelhante ao abordado nas aulas teóricas

10 Conclusões

No decorrer do desenvolvimento do projeto foram consolidados os conceitos de redes de computadores lecionados nas aulas teóricas, resultando numa aplicação composta por camadas independentes capaz de enviar e receber ficheiros através de uma porta de série, com baudrate e tamanho de pacote variável, tendo em conta erros nos dados transmitidos, informação duplicada e eventuais interrupções da porta de série. Além de satisfazer todos os critérios especificados no guião, a aplicação informa o utilizador do tamanho em bytes do ficheiro após a sua transmissão e receção, facilitando a verificação do sucesso da operação. Adicionalmente, após a receção, o utilizador é informado sobre o tempo da transmissão do ficheiro, funcionalidade que se destacou na criação de estatísticas de modo a avaliar a performance do programa.

11 Anexos

11.1 Camada de Ligação de Dados

Ficheiros Header

```
#ifndef LINK_LAYER_H
#define LINK_LAYER_H

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

//#define FRAME_SIZE 64000
unsigned int FRAME_SIZE;

typedef struct {
char prog;
int fd; /* /dev/ttySx File descriptor*/
unsigned int baudrate;
unsigned int seqNum;
unsigned int frameSize;
unsigned int readBytes;
unsigned char* frame; /*Trama*/
} LinkLayer;

LinkLayer linkLayer;//Global variable

#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1

#define TRANSMISSOR 1
#define RECEIVER 0

#define TIMEOUT 3
#define N_TRIES 3

//Statistics
#define T_PROP 10 //Signal propagation  time in miliseconds
#define FER 0 //Frame Error Ratio [0 , 1]

int receiveFrame(LinkLayer* linkLayer);
int readData(LinkLayer* lk);
int bcc2Calc(unsigned char* buffer, int length);
int bcc2Check(LinkLayer* lk);
int destuffing(LinkLayer* lk);
```

```
int stuffing(unsigned char* buff, unsigned int* size);

int llopen(int port, char transmissor);
int llread(int fd, unsigned char * buffer);
int llwrite(int fd, unsigned char * buffer, unsigned int length);
int llclose(int fd);
void alarmHandler(int sigNum);
int bcc2Error(float errorRatio);

#define C_IDX 2

//Info Frame
#define INFO 30

#define SEQ_NUM(NUM) (NUM << 6)
#define SEQ_NUM0 0
#define SEQ_NUM1 0x40

//SerialPort Control messages - Supervision
#define FLAG 0x7E
#define ADDRESS 0x03
#define ADDRESS1 0x01
//Control Field - C
#define SET 0x03
#define DISC  0x0B
#define UA  0x07

/*Receive fields MSbit R = N(r)*/
  // RR_0 0x05
  // RR_1 0x85
  #define RR(Num) (0x05 | Num << 7)

  //REJ_0 0x01
  //REJ_1 0x81
  #define REJ(Num) (0x1 | Num << 7)

//Stuffing
#define ESC 0x7d
#define ESC_EX 0x5d
#define FLAG_EX 0x5e

#endif
```

```
#ifndef STATE_MACHINES_H
#define STATE_MACHINES_H
//TRANSMISSOR and Receiver llopen() state machines

#define START 0
#define FLAG_RCV 1
#define A_RCV 2
#define C_RCV 3
#define BCC1_OK 4
```

```
10  #define END 7
11
12
13  #define RECEIVE 8
14
15
16  #endif
```

Ficheiro Código-Fonte

```c
#include "linkLayer.h"
#include "stateMachines.h"

unsigned int retryCount, state, stateRcv;
static struct termios oldtio;

#ifdef STATISCS
  srand(time(NULL));
#endif
/**
* @ SigAlm handler, it increments nTries,
* changes state to SET_SEND and stateRcv to END
*/

void alarmHandler(int sigNum){
  retryCount++;
  printf("Alarm triggered, retryCount =  %d\n", retryCount);
  stateRcv = END;
  state = START;
}


/**
* @ Establishes a serial connection between two machines
* @ parm port
* @ parm flag - boolean flag, 0 for emmisor and 1 stands for receiver
* @ return return the serial port's fd or a negative number if an error occurs
*/
int llopen(int port, char flag){
  unsigned char message[5];

  if(flag != TRANSMISSOR && flag != RECEIVER){
    perror("llopen()::Couldn't open serialPort fd\n");
    return -1;
  }

