FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

1º Trabalho Laboratorial

Catarina Justo dos Santos Fernandes - up201806610

Gustavo Sena Mendes - up201806078

Índice

Sumário

Este trabalho foi realizado no âmbito da Unidade Curricular de Redes de Computadores do curso do Mestrado Integrado em Engenharia Informática e Computação. O objetivo do trabalho consistiu em criar uma aplicação que permitisse a transferência de dados entre dois computadores ligados por uma porta de série e que conseguisse ultrapassar possíveis erros de comunicação e desconexões da ligação.
O trabalho foi realizado com sucesso, uma vez que foi desenvolvida uma aplicação capaz de transferir um ficheiro sem perda de dados, cumprindo assim o objetivo estabelecido.

Introdução

O objetivo principal do trabalho consistiu na implementação de um protocolo de ligação de dados, seguindo a especificação do guião fornecido, e no teste do protocolo com uma aplicação de transferência de ficheiros.
O relatório tem como objetivo descrever como é que a aplicação foi implementada e o seu funcionamento, detalhando a componente teórica do trabalho. Assim, o relat´rio terá a seguinte estrutura:

Arquitetura - blocos funcionais e interfaces;

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

Casos de uso principais - identificação; sequências de chamada de funções;

Protocolo de ligação lógica - identificação dos principais aspectos funcionais; descrição da estratégia de implementação destes aspectos com apresentação de extratos de código;

Protocolo de aplicação - identificação dos principais aspectos funcionais; descrição da estratégia de implementação destes aspectos com apresentação de extractos de código;

Validação - descrição dos testes efectuados com apresentação quantificada dos resultados, se possível;

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

Conclusões - síntese da informação apresentada nas secções anteriores; reflexão sobre os objectivos de aprendizagem alcançados;

Arquitetura

O nosso trabalho foi desenvolvido com duas camadas lógicas independentes: a camada da ligação de dados e camada da aplicação.
A camada da aplicação é a camada responsável pelo envio e receção de ficheiros e pela interação com o utilizador. Nesta camada também é feito o processamento dos pacotes recebidos tal como a distinção entre pacotes de controlo e de dados e processamento dos cabeçalhos.

A camada da ligação de dados é responsável pela a abertura, fecho, leitura e escrita na porta de série. Também é responsável por fazer o stuffing e destuffing das tramas e a respetiva delimitação. Caso haja a deteção de um erro, esta camada também é responsável pela retransmissão da trama onde ocorreu o erro.

Estrutura do código

A camada da aplicação utiliza a API da camada de estrutura de dados. Essa API é constituída por 4 funções principais:

- **llopen:** abre a porta de série e altera as suas configurações para as pretendidas
- **llclose:** fecha a porta de série e repõe as configurações originais
- **llread:** recebe a trama, faz o destuffing, verifica se a trama tem erro e manda o ACK adequado, RJ se houver erro, RR se não houver
- **llwrite:** faz o stuffing da mensagem, envia-a e espera o ACK adequado. Caso o ACK tenha sido RJ, a função irá reenviar a mensagem. Isto pode-se repetir até ao número máximo de vezes definido no código.

Estruturas de dados

```
typedef struct{
    char *port;  /*Path da porta de série passado pelo user*/
    int fileDescriptor;/*Descritor correspondente à porta série*/
    int status;/*TRANSMITTER | RECEIVER*/
    char *path;
}applicationLayer;
```

```
typedef struct {
  unsigned char control;    /**< @brief The control byte - [DATA] */
  unsigned char sequence;   /**< @brief The sequence byte - index on global data */
  int dataSize;      /**< @brief The size of the data array - [1..PACKET_SIZE] */
  unsigned char data[2*MAX_SIZE]; /**< @brief The data array */

  unsigned char *rawBytes; /**< @brief The array containing unprocessed bytes */
  int rawSize;       /**< @brief The size of the raw_bytes array */
  unsigned char bcc2;
}dataFrame;
```

```
typedef struct {
    unsigned char flag;
    unsigned char control;
    unsigned char address;
    unsigned char bcc1;
    unsigned char bcc2;
    unsigned char * data; //after stuffing
    unsigned char * rawData;  //before stuffing
    int size;
    int rawSize;
}infoFrame;
```

```
typedef struct {
  unsigned char control;
  unsigned char *fileSize;
  unsigned char *fileName;
  int filesizeSize;
  int filenameSize;

  unsigned char *rawBytes;
  int rawSize;
}controlFrame;
```

