1º Trabalho Laboratorial

Relatório

Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Trabalho realizado por:

Maria Gonçalves Caldeira (up201704507)

Raul Manuel Fidalgo da Silva Teixeira Viana (up201208089)

18 de outubro de 2020

Índice

Sumário 3
Introdução 3
Arquitetura 3
Estrutura do código 4
Casos de uso principais 5
Protocolo de ligação lógica 5
Protocolo de aplicação 6
Validação 7
Eficiência do protocolo de ligação de dados 8
Conclusões 10
Anexo – código fonte 11
Anexo II – cálculos da eficiência da ligação 37

Sumário

Este relatório foi elaborado no âmbito da unidade curricular de Redes e Computadores, do curso Mestrado Integrado em Engenharia Informática e Computação, para descrever o primeiro trabalho prático, que consistiu no desenvolvimento de uma aplicação capaz de transferir ficheiros de um computador para o outro através de uma porta série.

Assim, é possível afirmar que o trabalho foi concluído com sucesso, visto que o objetivo estabelecido foi cumprido, tendo sido escrita uma aplicação funcional e capaz de transferir ficheiros sem perda de dados.

Introdução

O objetivo principal deste trabalho foi implementar um protocolo de ligação de dados, de acordo com o guião fornecido, testando-o com uma aplicação simples de transferência de ficheiros. A transferência foi efetuada recorrendo a uma porta série do tipo RS-232. Para descrever a forma como a aplicação foi escrita e o seu funcionamento foi desenvolvido este relatório que pretende também relacionar a componente teórica com o trabalho desenvolvido e tem a seguinte estrutura:

Arquitetura: Exibição dos blocos funcionais e interfaces presentes;

Estrutura do código: demonstração das APIs, principais estruturas de dados, principais funções e sua relação com a arquitetura;

Casos de uso principais: identificação destes e demonstração das sequências de chamada de funções;

Protocolo de ligação lógica: identificação dos principais aspetos funcionais e descrição da estratégia de implementação dos mesmos com apresentação de extratos de código;

Protocolo de aplicação: identificação dos principais aspetos funcionais e descrição da estratégia de implementação dos mesmos com apresentação de extratos de código;

Validação: descrição dos testes efetuados com apresentação quantificada dos resultados;

Eficiência do protocolo de ligação de dados: caraterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido;

Conclusão: resumo da informação apresentada nas secções anteriores e conclusões finais.

Arquitetura

O trabalho está dividido em dois blocos funcionais, tendo sido criado um emissor e um recetor. Cada um destes blocos incorpora a camada de aplicação. Depois existe um outro bloco comum ao emissor e ao recetor, que incorpora a camada da ligação de dados. Desta forma foi implementada o isolamento entre camadas.

Estrutura do código

O código está dividido em três ficheiros de código principais, correspondentes às funções necessárias para a execução do programa. Assim, existe o ficheiro *send.c* – responsável pelas funções de envio da informação referente ao ficheiro a transferir, o ficheiro *receive.c* – responsável pelas funções de recebimento. Existe ainda um terceiro ficheiro principal: *link_layer.c*, responsável pelo envio, receção e processamento das tramas inerentes ao protocolo descrito. Existe ainda um header file, *constants.h* no qual estão declaradas as constantes necessárias aos ficheiros *send.c* e *receive.c* e ainda um ficheiro *alarm.c* responsável pela implementação do alarme e os respetivos header files com a declaração de todas as funções.

Funções principais:

Sender

- *sendControlPacket* – constrói e envia o pacote de controlo que sinaliza o início da transferência dos dados de informação;

- *sendFile* – constrói e envia os pacotes de informação;

- *printStats* – imprime os dados da ligação.

Receiver

- *readControlPacket* – recebe e lê o pacote de controlo que sinaliza o início da transmissão de dados;

- *receiveFile* – recebe os pacotes de informação;

- *processData* - guarda em disco o ficheiro recebido e confirma a receção do pacote de controlo que sinaliza o fim da transmissão de dados;

Link_layer

- *llopen* – inicia a ligação e retorna o file descriptor da porta utilizada;

- *llclose* – fecha a ligação;

- *startConnection* – abre a porta de série e guarda a sua configuração;

- *closeConnection* – fecha a porta de séries e volta a colocar as configurações no seu estado inicial;

- *readMessage* – lê a trama de controlo e envia-a para a state machine;

- *COM_currentMachine* – state machine responsável por validar as tramas de controlo;

- *Data_currentMachine* – state machine responsável por validar as tramas de informação;

- *llwrite* – responsável por receber o pacote da camada superior, construir a trama, enviá-la para o buffer da placa física e receber a resposta correspondente;

- *sendControl* – envia o pacote de controlo de início da ligação;

- *llread* – responsável por ler a trama recebida pela placa física, processá-la e retornar para a camada superior o pacote de dados;

- *readFrame* – lê a trama da placa física e valida-a através da chamada a *data_currentMachine*;

- *destuffFrame* – realiza o de-stuff da trama;

- *confirmIntegrity* – confirma se o BCC1 e o BC2 recebidos são iguais ao esperado.

Variáveis Globais

- FILENAME – caminho para o ficheiro a transferir

- SENDER PORT – porta para envio´

- RECEIVER_PORT – porta a receber

- BAUDERATE – capacidade da ligação

- MAX_CHUNK_SIZE – tamanho máximo do pacote de dados

- T_PROP – tempo simulado de propagação adicionado

- fd – file descriptor da placa física

- t0 e t1 – estruturas do tipo timespec para medição do tempo total de envio

Casos de uso principais

A aplicação foi desenhada como um programa de teste, sem interface de utilizador. Assim, todas as variáveis personalizáveis são definidas no ficheiro *constans.h*. Estas variáveis são as portas a serem utilizadas, o caminho relativo do ficheiro a ser transferido, e as variáveis da própria transmissão como o bauderate, tempos de timeout e número de tentativas repetidas. Deste modo o principal caso de uso da aplicação é a transferência do ficheiro, via porta série, entre dois computadores, o transmissor e o recetor.

A transmissão de dados dá-se com a seguinte sequência:

- Configuração da ligação entre os dois computadores.

- Transmissor abre o ficheiro a enviar.

- Estabelecimento da ligação.

- Transmissor envia dados.

- Recetor recebe os dados.

- Recetor guarda os dados num ficheiro com o mesmo nome do ficheiro enviado pelo emissor.

- Terminação da ligação.

Protocolo de ligação lógica

LLOPEN int llopen(int type);

Esta função tem a responsabilidade de estabelecer a ligação entre o emissor e o recetor.

No emissor, esta função envia a trama de controlo SET através da chamada da função ***sendControl*** e ativa o temporizador que é desativado depois de receber resposta (UA). Se não receber resposta dentro de um tempo definido em TIMEOUT, SET é reenviado. Este mecanismo de retransmissão é repetido um número máximo de vezes definido em MAX_TRIES, se este número for atingido o programa termina.