  char path[] = "/dev/ttyS", portString[2];

  portString[0] =  port + '0';
  portString[1] = '\0';

  strcat(path, portString);

  if((linkLayer.fd = open(path, O_RDWR | O_NOCTTY )) < 0){
    printf("llopen()::Couldn't open serialPort %d\n", port);
    return -1;
  }


  struct termios newtio;
```

```
51
52    if ( tcgetattr(linkLayer.fd, &oldtio) == -1) { /* save current port settings */
53      perror("tcgetattr");
54      exit(-1);
55    }
56
57    bzero(&newtio, sizeof(newtio));
58    newtio.c_cflag = linkLayer.baudrate | CS8 | CLOCAL | CREAD;
59    newtio.c_iflag = IGNPAR;
60    newtio.c_oflag = 0;
61
62    /* set input mode (non-canonical, no echo,...) */
63    newtio.c_lflag = 0;
64
65    newtio.c_cc[VTIME]    = 0;
66    newtio.c_cc[VMIN]     = 1;
67
68    tcflush(linkLayer.fd, TCIOFLUSH);
69
70    if ( tcsetattr(linkLayer.fd, TCSANOW, &newtio) == -1) {
71      perror("tcsetattr");
72      exit(-1);
73    }
74
75  linkLayer.frame = malloc(2*FRAME_SIZE);
76
77    if(flag == TRANSMISSOR){
78      int connected = 0;
79      retryCount = 0;
80
81      state = START;
82
83      while(!connected &&  retryCount != N_TRIES){
84        switch (state) {
85          case  START:
86            message[0] = FLAG;
87            message[1] = ADDRESS;
88            message[2] = SET;
89            message[3] = SET ^ ADDRESS;
90            message[4] = FLAG;
91
92            write(linkLayer.fd, message, 5); //Send set message
93             if(alarm(TIMEOUT) != 0){
94               printf("Alarm already scheduled in seconds\n");
95             }
96            state = RECEIVE;
97            break;
98          case  RECEIVE://UA State Machine
99            receiveFrame(&linkLayer);
100           if(linkLayer.frame[C_IDX] == UA)
101             state = END;
102           break;
```

```
103           case END:
104             printf("Right Before Disabling Alarm\n");
105             alarm(0); //Disables the alarm
106             connected = TRUE;
107             printf("Received an UA\n");
108           break;
109           }
110         }
111       if(retryCount == N_TRIES){
112         printf("llopen failed due to the number o retries reaching it's limit\n");
113         return -1;
114       }
115
116     }
117
118     else{
119         unsigned int conEstab = 0;
120         state = START;
121         while(!conEstab){
122           switch (state) { //Check if SET Message is received
123             case START:
124               receiveFrame(&linkLayer);
125               if(linkLayer.frame[C_IDX] == SET)
126                 state = END;
127             break;
128
129             case END:
130               conEstab = TRUE;
131             break;
132           }
133         }
134
135         //After a successful SET message was received, send a UA
136         printf("Starting to send UA\n");
137         message[0] = FLAG;
138         message[1] = ADDRESS;
139         message[2] = UA;
140         message[3] = UA ^ ADDRESS;
141         message[4] = FLAG;
142         if(write(linkLayer.fd, message, 5) == 0)
143           printf("Failed to transmit an UA\n");
144     }
145
146           printf("Connection established\n");
147     return linkLayer.fd;
148   }
149
150   int llwrite(int fd, unsigned char* buffer, unsigned int length){
151     retryCount = 0;
152     unsigned int lenghtStuffng = length +1;
153     unsigned char bcc2 = bcc2Calc(buffer, length);
154
```

```
155     linkLayer.frame[0] = FLAG;
156     linkLayer.frame[1] = ADDRESS;
157     linkLayer.frame[2] = SEQ_NUM(linkLayer.seqNum);
158     linkLayer.frame[3] = SEQ_NUM(linkLayer.seqNum) ^ ADDRESS;//BCC1
159     memmove(linkLayer.frame+4, buffer, length);
160     linkLayer.frame[4+length] =  bcc2;
161