Casos de uso principais

Escolha do ficheiro a enviar através da interface
O utilizador poderá configurar no emissor qual o ficheiro que quer enviar ao recetor. Para tal terá que introduzir o seguinte comando:

./application -p <port> -r/-w <file_path>

Em que

- *application* representa o nome da aplicação
- *-p <port>* representa a porta (*-p /dev/ttyS0,* por exemplo)
- *-r/-w* é a flag que indica se o programa corre como emissor (*-w*) ou como recetor (*-r*), neste caso será *-r*
- *<file_path>* representa o caminho do ficheiro a ser enviado

Envio do ficheiro do emissor para o recetor
A transmissão de dados dá-se com a seguinte sequência:

- Estabelecimento e configuração da ligação da porta de série
- Emissor abre o ficheiro a enviar e envia o pacote START.
- Recetor recebe o pacote START
- Emissor envia dados.
- Recetor recebe os dados.
- Recetor guarda os dados num ficheiro com uma variação do nome do ficheiro enviado pelo emissor, neste caso “cloned_%”
- Emissor envia o pacote STOP
- Recetor recebe o pacote STOP
- Término da ligação.

Protocolo de ligação lógica

O protocolo da ligação lógica implementado é responsável por:

1. Estabelecer a ligação da porta de série, começando por guardar os parâmetros iniciais da ligação para depois os substituir pelos parâmetros pretendidos
2. Transferir os dados pela porta de série, fazendo o stuffing e destuffing dos mesmos
3. Verificar os erros e controlar do fluxo de dados

Para o implementar recorremos às seguintes funções principais:

llopen
Esta função começa por estabelecer a ligação entre dois computadores através da porta de série abrindo-a e alterando as suas configurações para as pretendidas.
Caso a função seja chamada pelo recetor, a função chama *openReader()* que vai aguardar pela receção de uma frame SET, enviada pelo emissor, para depois enviar uma frame UA como resposta. Caso haja um erro, a função chama *llclose()* que termina a ligação.
O envio e receção das tramas de supervisão SET e UA é feito pelas funções *sendSupervisionFrame(),* que constrói e envia uma trama, e *receiveSupervisionFrame(),* que lê e processa uma trama através de uma máquina de estados. Essas funções aceitam como parâmetros o descritor da porta e o tipo de trama que querem enviar/receber.

```c
int openReader(int fd){

    if(receiveSupervisionFrame(fd,SET) >= 0){
        sendSupervisionFrame(fd,UA);
    }
    else{
        printf("Did not UA. exited program\n ");
        llclose(fd, RECEIVER);
        exit(-1);
    }

    //printf("Reader received SET frame and sent UA successfully\n");
    return 0;
}
```

Caso a função seja chamada pelo emissor, a função chama *openWriter()* que vai criar uma trama SET e enviá-la para o emissor, ativando um alarme com duração predefinida no código e ficando à espera de receber uma trama UA como resposta. Se o emissor não obtiver a resposta pretendida, o alarme é desencadeado e conta como um timeout. O ciclo é repetido até o emissor receber a resposta ou até o número máximo de timeouts for atingido. Se não tiver obtido resposta no final de todas as tentativas a ligação é terminada por chamada da função *llclose()* e é retornado -1.

```c
int openWriter(int fd){
    do{
        sendSupervisionFrame(fd,SET);
        alarm(ALARM_TIME);
        setAlarmFlag(0);

        if(receiveSupervisionFrame(fd, UA)>=0){
            resetAlarm();
            return 0;
        }

        if(getAlarmFlag()){
            printf("Timed Out\n");
        }
    }while(getAlarmCounter()<3);
    resetAlarm();

    perror("Error retriving supervision frame\n");
    llclose(fd,TRANSMITTER);
    exit(-1);
}
```