No recetor, esta função espera pela chegada de uma trama de controlo SET, ao queresponde com uma trama do tipo UA.

Tanto no recetor como no emissor é utilizada a função ***readMessage*** para ler a trama de controlo. A validação é feita por sua vez recorrendo à função ***COM_currentMachine***.

As escritas são feitas trama a trama, no entanto a leitura é feita carater a carater.

LLWRITE `int llwrite(int fd, unsigned char packet[], int index);`

Esta função é a responsável por enviar tramas do emissor para o recetor. Inicialmente é construída a trama em volta do pacote recebido. É calculado e inserido o bcc1, é feito o stuffing da trama e finalmente calculado e inserido o bcc2. Por fim é enviada a trama, com recurso à função ***readResponse***, e acionado o mecanismo de espera e reenvio em caso de erro.

LLREAD `int llread(int fd, unsigned char* packet);`

Nesta função, chamada pelo recetor, as tramas são recebidas e processadas. A leitura é feita caracter a caracter, pelo que se torna necessário à sua validação através da função ***data_currentMachine***. Depois desta validação é chamada a função auxiliar ***destuffFrame*** onde é realizado o processo de destuff. Posteriormente a função ***confirmIntegrity*** efetua o cálculo do bcc1 e do bcc2 e compara-os aos recebidos retornando o resultado. Em caso de não se verificarem erros ***llread*** envia por fim uma trama de confirmação e pedido de nova trama, em caso de deteção de erro envia uma trama de rejeição e pedido de reenvio da trama de informação.

LLCLOSE `int llclose(int fd, int type);`

Esta função termina a ligação entre o emissor e o recetor. No emissor é enviada a trama DISC, e esperada uma trama UA. No recetor é esperada uma trama DISC e enviada de seguida uma trama UA.

Por fim é chamada a função ***closeConnection*** que recoloca os parâmetros iniciais da placa física.

Protocolo de aplicação

O protocolo de aplicação implementado tem as seguintes características:

- Envio de um pacote de controlo START e END, respetivamente no início e no fim do envio dos dados, pacotes entes com as informações do tamanho e nome do ficheiro em transferência;

- A divisão do ficheiro em fragmentos a serem enviados, por parte do emissor e a reconstrução destes por parte do recetor;

- Construção dos pacotes de dados, com introdução de *header* contendo o número de sequência e o seu tamanho;

- Abertura e leitura do ficheiro a ser enviado e criação e escrita em disco do ficheiro recebido.

Para o correto funcionamento destas funcionalidades foram implementadas as seguintes funções:

sendControlPacket `int sendControlPacket(int fd, int control_type, FileInfo fileInfo)`

Esta função recebe um argumento *control_type* que define qual o tipo de pacote de controlo a enviar, START ou END. Constrói o pacote respetivo e envia-o.

sendFile `int sendFile(FileInfo fileInfo)`

Esta função recebe como argumento a estrutura *fileInfo* que contém toda a informação do ficheiro e ser enviado. De uma forma consistente vai lendo pedaços do ficheiro, construindo o pacote de dados em volta destes e enviado estes pacotes de dados para a camada de ligação.

readControlPAcket `int readControlPacket()`

Esta função recebe o pacote de controlo START. Depois de descodificar o tamanho e o nome do ficheiro abre o ficheiro em disco para escrita, com as permissões adequadas.

reciveFIle `int receiveFile(FileInfo fileInfo)`

Esta função recebe de forma consistente pacotes de informação contendo pedaços do ficheiro a receber. Se o pacote não for do tipo de controlo END a função processData é chamada para processar os dados contidos no pacote. Pelo contrário, se o pacote não for de dados, significa que a transmissão de dados acabou, a função retorna depois de fechar o ficheiro em disco.

processData `int receiveFile(FileInfo fileInfo)`

Esta função recebe os dados que estavam contidos no pacote de dados e escreve-os em disco.

Validação

De forma a estudar a aplicação desenvolvida, foram efetuados os seguintes testes:

- Envio de ficheiros de vários tamanhos.

- Geração de curto circuito no cabo enquanto se envia um ficheiro.

- Interrupção da ligação por alguns segundos enquanto se envia um ficheiro.

- Envio do um ficheiro com variação na percentagem de erros simulados.

- Envio do um ficheiro com variação do tamanho de pacotes.

- Envio do um ficheiro com variação das capacidades de ligação (*baudrate*).

- Envio de um ficheiro com variação simulada do tempo de propagação.

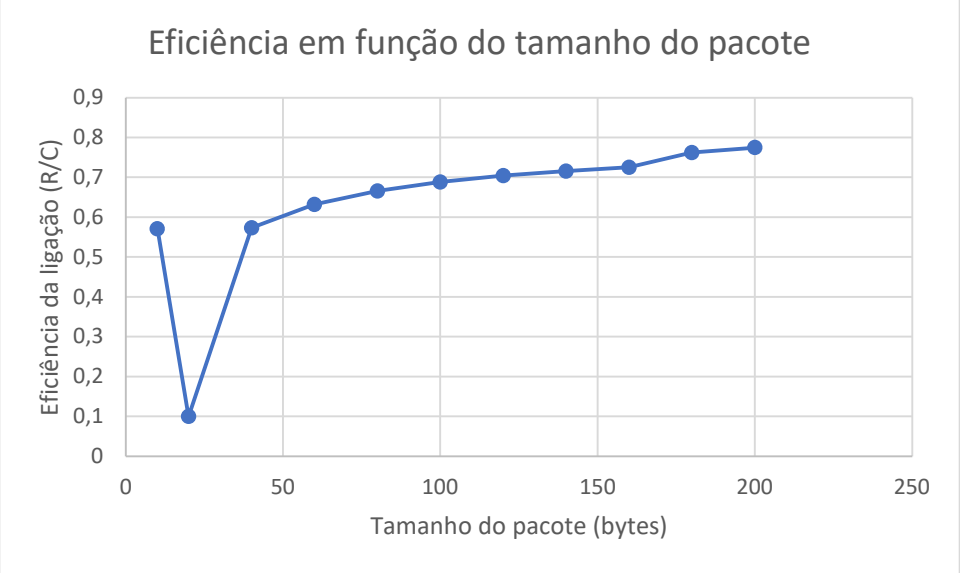
Todos os testes foram concluídos com sucesso.

Eficiência do protocolo de ligação de dados

De forma a avaliar a eficiência do protocolo desenvolvido, foram feitos os seguintes quatro testes e elaborados uma tabela e um gráfico. Os testes foram efetuados variando apenas a condição em análise, mantendo os outros parâmetros fixos nos valores descritos na tabela 1 do Anexo II. Todas as tabelas, contendo os cálculos efetuados estão presentes no Anexo II.