162     stuffing(linkLayer.frame + 4, &(lenghtStuffng));
163     linkLayer.frameSize = lenghtStuffng + 5;
164     linkLayer.frame[linkLayer.frameSize-1] = FLAG;
165
166
167     unsigned char frameCpy[linkLayer.frameSize];
168     unsigned int frameISize = linkLayer.frameSize;
169     memmove(frameCpy, linkLayer.frame, linkLayer.frameSize);
170
171     state=START;
172
173     unsigned char sent = 0;
174     unsigned int bytesWritten = 0;
175     while(!sent && retryCount < N_TRIES){
176       switch (state) {
177         case START:
178           linkLayer.frameSize = frameISize;
179           memmove(linkLayer.frame, frameCpy, linkLayer.frameSize);
180           #ifdef STATISCS
181             usleep(T_PROP*1000);
182           #endif
183           bytesWritten = write(fd, linkLayer.frame, linkLayer.frameSize);
184           alarm(TIMEOUT);
185           state = RECEIVE;
186         break;
187         case RECEIVE:
188           if(receiveFrame(&linkLayer)){
189             printf("llwrite: expected control frame but received Info instead\n");
190             printf("CONTROL FIELD %x\n", linkLayer.frame[C_IDX]);
191           }
192           else if(linkLayer.frame[C_IDX] == RR(0) || linkLayer.frame[C_IDX] == RR(1)){
193             state = END;
194             printf("llwrite RR\n");
195           }
196           else if(linkLayer.frame[C_IDX] == REJ(0) || linkLayer.frame[C_IDX] == REJ(1)){
197             printf("llwrite REJ\n");
198             state = START;
199           }
200         break;
201
202         case END:
203           alarm(0);
204           linkLayer.seqNum = (linkLayer.seqNum + 1) % 2;
205           sent = 1;
206         break;
```

```
207       }
208     }
209     if(retryCount == N_TRIES){
210       printf("llwrite::Exceeded number of tries\n");
211       exit(1);
212     }
213 
214     return bytesWritten - (lenghtStuffng-length) - 5; //FRAME_HEADER
215   }
216 
217   int llread(int fd, unsigned char * buffer){
218     unsigned char message[] = {FLAG, ADDRESS, 0, 0, FLAG};
219     int i;
220     for(i = 0; i < linkLayer.frameSize; i++)
221       printf("%x\n", linkLayer.frame[i]);
222 
223     unsigned char response = receiveFrame(&linkLayer);
224     printf("llread will send %x\n", response);
225     message[2] = response;
226     message[3] = response ^ ADDRESS;
227     #ifdef STATISCS
228       usleep(T_PROP*1000);
229     #endif
230     write(fd, message, 5);
231 
232     memcpy(buffer, linkLayer.frame, linkLayer.readBytes);
233     return linkLayer.readBytes;
234   }
235 
236   int possibleControlField(unsigned char controlField){
237       if(controlField == SET || controlField ==  UA || controlField ==  DISC ||
238       controlField ==  RR(0) || controlField ==  RR(1) ||
239       controlField ==  REJ(0) ||
240       controlField ==  REJ(1))
241         return 1;
242       return 0;
243   }
244 
245   /**
246   * Generic frame receiver, it can handle Info Frames as well as Supervision Frame_Size
247   * @param framelkLayer
248   * @return 0 if SupervisionFrame was received without errors and  RR(Num) or REJ(Num)  for data
      ↪  frame
249   */
250   int receiveFrame(LinkLayer* lkLayer){ //ADDRESS 0x03 or 0x01
251           int stop = 0;
252     unsigned int newSeqNum;
253 
254           int i = 0;
255           stateRcv = START;
256 
257           while(!stop){
```

```
258                 switch (stateRcv) {
259                         case START:
260                 memset(lkLayer->frame, 0, lkLayer->frameSize);
261                             read(lkLayer->fd, lkLayer->frame + i, 1);
262                             if(lkLayer->frame[i] == FLAG){
263                                     stateRcv = FLAG_RCV;
264                                     i++;