llclose

Do lado do emissor, esta função chama *closeWriter()*, que envia uma trama DISC e fica a aguardar uma resposta DISC do recetor. Caso receba essa resposta, envia uma trama UA para o recetor e retorna 0. Caso não receba a resposta apropriada, há um timeout e o ciclo repete-se até atingir o número máximo predefinido, em semelhança à função *openWriter()*. O recetor chama a função *closeReader()*, que aguarda até receber a trama DISC enviada pelo emissor e em seguida envia uma trama DISC ao emissor. Se após isso receber uma trama UA vinda do emissor a função retorna 0 mas se isso não acontecer, é assinalado um timeout e o ciclo recomeça, de forma semelhante à função *closeWriter()* descrita anteriormente.

```c
int closeWriter(int fd){
    do{
        sendSupervisionFrame(fd,DISC);
        alarm(ALARM_TIME);
        setAlarmFlag(0);

        if(receiveSupervisionFrame(fd, DISC)){
            sendSupervisionFrame(fd,UA);
            alarm(0);
            setAlarmCounter(0);
            printf("\nClosed Writer\n");
            return 0;
        }

        if(getAlarmFlag()){
            printf("Timed Out\n");
        }
    }while(getAlarmCounter()<3);
    setAlarmCounter(0);
    printf("Error recieving DISC Frame...\n");
    return -1;
}
```

```c
int closeReader(int fd){
    do{
        alarm(ALARM_TIME);
        setAlarmFlag(0);
        int resp;
        while((resp=receiveSupervisionFrame(fd, DISC)) <-0){
            printf("Error recieving DISC Frame...\n");
        }
        if(sendSupervisionFrame(fd,DISC)!=5){
            printf("Error Sending DISC\n");
            continue;
        }
        if(receiveSupervisionFrame(fd, UA)==TRUE){
            resetAlarm();
            printf("\nClosed Reader\n");
            return 0;
        }

        if(getAlarmFlag()){
            printf("Timed Out\n");
        }
    }while(getAlarmCounter()<3);
    resetAlarm();
    printf("Error recieving UA Frame...\n");
    return -1;
}
```

llread

Esta função faz todo o processamento necessário da trama de informação. Através da função *messageDestuffing()* a trama é lida e é feita a validação do cabeçalho e o destuffing. Após isso, é feita a verificação dos dados através do Block Check Character correspondente ao pacote. Se o pacote for válido é enviada uma trama RR, caso contrário uma trama REJ.

```c
int llread(int fd, char* buffer){
    unsigned char bcc2=0xff;
    infoFrame frame = messageDestuffing(buffer,fd,&bcc2);
    printInfoFrame(frame);

    if(frame.bcc1!=(frame.address ^frame.control) || frame.bcc2 !=bcc2){
        sendSupervisionFrame(fd, CONTROL_RJ(currFrame));
        for (int i= 0; i<frame.rawSize;i++){
            if(i%10==0)
                printf("\n");
            printf("RD%d:%02x ",i,frame.rawData[i]);
        }
        printf("Sent Negative Response\n");
        free(frame.rawData);
        free(frame.data);
        return -1;
    }
    else{
        memcpy(buffer, frame.data,frame.size);
        sendSupervisionFrame(fd, CONTROL_RR(currFrame));
        if(currFrame)
            currFrame--;
        else
            currFrame++;

        free(frame.rawData);
        free(frame.data);
        return frame.size;
    }
}
```