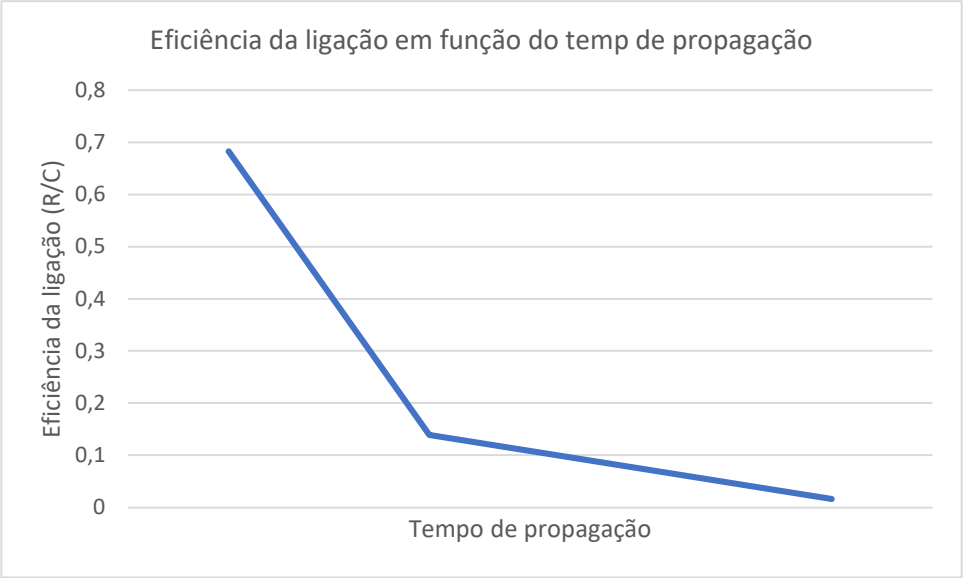
Variação do tamanho das tramas de dados

O gráfico seguinte permite concluir que quanto maior o tamanho de cada pacote, mais eficiente é a aplicação. Isto acontece porque é enviada mais informação de uma vez o que faz com menos tramas sejam enviadas e o *overhead* do seu processamento seja menor, logo o programa executa mais rapidamente. Existe um valor, 40 *bytes*, que muito provavelmente corresponde a um desvio significativo, sendo muito provavelmente, um erro de medição.



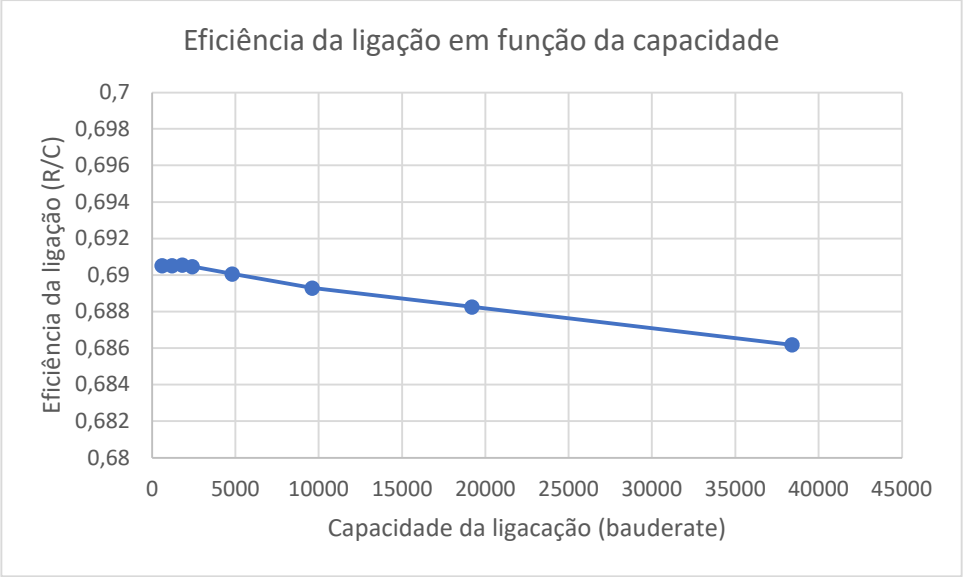
Variação do tempo de propagação

Foi introduzido um atraso de forma a simular o tempo de propagação e o gráfico seguinte demonstra que o aumento do tempo de propagação diminui de forma significativa a eficiência da ligação.



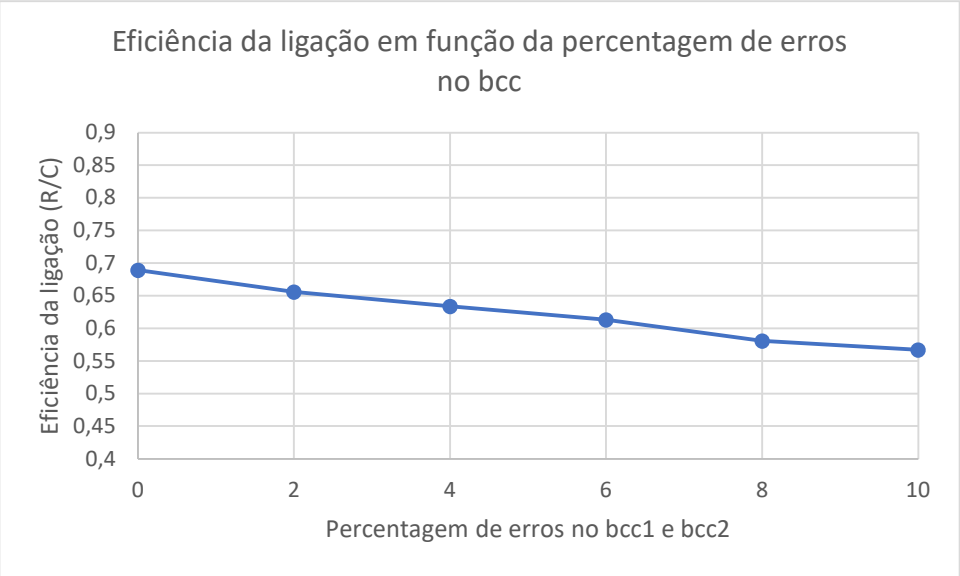
Variação da capacidade da ligação (C)

Com este gráfico podemos concluir que com o aumento da capacidade de ligação, diminui a eficiência. Se bem que a variação na eficiência da ligação é muito pequena e, portanto, a variação da capacidade da ligação tem um impacto muito pequeno na sua eficiência.