265                             }
266                             break;
267                         case FLAG_RCV:
268                 read(lkLayer->fd, lkLayer->frame + i, 1);
269                                 if(lkLayer->frame[i] == ADDRESS){
270                                         stateRcv = A_RCV;
271                                         i++;
272                                 }
273                                 else if(lkLayer->frame[i] != FLAG){//Other unexpected info
274                                         stateRcv = START;
275                                         i = 0;
276                                 }
277                                 break;
278
279                         case A_RCV:
280                                 read(lkLayer->fd, lkLayer->frame + i, 1);
281
282                 if(possibleControlField(lkLayer->frame[i])){
283                                                 stateRcv = C_RCV;
284                                                 i++;
285                                         }
286
287                                 else if(lkLayer->frame[i] == SEQ_NUM0 || lkLayer->frame[i] ==
                                    ↪  SEQ_NUM1){
288                     lkLayer->readBytes = 0;
289                     newSeqNum = lkLayer->frame[i] >> 6;
290                     i++;
291                     read(lkLayer->fd, lkLayer->frame + i, 1);
292                     if((lkLayer->frame[i-2] ^ lkLayer->frame[i-1]) == lkLayer->frame[i]){//BCC1
                        ↪  Check <=> A ^ C (seqNum) = BCC1
293                       if(lkLayer->seqNum == newSeqNum){
294                           printf("Duplicated frame, %d\n", newSeqNum);
295                           return RR((newSeqNum+1)%2);
296                       }
297
298
299                       if(readData(lkLayer) == 0){
300                         lkLayer->seqNum = (lkLayer->seqNum+1)%2;
301                         return RR(newSeqNum);
302                       }
303                       else
304                         return REJ(lkLayer->seqNum); //Requesting  REJ_0
305                     }
306                     i--;
307                     break;
```

```
308                 }
309             break;
310
311                         case C_RCV:
312                                 read(lkLayer->fd, lkLayer->frame + i, 1);
313                                 if(lkLayer->frame[i] == (ADDRESS ^ lkLayer->frame[i-1])){
314                                         stateRcv = BCC1_OK;
315                                         i++;
316                                 }
317                                 break;
318                         case BCC1_OK:
319                                 read(lkLayer->fd, lkLayer->frame + i, 1);
320                                 if(lkLayer->frame[i] == FLAG){
321                                         stateRcv = END;
322                                 }
323                                 break;
324                         case END:
325                                 stop = 1;
326                                 break;
327                 }
328         }
329   return 0;
330 }
331
332 /**
333 * Reads the data in a Information frame
334 * @param lk - LinkLayer's info struct
335 * @return 0 if there's no error, 1 for destuffing error and 2 for Bcc2 check error
336 */
337 int readData(LinkLayer* lk){
338   char stop = 0;
339   unsigned int i = 0;
340   lk->readBytes = 0;
341   while(!stop){
342     if(read(lk->fd, &lk->frame[i], 1) == 1){
343       lk->readBytes ++;
344       i++;
345     }
346
347     if(lk->frame[i-1] == FLAG){
348       stop = 1;
349       lk->readBytes--; //Subtracted non data - wrong increment
350     }
351   }
352