Variação do FER

A geração de erros no *BCC1* e no *BCC2* não tem um impacto muito grande na eficiência do programa, se bem que o seu efeito é consistente. Isto deve-se principalmente ao facto de que quando há erros no *BCC1* ou no *BCC2* é enviado imediatamente um pedido de repetição do envio da trama. Deste modo a diminuição da eficiência da ligação deve-se à maior quantidade de tramas que necessariamente têm de ser enviadas quando a percentagem de erros aumenta.



Conclusões

O tema deste trabalho é o protocolo de ligação de dados, que consiste em fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um meio de transmissão, neste caso, um cabo série.

Adicionalmente, foi dado a conhecer o termo independência entre camadas, e cada um dos blocos funcionais da arquitetura da aplicação desenvolvida, *writer* e *reader*, cumpre esta independência. Na camada de ligação de dados não é feito qualquer processamento que incida sobre o cabeçalho dos pacotes a transportar em tramas de Informação. Por outro lado, no que respeita à camada de aplicação, esta não conhece os detalhes do protocolo de ligação de dados, mas apenas a forma como este serviço é acedido.

Assim, o trabalho foi concluído com sucesso, tendo-se cumprido todos os objetivos propostos, sendo que a sua elaboração contribuiu para um aprofundamento do conhecimento teórico e prático acerca da temática de transferência de informação digital.

Anexo – código fonte

contants.h

```
#pragma
once
```

```
/******** Alterável *************/
#define FILENAME "pinguim.gif"   // "pinguim.gif"
#define SENDER_PORT "/dev/ttyS10"
#define RECEIVER_PORT "/dev/ttyS11"
#define BAUDRATE B9600
#define MAX_CHUNK_SIZE 100 //tem de caber o nome do ficheiro e o seu
tamanho no pacote de controlo
                          //para  pinguim.gif min = 21;
#define T_PROP 5000000L // 500000 nanoseconds -> 500 microsends
#define TIME_CORRECTION 200.0 //to seconds
#define PROB_ERROR 0   // em percentagem
/*******************************/

#define _POSIX_SOURCE 1 /* POSIX compliant source */

#define MAX_FRAME_SIZE 2 * MAX_CHUNK_SIZE

#define CONTROL_SIZE 5
#define DATA_PACKET_SIZE 4


#define FALSE 0
#define TRUE 1

#define FLAG 0x7E
#define CONTROL_SET 0x02
#define CONTROL_DISC 0x0b
#define CONTROL_UA 0x07
#define CONTROL 0x03
#define START_CONTROL 0x02
#define END_CONTROL 0x03
#define BCC(X, Y) (X) ^ (Y)

#define C_R0  0x05
#define C_R1  0x85
#define C_REJ0 0x81
#define C_REJ1 0x01

#define C0 0x00
```

```
#define C1 0x40
#define BYTE_STUFF 0x20
#define ESC 0x7d

#define FILE_SIZE_FIELD 0x00
#define FILE_NAME_FIELD 0x01
#define DATA_FIELD 0x01


#define TIMEOUT 3
#define MAX_TRIES 5

#define RECEIVER 2
#define SENDER 1

extern struct termios oldtio,newtio;

typedef struct{
    char* send_fileName;
    char* receive_fileName;
    int open_fd;
    int close_fd;
    int fileSize;
} FileInfo;

enum phase{
  OPENING_CONNECTION,
  SENDING_DATA,
  CLOSING_CONNECTION
};

int fd;
```

send.c

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

```c
#include <strings.h>
#include <time.h>

#include "constants.h"
#include "link_layer.h"
#include "alarm.h"

extern enum phase link_phase;
int fd;
clock_t tic, toc;
int current_percentage_error;


int continueFlag = FALSE;
int numTries = 0;

int sendControlPacket(int fd, int control_type, FileInfo
FileInfo);
int sendFile(FileInfo fileInfo);
void printStats();

int main(int argc, char** argv)
{
    srand(time(NULL));
    current_percentage_error = rand() % 101;
    int c, res;
    struct termios oldtio,newtio;

    (void) signal(SIGALRM, atende);  // instala  rotina que atende
interrupcao

    printf("      -->SEnder<--\n");


    link_phase = OPENING_CONNECTION;
    fd = llopen(SENDER);
    if(fd == -1){
      perror("[ERROR] Could not establish connection\n");
      exit(-1);
    }  //else -> Connection online

    printf("[CONNECTION ONLINE]\n");


    /*    +++++++DATA SEnding+++++++++++++++   */
    FileInfo fileInfo;
```

```
struct stat meta_data;
int file_fd;

//open file
file_fd = open(FILENAME, O_RDONLY);
if(file_fd == -1){
  perror("[ERROR]  Error opening file, aborting\n");
  return -1;
}

if(fstat(file_fd, &meta_data) == -1){
  perror("[ERROR]  Error in fstat\n");
  return -1;
}

fileInfo.fileSize = meta_data.st_size;
fileInfo.open_fd = file_fd;
fileInfo.send_fileName = FILENAME;


//construct and send opening control packet
link_phase = SENDING_DATA;
link_control.N_s = 1;
if(sendControlPacket(fd, START_CONTROL, fileInfo) == -1){
  perror("[ERROR]\n Error sending start control packet\n");
  exit(1);
}

//send file
link_phase = SENDING_DATA;
link_control.N_s = 1;
link_control.framesReceived = 0;
link_control.framesSent = 0;
link_control.RJreceived =0;
link_control.RJsent = 0;
link_control.RRreceived =0;
link_control.RRsent = 0;
//start counting time
tic = clock();

if(sendFile(fileInfo) == -1){
  printf("[ERROR]\n  Error in llwrite - ABORTING\n");
  exit(2);
}
//stop counting time
toc = clock();
```

```
    //construct and send closing control packet
    if(sendControlPacket(fd, END_CONTROL, fileInfo) == -1){
      perror("[ERROR]\n Error sending ending control packet\n");
      exit(-1);
    }

    /*    ++++++++++++++++++++++++++++++++++++   */


    llclose(fd, SENDER);
    printf("[CONNECTION CLOSED]\n");

    printStats();

    return 0;
}

int sendControlPacket(int fd, int control_type, FileInfo
fileInfo){
  unsigned int index = 0;

    unsigned int file_size_length = sizeof(fileInfo.fileSize);
    //[C, T1, L1, ..., T2, L2, ...] = 5 (COntrol Size)
    unsigned char packet[CONTROL_SIZE + file_size_length +
strlen(fileInfo.send_fileName)];

    packet[index++] = control_type;

    //insert file size T1, L1 and value
    packet[index++] = FILE_SIZE_FIELD;
    packet[index++] = file_size_length;
    unsigned char byteArray[file_size_length];
    //transformar int em array de chars
    for (int i = 0; i < file_size_length; i++){
      byteArray[i] = (fileInfo.fileSize >> 8*(file_size_length - 1
- i)); //masking
    }
    //colocar os chars no packet
    for (int i = 0; i < file_size_length; i++){
      packet[index++] = byteArray[i];
    }

    //insert file name
    packet[index++] = FILE_NAME_FIELD;
    packet[index++] = strlen(fileInfo.send_fileName);
```

```
    for (int i = 0; i < strlen(fileInfo.send_fileName); i++){
      packet[index++] = fileInfo.send_fileName[i];
    }
    int res = llwrite(fd, packet, index);
    if (res == -1){
      return -1;
    }


  return 0;
}

int sendFile(FileInfo fileInfo){
  unsigned char buffer[MAX_CHUNK_SIZE];
  unsigned int chunks_sent = 0;
  unsigned int chunks_to_send = fileInfo.fileSize / MAX_CHUNK_SIZE
+ (fileInfo.fileSize % MAX_CHUNK_SIZE != 0); //se na divisão pelo
tamanho máximo sobrarem bytes é necessário adicionar mais um chunk
com menos bytes do que o tamanho máximo

  unsigned int byte_read = 0;
  unsigned int bytes_written = 0;
  unsigned int total = 0;

  printf("[INFO]\n  Sending file %s with %d bytes in %d splited
parts\n", FILENAME, fileInfo.fileSize, chunks_to_send);

  while(chunks_sent < chunks_to_send){
    byte_read = read(fileInfo.open_fd, &buffer, MAX_CHUNK_SIZE);
    unsigned char packet[DATA_PACKET_SIZE + byte_read];

    //construct packet
    // [C, N, L2, L1, P1, ..., Pk]
    packet[0] = DATA_FIELD;
    packet[1] = chunks_sent % 255;
    packet[2] = byte_read / 256;
    packet[3] = byte_read % 256; //última posição é 256 * L2 + L1
    memcpy(&packet[4], &buffer, byte_read); //coloca o conteudo do
buffer nas posições seguintes do packet

    bytes_written = llwrite(fd, packet,byte_read +
DATA_PACKET_SIZE);
    if(bytes_written == -1)return -1;
    total += bytes_written;
    chunks_sent++;
  }
```

```
  printf("[INFO]\n  Sent %d data bytes in %d chunks\n", total -
DATA_PACKET_SIZE * chunks_sent,chunks_sent);
}

void printStats(){
  printf("\n     ***Statistics***\n");
  printf("Number os frames sent: %d\n", link_control.framesSent);
  printf("Number of RR frames received: %d\n",
link_control.RRreceived);
  printf("Number of REJ frames received: %d\n",
link_control.RJreceived);
  printf("     **************\n");
}
```