353     lk->frameSize = lk->readBytes; //FrameSize -> data + bbc2
354
355   if(destuffing(lk) != 0){
356     printf("Destuffing Error\n");
357     lk->readBytes = 0;
358     return 1;
359   }
```

```
360
361   if(bcc2Check(lk) || bcc2Error(FER)){
362     printf("Failed BCC2 check\n");
363     lk->readBytes = 0;
364     return 2;
365   }
366
367   return 0;
368 }
369
370 int bcc2Error(float errorRatio){
371   if((rand() % 100 + 1) <= errorRatio*100)
372     return 1;
373   return 0;
374 }
375
376 int bcc2Calc(unsigned char* buffer, int length){
377   int i;
378   unsigned char xorResult = buffer[0]; //D0
379
380   for (i = 1; i < length; i++) {
381     xorResult ^= buffer[i];
382   }
383
384   return xorResult;
385 }
386
387 int bcc2Check(LinkLayer* lk){
388   int i;
389   unsigned char xorResult = lk->frame[0]; //D0
390
391   for (i = 1; i < lk->frameSize-1; i++) {
392     xorResult ^= lk->frame[i];
393   }
394                   /*BCC2*/
395   if(xorResult != lk->frame[lk->frameSize-1]){
396     printf("->BCC2Check result: %x, instead in the frame was %x\n", xorResult,
      ↪  lk->frame[lk->frameSize-1]);
397     return 1;
398   }
399
400   lk->frameSize--;//BCC2 Ignored (Deleted)
401   lk->readBytes--;
402   return 0; //BCC2 field is correct
403 }
404
405
406 int stuffing(unsigned char* buff, unsigned int* size){
407
408   unsigned int i;
409
410   for(i=0; i< *size;){
```

```
411       if(buff[i] == FLAG){
412         memmove(&buff[i+2], &buff[i+1], (*size) - (i+1));
413         buff[i] = ESC;
414         buff[i+1] = FLAG_EX;
415         (*size) += 1;
416         i+=2;
417       }
418
419       else if(buff[i] == ESC){
420         memmove(&buff[i+2], &buff[i+1], (*size) - (i+1));
421         buff[i+1] = ESC_EX;
422         (*size) += 1;
423         i+=2;
424       }
425       else
426         i++;
427     }
428     return 0;
429   }
430
431   int destuffing(LinkLayer* lk){
432
433     unsigned int i, deletedBytes = 0;
434
435     for(i=0; i< lk->frameSize; i++){
436       if(lk->frame[i] == ESC){
437         if(lk->frame[i+1]==FLAG_EX){
438           lk->frame[i+1]= FLAG;
439         }
440         else if(lk->frame[i+1]==ESC_EX){
441           lk->frame[i+1]= ESC;
442         }
443         else{
444           printf("Unstuffing error\n");
445           return -1;
446         }
447         memmove(&lk->frame[i], &lk->frame[i+1], lk->frameSize - (i+1));
448         lk->frameSize--;
449         deletedBytes++;
450       }
451     }
452     lk->readBytes -= deletedBytes;
453     return 0;
454   }
455
456   int llclose(int fd){
457     printf("Entered llclose \n");
458     unsigned char discMsg[] = {FLAG, ADDRESS, DISC, DISC ^ ADDRESS, FLAG};
459     retryCount = 0;
460     if(linkLayer.prog == RECEIVER){
461       while(retryCount < N_TRIES){
462         receiveFrame(&linkLayer);
```

```
463         if(linkLayer.frame[C_IDX] == DISC){
464           printf("DISC Received\n");
465           write(fd, discMsg, 5);
466           alarm(TIMEOUT);
467           receiveFrame(&linkLayer);
468           if(linkLayer.frame[C_IDX] == UA){
469               printf("UA Received\n");
470               alarm(0);
471               break;
472           }
473         }
474       }
475     }
476     else{
477       while(retryCount < N_TRIES){
478         write(fd, discMsg, 5);
479         alarm(TIMEOUT);
480         receiveFrame(&linkLayer);
481         if(linkLayer.frame[C_IDX] == DISC){
482           printf("DISC Received\n");
483           discMsg[2] = UA;
484           discMsg[3] = UA ^ ADDRESS;
485           write(fd, discMsg, 5);
486           alarm(0);
487           break;
488         }
489       }
490     }
491     if(retryCount == N_TRIES){
492       printf("llclose::Exceeded number of tries\n");
493       exit(1);
494     }
495 
496     sleep(1);
497     free(linkLayer.frame);
498     if ( tcsetattr(fd, TCSANOW, &oldtio) == -1) {
499       perror("tcsetattr");
500       exit(-1);
501     }
502     printf("Exiting llclose...\n");
503     close(fd);
504     return 0;
505   }
```

11.2 Camada Aplicacional

Ficheiro Header

```
#ifndef APP_LAYER_H
#define APP_LAYER_H

#include <time.h>

//PACKET_SIZE = FRAME_SIZE - 6
// #define PACKET_SIZE FRAME_SIZE-6
unsigned int PACKET_SIZE;
typedef struct{
  int fileFD;
  int serialPortFD;
  char* fileName;
  unsigned int fileSize;
  unsigned char* packet;
  unsigned int packetSize;
} AppLayer;

int sendControlPacket(AppLayer* appLayer, unsigned char control);
int receiveStartPacket(AppLayer* appLayer);
unsigned int receiveFile(AppLayer* appLayer);
unsigned int sendFile(AppLayer* appLayer);

void getFileSize(AppLayer* appLayer);
double getElapsedTimeSecs(struct timespec* start, struct timespec* end);

//PACKET CONTROL
#define DATA_PACKET 1
#define START_PACKET 2
#define END_PACKET 3

//PACKT Type
#define T_SIZE 0
#define T_NAME 1
#endif
```

Ficheiro Código-Fonte

```
#include "appLayer.h"
#include "linkLayer.h"
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>

#define STATISCS


int main(int argc, char** argv)
{

  if ( (argc < 3) ||
        ((strcmp("0", argv[1])!= 0) &&
        (strcmp("1", argv[1])!= 0)) ||
        ((strcmp("r", argv[2])!= 0) &&
        (strcmp("w", argv[2])!= 0))){
    printf("Usage:\n\tserialCom <numPort> r [fileName]\n\tserialCom <numPort> w <fileName>\n");
    exit(1);
  }

  if((strcmp("w", argv[2]) == 0) && argc < 4){
    printf("Usage ex. serialCom 0 w filePath\n");
    exit(2);
  }

const unsigned int baudArray[] = {B1200, B2400, B4800, B19200, B38400, B115200};

unsigned int choice;

do{
  printf("Please choose the desired baudrate\n"
  "1-B1200 | 2-B2400 | 3-B4800 | 4-B19200 | 5-B38400 | 6-B115200\n"
  "Choice: ");
  scanf("%d", &choice);
}while(choice < 1 || choice > 6);
linkLayer.baudrate = baudArray[choice-1];

do{
  printf("Please insert the frame size (bytes) [512 - 64000]\n"
        "Choice: ");
  scanf("%d", &choice);
}while(choice < 512 || choice > 64000);
FRAME_SIZE = choice;

  struct sigaction sigact;
  sigact.sa_handler = alarmHandler;
  sigact.sa_flags = 0;

  if(sigaction(SIGALRM, &sigact, NULL) != 0){
```

```
51      perror("Failed to install the signal handler\n");
52      return -1;
53    }
54
55    AppLayer appLayer;
56          if(strcmp(argv[2], "r") == 0){
57
58      linkLayer.prog = RECEIVER;
59
60      if((appLayer.serialPortFD = llopen(atoi(argv[1]) , RECEIVER)) < 0){
61                                  printf("Receiver failed to establish the connection\n");
62          return 1;
63      }
64
65      PACKET_SIZE = FRAME_SIZE-6;
66      appLayer.packet = malloc(PACKET_SIZE);
67
68      linkLayer.seqNum = 1;
69
70      if(receiveStartPacket(&appLayer) != 0){
71        printf("Didn't receive the start packet\n");
72      }
73
74      if(argc == 4)
75        appLayer.fileName = argv[3];
76
77      if((appLayer.fileFD = open(appLayer.fileName, O_CREAT | O_WRONLY, 0666)) < 0){
78        printf("Couldn't create file named %s\n", appLayer.fileName);
79        return 1;
80      }
81
82      printf("Received %u Bytes\n", receiveFile(&appLayer));
83
84      close(appLayer.fileFD);
85      llclose(appLayer.serialPortFD);
86
87          }
88          else{
89
90      linkLayer.prog = TRANSMISSOR;
91                  if((appLayer.serialPortFD = llopen(atoi(argv[1]), TRANSMISSOR)) < 0){
92                          printf("Transmissor failed to establish the connection fd %d\n",
                             ↪  appLayer.serialPortFD);
93        return 1;
94      }
95
96      linkLayer.seqNum = 0;