receive.c

```
#include
"constants.h"
                #include "link_layer.h"

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


                extern enum phase link_phase;
                FileInfo fileInfo;
                struct timespec t0, t1;
                int current_percentage_error;

                int continueFlag = FALSE;
                int numTries = 0;

                int readControlPacket();
                int receiveFile(FileInfo fileInfo);
                void processData(unsigned char* packet, FileInfo fileInfo);
                void printStats(double time_taken);
```

```c
int main(int argc, char** argv)
{
    srand(time(NULL));
    current_percentage_error = rand() % 101;
    int fd,c, res;
    struct termios oldtio,newtio;
    bzero(&fileInfo, sizeof(fileInfo));
    unsigned char packet[MAX_FRAME_SIZE + DATA_PACKET_SIZE];

    printf("      -->RECEIVER<--\n");

    link_phase = OPENING_CONNECTION;
    fd = llopen(RECEIVER);
    if(fd == -1){
      perror("[ERROR] Could not establish connection\n");
      exit(-1);
    }  //else -> Connection online

    printf("[CONNECTION ONLINE]\n");


    /*    +++++++DATA Receiving+++++++++++++++   */

    //receive start control packet
    link_phase = OPENING_CONNECTION;
    link_control.N_s = 0;

    if(readControlPacket() == -1){
      perror("[ERROR]\n Error reading start control packet\n");
      exit(1);
    }
    printf("[INFO]\n  Ready to receive data\n");
    //receive data packets
    link_phase = SENDING_DATA;
    link_control.RJreceived =0;
    link_control.RJsent = 0;
    link_control.RRreceived =0;
    link_control.RRsent = 0;
    link_control.framesReceived =0;

    clock_gettime(0, &t0);

    if(receiveFile(fileInfo) == -1){
      printf("[ERROR]\n  Error in llread\n");
      exit(2);
```

```
    }

    clock_gettime(0, &t1);
    double time_taken = ((double) t1.tv_sec - t0.tv_sec) +
((double)(t1.tv_nsec - t0.tv_nsec) /1000000000L);
    /*    ++++++++++++++++++++++++++++++++++   */

    llclose(fd, RECEIVER);
    printf("[CONNECTION CLOSED]\n");

    printStats(time_taken);

    return 0;
}


int readControlPacket(){
  unsigned char packet[MAX_FRAME_SIZE];
  int res = llread(fd, packet, OPENING_CONNECTION);
  if(res == -1){
    return -1;
  }

  int index = 1; //after flag
  int file_size = 0;

  if( packet[index] != FILE_SIZE_FIELD){
    return -1;
  }
  else{
      index++;
      int file_size_length = packet[index++];
      //transforming chars byte into an int again
      for(int i = 0; i < file_size_length; i++){
          file_size += packet[index++] << 8 * (file_size_length -
1 - i);
      }
  }
  if( packet[index] != FILE_NAME_FIELD) return -1;
  else{
      index++;
      int name_length = packet[index++];
      char file_name[name_length + 1];
      for(int i = 0; i < name_length; i++){
          file_name[i] = packet[index++];
      }
```

```
      file_name[name_length] = '\0';
      //testing
      file_name[0] = '1';
      //
      fileInfo.receive_fileName = file_name;
      remove(fileInfo.receive_fileName);
      fileInfo.close_fd = open(fileInfo.receive_fileName, O_RDWR |
O_CREAT , 777);
      printf("[INFO]\n Prepared to receive file: %s with size:
%d\n", file_name, file_size);
  }
 return 0;
}

int receiveFile(FileInfo fileInfo){
  unsigned char max_buf[MAX_CHUNK_SIZE + DATA_PACKET_SIZE];

  unsigned int bytes_read = 0;
  unsigned int received = 0;
  int aux = 0;
  while(! received){
    if((aux = llread(fd, max_buf, SENDING_DATA)) != 0){

      printf("[INFO]\n  Received packet #%d\n",
link_control.framesReceived);

      bytes_read += aux;
      if(max_buf[0] == DATA_FIELD) {
        processData(max_buf, fileInfo);
      }
      else if(max_buf[0] == END_CONTROL){
        received = 1;
      }
    }
  }
  close(fileInfo.close_fd);
  return 0;
}


void processData(unsigned char* packet, FileInfo fileInfo){

       int dataSize = 256 * packet[2] + packet[3];

       int res = write(fileInfo.close_fd, &packet[4], dataSize);
}
```

```
void printStats(double time_taken){
  printf("\n     ***Statistics***\n");
  printf("Total transfer time: %f seconds\n",time_taken);
  printf("Number of frames received: %d\n",
link_control.framesReceived);
  printf("Number of RR frames sent: %d\n", link_control.RRsent);
  printf("Number of REJ frames sent: %d\n", link_control.RJsent);
  printf("      **************\n");


}
```