97
98      appLayer.fileName = argv[3];
99      PACKET_SIZE = FRAME_SIZE-6;
100     appLayer.packet = malloc(PACKET_SIZE);
101
```

```
102       if((appLayer.fileFD = open(appLayer.fileName, O_RDONLY)) < 0){
103         printf("Couldn't open file named %s\n", appLayer.fileName);
104         return 1;
105       }
106
107       getFileSize(&appLayer);
108
109       if(sendControlPacket(&appLayer, START_PACKET) != 0){
110         printf("Couldn't send start packet\n");
111       }
112
113       printf("Sent %d bytes from file\n", sendFile(&appLayer));
114
115       free(appLayer.packet);
116       llclose(appLayer.serialPortFD);
117       close(appLayer.fileFD);
118     }
119     return 0;
120   }
121
122   unsigned int receiveFile(AppLayer* appLayer){
123     unsigned int readBytes = 0;
124   struct timespec start, end;
125   clock_gettime(CLOCK_REALTIME, &start);
126     while(1){
127         appLayer->packetSize =  llread(appLayer->serialPortFD, appLayer->packet);
128
129       if(appLayer->packetSize != 0){
130         if(appLayer->packet[0] == END_PACKET)
131           break;
132
133         readBytes += appLayer->packetSize-4;
134         write(appLayer->fileFD, appLayer->packet + 4, appLayer->packetSize-4);
135       }
136   }
137   clock_gettime(CLOCK_REALTIME, &end);
138   printf("Time elapsed: %f s\n", getElapsedTimeSecs(&start, &end));
139   #ifdef STATISCS
140     printf("FileSize: %d || %d\n", appLayer->fileSize, appLayer->fileSize / PACKET_SIZE);
141     printf("Tf = %f s\n", getElapsedTimeSecs(&start, &end)/(appLayer->fileSize / PACKET_SIZE));
142   #endif
143     return readBytes;
144   }
145
146   unsigned int sendFile(AppLayer* appLayer){
147     unsigned int writtenBytes = 0;
148     int readFromFile  = 1;
149     int llwriteReturn;
150
151     appLayer->packet[0] = DATA_PACKET;
152     appLayer->packet[1] = 0;
153     appLayer->packet[2] = (PACKET_SIZE - 4) / 256;
```

```
154     appLayer->packet[3] = (PACKET_SIZE - 4) % 256;
155
156
157     readFromFile = read(appLayer->fileFD, appLayer->packet + 4, PACKET_SIZE-4);
158     while(readFromFile){
159         appLayer->packet[1] = (appLayer->packet[1] + 1) % 256;
160         llwriteReturn = llwrite(appLayer->serialPortFD, appLayer->packet, readFromFile+4);
161
162          if(llwriteReturn-4){
163             readFromFile =  read(appLayer->fileFD, appLayer->packet + 4, PACKET_SIZE-4);
164             writtenBytes += llwriteReturn - 4;
165         }
166     }
167      sendControlPacket(appLayer, END_PACKET);
168      return writtenBytes;
169 }
170
171 int receiveStartPacket(AppLayer* appLayer){
172     while(!(appLayer->packetSize = llread(appLayer->serialPortFD, appLayer->packet)));
173
174     if(appLayer->packet[0] != START_PACKET)
175         return 1;
176
177     if(appLayer->packet[1] != T_SIZE)
178       return 1;
179
180     int i;
181     appLayer->fileSize = 0;
182     for(i = 0; i < appLayer->packet[2]; i++){
183       appLayer->fileSize += appLayer->packet[3 + i];
184     }
185
186
187     if(appLayer->packet[3 + appLayer->packet[2]] != T_NAME)
188         return 1;
189
190     appLayer->fileName = malloc(appLayer->packet[4 + appLayer->packet[2]]);
191     memcpy(appLayer->fileName, appLayer->packet + 5 + appLayer->packet[2], appLayer->packet[4 +
      ↪  appLayer->packet[2]]);
192
193     return 0;
194 }
195
196 int sendControlPacket(AppLayer* appLayer, unsigned char control){
197     unsigned char i;
198
199     appLayer->packet[0] = control;
200     appLayer->packet[1] = T_SIZE;
201
202
203
204     unsigned char sizeNBytes = appLayer->fileSize / 255;
```

```
205
206     if(appLayer->fileSize % 255){
207       sizeNBytes++;
208
209       for(i = 0; i < sizeNBytes-1; i++){
210         appLayer->packet[3 + i] = 255;