Link_layer.h

```
#pragma
once

#include "constants.h"


extern int continueFlag;
extern int numTries;

#define SENDER 1
#define RECEIVER 2

enum state {
       START,
       READ_FLAG,
       READ_CONTROL,
    READ_BCC,
       BCC_OK,
       DATA,
       STOP
};

typedef struct {
       unsigned int N_s;
       unsigned int framesSent;
       unsigned int framesReceived;
       unsigned int RRsent;
       unsigned int RJsent;
       unsigned int RRreceived;
       unsigned int RJreceived;
```

```
}Link_control;

extern Link_control link_control;

int llopen(int type);

int llclose(int fd, int type);

int readMessage(int fd, unsigned char commandExpected[]);

int startConnection(int type);

int closeConnection(int fd);

int llwrite(int fd, unsigned char packet[], int index);

int sendControl();

unsigned char COM_currentMachine(enum state* current, unsigned char
buf);
void data_currentMachine(enum state* current, unsigned char buf);

int llread(int fd, unsigned char* packet, int stage);

int readFrame(int fd, unsigned char* frame);

int  destuffFrame(unsigned char* frame, int frame_length, unsigned char*
final_frame);

int readResponse(int fd);

int confirmIntegrity(unsigned char* final_frame, int final_frame_length,
int stage);
```

Link_layer.c

```
#include
"link_layer.h
"
        #include "constants.h"
        #include "alarm.h"

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

```c
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <strings.h>
#include <time.h>

unsigned char SET[5] = { FLAG, CONTROL, CONTROL_SET, BCC(CONTROL,
CONTROL_SET), FLAG};
unsigned char UA[5] = {FLAG, CONTROL, CONTROL_UA, BCC(CONTROL,
CONTROL_UA), FLAG};
unsigned char DISC[5] = {FLAG, CONTROL, CONTROL_DISC, BCC(CONTROL,
CONTROL_DISC), FLAG};
unsigned char RR1[5] = {FLAG, CONTROL, C_R1, BCC(CONTROL, C_R1),
FLAG};
unsigned char RR0[5] = {FLAG, CONTROL, C_R0, BCC(CONTROL, C_R0),
FLAG};
unsigned char REJ0[5] = {FLAG, CONTROL, C_REJ1, BCC(CONTROL,
C_REJ1), FLAG};
unsigned char REJ1[5] = {FLAG, CONTROL, C_REJ0, BCC(CONTROL,
C_REJ0), FLAG};

struct termios newtio, oldtio;
enum state current;
Link_control link_control;
extern int fd;
extern int current_percentage_error;
int bccFlag = FALSE;


int llopen(int type){

  fd = startConnection(type);
  link_control.N_s = 0;
  int res;
  //sender
  if(type == SENDER){
    //send SET
    printf("[STARTING CONNECTION]\n");
    printf("[SENDING SET]\n");

    do{
```

```
      res = sendControl();
      //receive UA
        setAlarm(TIMEOUT);
      continueFlag = 0;                // activa alarme
      readMessage(fd, UA);
      cancelAlarm();
    }while(continueFlag && numTries <= MAX_TRIES);
    if (numTries >= MAX_TRIES) return -1;
    else numTries = 1;
    cancelAlarm();
  }

    //receiver
    else{
      //receive SET
      readMessage(fd, SET);
      printf("[SET RECEIVED]\n");

      res = write(fd, UA, sizeof(UA));

      printf("[UA SENDED]\n");
    }

    //OK
    return fd;
}


int llclose(int fd, int type){

  //sender
  if(type == SENDER){
    //send DISC
    printf("[CLOSING CONNECTION]\n[INFO]\n  Sending DISC\n");

    do{
      write(fd, DISC, sizeof(DISC));
      //receive UA
      setAlarm(TIMEOUT);
      continueFlag = 0;                  // activa alarme de 3s
      readMessage(fd, UA);
      cancelAlarm();
    }while(continueFlag && numTries <= MAX_TRIES);
    if (numTries >= MAX_TRIES) return -1;
    else numTries = 1;
```

```
    cancelAlarm();
    printf("[INFO]\n  UA received\n");
  }
  //receiver
  else{
    //receive DISC
    setAlarm(TIMEOUT);                  // activa alarme de 3s
    readMessage(fd, DISC);
    cancelAlarm();
    printf("[INFO]\n  DISC received\n");
    //send UA
    printf("[INFO]\n  Sending UA\n");
    write(fd, UA, sizeof(UA));
  }

  if(! closeConnection(fd)) return FALSE;

  return TRUE;
}


int startConnection(int type){

    /*

*****************************************************************
*****
    Open serial port device for reading and writing and not as
controlling tty
    because we don't want to get killed if linenoise sends CTRL-C.

*****************************************************************
******
  */
    int fd;
    if(type == SENDER) fd = open(SENDER_PORT, O_RDWR | O_NOCTTY );
    else fd = open(RECEIVER_PORT, O_RDWR | O_NOCTTY );

    if (fd <0) {perror(SENDER_PORT); return -1; }

    if ( tcgetattr(fd,&oldtio) == -1) { /* save current port
settings */
      perror("tcgetattr");
      return -1;
    }
```

```
    bzero(&newtio, sizeof(newtio));
    newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
    newtio.c_iflag = IGNPAR;
    newtio.c_oflag = 0;

    /* set input mode (non-canonical, no echo,...) */
    newtio.c_lflag = 0;

    newtio.c_cc[VTIME]    = 0;   /* inter-character timer unused
*/
    newtio.c_cc[VMIN]     = 1;   /* blocking read until 5 chars
received */

  /*
    VTIME e VMIN devem ser alterados de forma a proteger com um
temporizador a
    leitura do(s) pr?ximo(s) caracter(es)
  */
    tcflush(fd, TCIOFLUSH);

    if ( tcsetattr(fd,TCSANOW,&newtio) == -1) {
      perror("tcsetattr");
      return -1;
    }
    printf("New termios structure is set\n");
  /*******************************************/
  return fd;
}


int readMessage(int fd, unsigned char commandExpected[]){
  current = START;

  unsigned char buf[1];
  int res;
  while (current != STOP){
    res = read(fd, buf, 1);
    COM_currentMachine(&current, buf[0]);
  }
}


unsigned char COM_currentMachine(enum state* current, unsigned
char buf){
  unsigned char control_byte;
```

```
  switch (*current){
                        case START:
                                if (buf==FLAG) *current = READ_FLAG;
        else *current=START;
                        break;

                        case READ_FLAG:
                                if(buf == CONTROL) *current =
READ_CONTROL;
        else if(buf==FLAG)
                                        *current = READ_FLAG;
                                else
                                        *current= START;
                        break;

                        case READ_CONTROL:
                                if((buf == CONTROL_SET) || (buf ==
CONTROL_UA) || (buf = CONTROL_DISC) || (buf == C0) || (buf ==
C1)){
          control_byte = buf;
          *current=READ_BCC;
        }
                                else if(buf==FLAG){
                                        *current=READ_FLAG;
        }
                                else
                                        *current=START;
                        break;

                        case READ_BCC:
                                if(buf == BCC(CONTROL, control_byte))
*current=BCC_OK;
        else if (buf==FLAG) *current=READ_FLAG;
                                else
                                        *current=START;
                        break;

                        case BCC_OK:
                                  if(buf==FLAG){
            *current = STOP;
          }
          else *current = FLAG;
                        break;

              }
  return control_byte;
```

```
}

void data_currentMachine(enum state* current, unsigned char buf) {
  unsigned char control_byte;

       switch(*current) {
              case START:
                     if(buf == FLAG){
                            *current = READ_FLAG;
                     }
      else *current=START;
                     break;
              case READ_FLAG:
                            if(buf == CONTROL) *current =
READ_CONTROL;
        else if(buf==FLAG)
                                   *current = READ_FLAG;
                            else
                                   *current= START;
                     break;
              case READ_CONTROL:
                            if((buf == C0) || (buf == C1)){
          control_byte = buf;
          *current=READ_BCC;
        }
                            else if(buf==FLAG)

                                   *current=READ_FLAG;
                            else
                                   *current=START;
                     break;
              case READ_BCC:
                            if(buf == BCC(CONTROL, control_byte))
*current=BCC_OK;
        else if (buf==FLAG) *current=READ_FLAG;
                            else
                                   *current=START;
      break;
    case BCC_OK:
      if(buf!=FLAG){
        *current = DATA;
        return;
      }
      else *current=START;

      break;
```

```
    case DATA:
      if(buf==FLAG){
        *current = STOP;
        return;
                    }
      break;

    case STOP:
      break;
              default:
                     break;
       }
}


int closeConnection(int fd){

  //Voltar a colocar a estrutura termios no estado inicial
  sleep(2);
  if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
    perror("tcsetattr");
    return FALSE;
  }
  close(fd);
  return TRUE;
}


int llwrite(int fd, unsigned char packet[], int packet_size){
  if(link_control.N_s == 0) link_control.N_s = 1;
  else link_control.N_s = 0;

  int continueFlag = TRUE;
  int res;
  unsigned int framePosition;;
  unsigned int tries = 0;
  unsigned char frame[2 * packet_size + 6]; // 6 =
F+A+C+BCC1+BCC2+F || 2*packet para assegurar que existe espaço
suficiente para byte stuffing



    framePosition = 4;
    //compose frame
    //frame: [F, A, C, BCC1, [packet], BCC2, F]
```

```
    //frame header
    frame[0] = FLAG;
    frame[1] = CONTROL;
    if(link_control.N_s == 0) frame[2] = C0;
    else frame[2] = C1;
    frame[3] = BCC(CONTROL, frame[2]);

    //process data

    unsigned int packetPosition = 0;
    unsigned char current_packet_char;

    while(packetPosition < packet_size){
      current_packet_char = packet[packetPosition++];

      if(current_packet_char == FLAG || current_packet_char ==
ESC){
        frame[framePosition++] = ESC;
        frame[framePosition++] = current_packet_char ^ BYTE_STUFF;
      }
      else frame[framePosition++] = current_packet_char;

    }
    unsigned char bcc2 = 0;
    for (int i = 0; i < packet_size; i++){//packing packet in
frame
      bcc2 ^= packet[i];
    }
    //frame footer
    if(bcc2 == FLAG || bcc2 == ESC){
      frame[framePosition++] = ESC;
      frame[framePosition++] = bcc2 ^ BYTE_STUFF;
    }
    else frame[framePosition++] = bcc2;
    frame[framePosition++] = FLAG;

  do{

    res = write(fd, frame, framePosition);

    setAlarm(TIMEOUT);
    continueFlag = 0;

    if(readResponse(fd) == -1){
      cancelAlarm();
      continueFlag = 1;
```

```
      link_control.RJreceived++;
      printf("[INFO]\n  Received REJ #%d\n",
link_control.RJreceived);
      continue;
    }
  }while(continueFlag && numTries <= MAX_TRIES);
  if (numTries >= MAX_TRIES) return -1;
  else numTries = 1;
  cancelAlarm();

  printf("[INFO]\n Sent frame number %d with size %d\n",
link_control.framesSent, framePosition);
  link_control.framesSent++;
return res;
}


int sendControl(){
  int res = write(fd, SET, sizeof(SET));
  return res;
}


int llread(int fd, unsigned char* packet, int stage){
  unsigned char frame[MAX_FRAME_SIZE];
  unsigned char control_field = 0x00;
  int done = FALSE;
  int frame_length = 0;
  int packet_length = 0;

  while(! done){
    frame_length = readFrame(fd, frame);
    if(frame_length == -1){
      return -1;
    }

    // destuff frame
    unsigned char final_frame[frame_length];
    int final_frame_length = destuffFrame(frame, frame_length,
final_frame);

    control_field = frame[2];
    // //confirm data Integrity
    if(! confirmIntegrity(final_frame, final_frame_length,
stage)){
```

```
      printf("[ERROR]\n  Packet with error, asking re-
emission\n");
      if(control_field == C0){
        write(fd, REJ1, sizeof(REJ1));
        link_control.RJsent++;
        printf("[INFO]\n REJ1 sent\n");
      }
      else if(control_field == C1){
        write(fd, REJ0, sizeof(REJ0));
        link_control.RJsent++;
        printf("[INFO]\n REJ0 sent\n");
      }
      return 0;
    }
    else{
      // //Get the packet from within the frame

      for (int i = 4; i < final_frame_length - 2; i++) {
        packet[packet_length] = final_frame[i];
        packet_length++;
      }

      //simulate propagation time
      nanosleep((const struct timespec[]){{0, T_PROP}}, NULL);

      // //send proper response()
      if(control_field == C1){
        write(fd, RR0, sizeof(RR0));
        link_control.RRsent++;
        link_control.framesReceived++;
        printf("[INFO]\n  RR0 sent\n");
        done == TRUE;
        break;
      }
      else{
        write(fd, RR1, sizeof(RR1));
        link_control.RRsent++;
        link_control.framesReceived++;
        printf("[INFO]\n  RR1 sent\n");
        done == TRUE;
        break;
      }
    }
  }
  return packet_length;
}
```