211       }
212       appLayer->packet[3 + sizeNBytes-1] = appLayer->fileSize - 255*(sizeNBytes-1);
213     }
214     else{
215       for(i = 0; i < sizeNBytes; i++){
216         appLayer->packet[3 + i] = 255;
217       }
218     }
219
220
221     appLayer->packet[2] = sizeNBytes;
222     appLayer->packet[sizeNBytes+3] = T_NAME;
223     unsigned int fileNameSize = strlen(appLayer->fileName)+1;
224     appLayer->packet[sizeNBytes+4] = fileNameSize;
225
226
227     memcpy(appLayer->packet + sizeNBytes+5, appLayer->fileName, fileNameSize);
228
229     appLayer->packetSize = 5 + sizeNBytes + fileNameSize;
230
231    if(!llwrite(appLayer->serialPortFD, appLayer->packet, appLayer->packetSize))
232         return 1;
233     return 0;
234   }
235
236   void getFileSize(AppLayer* appLayer){
237     struct stat statBuf;
238
239     fstat(appLayer->fileFD, &statBuf);
240     appLayer->fileSize = statBuf.st_size;
241     printf("FileSize %d\n", appLayer->fileSize);
242   }
243
244   double getElapsedTimeSecs(struct timespec* start, struct timespec* end){
245     return (end->tv_sec + end->tv_nsec/1000000000) - (start->tv_sec + start->tv_nsec/1000000000);
246   }
```

11.3 Ficheiro Excel

Ficheiro	Tamanho(Bytes)				S(FER,a)	S = R/C (R=débitorecebido, bit/s)		a = Tprop/Tf		
pinguim.gif	10968					S =(1-FER)/(1+2*a)				
FER	T_prop (ms)	C (Débito da Ligação)	Tamanho Trama I	Tempos de transmição do Ficheiro						
				Tf1	Tf2	Tf3	Tf(média)	a	Scalculada	Steórica
0	10	19200	512	0,2727	0,2727	0,2727	0,2727	0,0367	0,9317	0,9143
0	20	19200	512	0,3182	0,3182	0,3182	0,3182	0,0629	0,8883	0,8421
0	30	19200	512	0,3182	0,3182	0,3182	0,3182	0,0943	0,8414	0,7805
0	40	19200	512	0,3636	0,3182	0,3636	0,3485	0,1148	0,8133	0,7273
0,25	**10**	**19200**	512	0,3636	0,3636	0,3636	0,3636	0,0275	0,7109	0,6857
0,25	**20**	**19200**	512	0,3636	0,3636	0,5455	0,4242	0,0471	0,6854	0,6316
0,25	**30**	**19200**	512	0,5000	0,4091	0,4545	0,4545	0,0660	0,6625	0,5854
0,25	**40**	**19200**	512	0,4545	0,4545	0,5455	0,4848	0,0825	0,6438	0,5455
0,5	10	19200	512	0,6364	0,4545	0,5455	0,5455	0,0183	0,4823	0,4571
0,5	20	19200	512	0,5909	0,5455	0,5455	0,5606	0,0357	0,4667	0,4211
0,5	30	19200	512	0,7273	0,6364	0,7273	0,6970	0,0430	0,4604	0,3902
0,5	40	19200	512	0,7273	0,6818	0,6818	0,6970	0,0574	0,4485	0,3636
0,75	**10**	**19200**	512	1,1818	0,7727	1,3182	1,0909	0,0092	0,2455	0,2286
0,75	**20**	**19200**	512	1,2273	1,6818	0,6818	1,1818	0,0169	0,2418	0,2105
0,75	**30**	**19200**	512	1,5455	1,2727	1,7273	1,5152	0,0198	0,2405	0,1951
0,75	**40**	**19200**	512	1,5909	1,3182	1,0000	1,3030	0,0307	0,2355	0,1818
0	10	38400	1024	0,2727	0,2727	0,2727	0,2727	0,0367	0,9317	0,9143
0	20	38400	1024	0,2727	0,2727	0,2727	0,2727	0,0733	0,8721	0,8421
0	30	38400	1024	0,2727	0,3636	0,2727	0,3030	0,0990	0,8347	0,7805
0	40	38400	1024	0,3636	0,3636	0,3636	0,3636	0,1100	0,8197	0,7273
0,25	**10**	**38400**	1024	0,3636	0,5455	0,3636	0,4242	0,0236	0,7162	0,6857
0,25	**20**	**38400**	1024	0,4545	0,3636	0,4545	0,4242	0,0471	0,6854	0,6316
0,25	**30**	**38400**	1024	0,4545	0,4545	0,5455	0,4848	0,0619	0,6674	0,5854
0,25	**40**	**38400**	1024	0,4545	0,5455	0,4545	0,4848	0,0825	0,6438	0,5455
0,5	10	38400	1024	0,6364	0,3636	0,4545	0,4848	0,0206	0,4802	0,4571
0,5	20	38400	1024	0,3636	0,6364	0,6364	0,5455	0,0367	0,4658	0,4211
0,5	30	38400	1024	0,6364	0,9091	0,7273	0,7576	0,0396	0,4633	0,3902
0,5	40	38400	1024	0,7273	0,9091	0,4545	0,6970	0,0574	0,4485	0,3636
0,75	**10**	**38400**	1024	1,1818	0,7273	1,2727	1,0606	0,0094	0,2454	0,2286
0,75	**20**	**38400**	1024	1,0000	1,4545	1,5455	1,3333	0,0150	0,2427	0,2105
0,75	**30**	**38400**	1024	1,0909	1,0909	1,0000	1,0606	0,0283	0,2366	0,1951
0,75	**40**	**38400**	1024	1,0000	1,4545	1,1818	1,2121	0,0330	0,2345	0,1818