```c
int readFrame(int fd, unsigned char* frame){
  enum state current = START;
  unsigned char byte_read;
  int position = 0;

  while (current != STOP){
    read(fd, &byte_read, 1);
    data_currentMachine(&current, byte_read);
    frame[position++] = byte_read;
  }

  return position;
}


int  destuffFrame(unsigned char* frame, int frame_length, unsigned
char* final_frame){

  final_frame[0] = frame[0];
  final_frame[1] = frame[1];
  final_frame[2] = frame[2];
  final_frame[3] = frame[3]; // FLAG, A, C, BCC1

  int j = 4, i = 4;

  while(i < frame_length - 1){
    if(frame[i] != ESC){
      final_frame[j++] = frame[i];
    }
    else{
      i++;
      if(frame[i] == (FLAG ^ BYTE_STUFF)) final_frame[j++] = FLAG;
      else if (frame[i] == (ESC ^ BYTE_STUFF)) final_frame[j++] =
ESC;
    }
    i++;
  }

  final_frame[j++] = frame[i++];

  return j;
}

int readResponse(int fd){
```

```
  unsigned char byte_read, control_field;
  current = START;
  while(current != STOP && !continueFlag){
    int res = read(fd, &byte_read, 1);
    COM_currentMachine(&current, byte_read);
    if(current == READ_BCC){
      control_field = byte_read;
    }
  }
  if(control_field == C_R0 && link_control.N_s == 1){
    return 0;
  }
  else if(control_field == C_R1 && link_control.N_s == 0){
    return 0;
  }
  else{
    return -1;
  }
}

int confirmIntegrity(unsigned char* final_frame, int
final_frame_length, int stage){
  unsigned char adress_field = final_frame[1];
  unsigned char control_field = final_frame[2];
  unsigned char BCC1 = final_frame[3];


  current_percentage_error = rand() % 101;
  if(current_percentage_error < PROB_ERROR && stage ==
SENDING_DATA && bccFlag == TRUE){
    printf("[INFO]\n  Forced BCC1 error\n");
    bccFlag= FALSE;
    return FALSE;
  }


  if((BCC1 == BCC(adress_field, control_field)) && (control_field
== C0 || control_field == C1)){
  //calculate expected bcc2 ( data packet is between 4 and size -
2 of frame)
    unsigned char expected_bcc2 = 0;
    for( int i = 4; i < final_frame_length - 2; i++){
      expected_bcc2 ^= final_frame[i];
    }
    unsigned char bcc2 = final_frame[final_frame_length - 2];
```

```
            current_percentage_error = rand() % 101;
            if(current_percentage_error < PROB_ERROR && stage ==
        SENDING_DATA  && bccFlag== TRUE){
            printf("[INFO]\n  Forced BCC2 error\n");
            bccFlag = FALSE;
            return FALSE;
            }

            if(bcc2 != expected_bcc2){
              printf("[ERROR]\n  Error in bcc2\n");
              return FALSE;
            }
          }
          else if ((control_field != C1) || (control_field != C0)){
            printf("[ERROR]\n  Error in control field received\n");
            return FALSE;
          }
          bccFlag = TRUE;
          return TRUE;
        }
```

Alarm.h

```
#pragma
once

        #include <stdio.h>
        #include <signal.h>
        #include <unistd.h>

        extern int continueFlag;
        extern int numTries;

        void setAlarm();

        void cancelAlarm();

        void atende();
```

Alarm.c

```
#include
<stdlib.h>

#include "alarm.h"
#include "constants.h"
#include "link_layer.h"


enum phase link_phase;

void atende(int signal) {
  if(signal != SIGALRM)
    return;

  printf("[TIMEOUT]\n  Alarm #: %d\n", numTries + 1);
       continueFlag = 1;
       numTries++;
}

void setAlarm() {
  // set sigaction struct
       struct sigaction sa;
       sa.sa_handler = &atende;
       sigemptyset(&sa.sa_mask);
       sa.sa_flags = 0;

       sigaction(SIGALRM, &sa, NULL);

       continueFlag = 0;

       alarm(TIMEOUT); // install alarm
}

void cancelAlarm() {
  // unset sigaction struct
       struct sigaction action;
       action.sa_handler = NULL;

       sigaction(SIGALRM, &action, NULL);

       alarm(0); // uninstall alarm
}
```

Anexo II – cálculos da eficiência da ligação

Valores de referência	
Total de bytes do ficheiro	10968
Total de bits do ficheiro	87744
C (baudrate)	38400
Tamanho de pacote	200
T_{prop} simulado	0

Tabela 1: valores base

Probabilidade de erro (bcc1 + bcc2)	Tempo (s)	R (bits/s)	S (R/C)
0 + 0	6,63	13234,38914	0,689291101
2 + 2	6,97	12588,80918	0,655667145
4 + 4	7,21	12169,76422	0,633841886
6 + 6	7,455	11769,81891	0,613011402
8 + 8	7,87	11149,17408	0,58068615
10 + 10	8,06	10886,35236	0,566997519

Tabela 2: Valores da eficiência da ligação para diferentes quantidades de erro nos bcc

Tamanho de pacote	Tempo (s)	R (bits/s)	S (R/C)
20	9,98	1923,847695	0,100200401
40	7,97	11009,28482	0,573400251
60	7,23	12136,09959	0,63208852
80	6,87	12772,0524	0,665211063
100	6,64	13214,45783	0,688253012
120	6,49	13519,87673	0,704160247
140	6,39	13731,4554	0,715179969
160	6,30	13927,61905	0,725396825
180	6,00	14624	0,761666667
200	5,90	14871,86441	0,774576271

Tabela 3: eficiência da ligação para diferentes tamanhos do pacote

C (baudrate)	Tempo (s)	R (bits/s)	S (R/C)
600	211,78	414,3167438	0,690527906
1200	105,89	828,6334876	0,690527906
1800	70,59	1243,008925	0,690560514
2400	52,95	1657,110482	0,690462701
4800	26,49	3312,344281	0,690071725
9600	13,26	6617,19457	0,689291101
19200	6,64	13214,45783	0,688253012
38400	3,33	26349,54955	0,686186186

Tabela 4: eficiência da ligação para diferentes capacidades

Tempo de propagação (ns)	Tempo (s)	R (bits/s)	S (R/C)
$5*10^2$	6,69	13115,69507	0,683109118
$5*10^3$	7,2	2666,666667	0,138888889
$5*10^4$	12,90	1488,372093	0,07751938
$5*10^5$	62,4	307,6923077	0,016025641

Tabela 5: eficiência da ligação para diferentes tempos de propagação