llwrite

Esta função encapsula o pacote recebido numa trama de informação e faz o respetivo stuffing através da função auxiliar *messageStuffing()*. Após isso é feita a escrita do pacote e a espera e processamento da resposta. Caso a trama recebida seja um RR, a função retornará 0 e o próximo pacote poderá ser transferido. Caso tenha sido REJ, a função irá retornar -1, sinal para a função que chamou *llwrite()* a chamar novamente. Se não obtiver resposta, a trama é reenviada, processo que pode ser repetido o núumero de tentativas predefinido no código. Caso esse número seja ultrapassado a função termina com erro.

```c
int llwrite(int fd, unsigned char* buffer,int length){
    infoFrame frame = messageStuffing(buffer, length);
    printInfoFrame(frame);
    int size;
    alarm(ALARM_TIME);
    do{
        if((size=write(fd, frame.rawData, frame.rawSize))>=0){
            printf("Message sent\n");
        }
        else{
            alarm(0);
            setAlarmFlag(0);
            printf("Message not sent\n");
            free(frame.rawData);
            free(frame.data);
            return -1;
        }
        if(getAlarmFlag()){
            alarm(ALARM_TIME);
            setAlarmFlag(0);
        }
        unsigned char response = readSupervisionFrame(fd);
        if(response==0xff)
            continue;
        if(response==CONTROL_RJ(1)||response==CONTROL_RJ(0)){
            alarm(0);
            setAlarmFlag(0);
            printf("Negative response\n");
            free(frame.rawData);
            free(frame.data);
            return -1;
        }
        else if (response==CONTROL_RR(currFrame)){
            resetAlarm();
            if(currFrame)
                currFrame--;
            else
                currFrame++;

            free(frame.rawData);
            free(frame.data);
            return size;
        }
    }while (getAlarmCounter()<RT_ATTEMPTS);
    resetAlarm();
    free(frame.rawData);
    free(frame.data);
    return -2;
}
```

Protocolo de aplicação

O protocolo de aplicação implementado é responsável por:

1. Geração e transferência dos pacotes de controlo e de dados
2. Leitura e escrita do ficheiro a transferir

Para o implementar recorremos às seguintes funções principais:

transmitterApp

Esta função lê o ficheiro a transmitir através da função *lstat().* Depois disso cria uma trama de controlo com base no tamanho do ficheiro que leu anteriormente e no respetivo nome e envia-a ao recetor.

```c
//Generates and sends START control frame
unsigned int L1 = sizeof(fileStat.st_size);  //Size of file
unsigned int L2 = strlen(path);  //Length of file name
unsigned int frameSize = 5 + L1 + L2;
unsigned char *controlFrame = buildControlFrame(START_FRAME, fileStat.st_size, path, L1, L2, frameSize);
int resp;
while((resp=llwrite(fd, controlFrame, frameSize))==-1){
    usleep(STOP_AND_WAIT);
}
if(resp==-2){
    perror("Error sending START frame.\n");
    free(controlFrame);
    return -1;
}
usleep(STOP_AND_WAIT);
```

Após isto, a função entra no ciclo principal do programa e começa a ler bytes do ficheiro a enviar e a encapsulá-los numa trama de informação para enviar posteriormente através da função *llwrite().* Caso llwrite() retorne -1 significa que recebeu uma trama REJ que requer que o pacote atual seja reenviado.

```
//Generates and sends data packets
char *buf=(char*)malloc(sizeof(char)*MAX_SIZE);
unsigned int bytesToSend, noBytes;
unsigned int sequenceNumber = 0;
while((noBytes = read(fileFd, buf, MAX_SIZE-4))){
    bytesToSend = noBytes + 4;
    unsigned char *data=(unsigned char *)malloc(sizeof(unsigned char)*bytesToSend);
    data[0] = DATA;
    data[1] = sequenceNumber % 255;
    data[2] = noBytes /256;
    data[3] = noBytes % 256;
    memcpy(&data[4], buf, noBytes);
    while((resp=llwrite(fd, data, bytesToSend))==-1){
        usleep(STOP_AND_WAIT);
    }
    if(resp==-2){
        perror("Error sending Data frames\n");
        free(data);
        free(controlFrame);
        return -1;
    }
    free(data);
    sequenceNumber++;
    usleep(STOP_AND_WAIT);
}
free(buf);
printf("Number of data frames sent: %d\n", sequenceNumber);
usleep(STOP_AND_WAIT);
```

Por fim, é enviado um end packet para sinalizar o fim dos dados do ficheiro.

```
//Generates and sends END control frame
unsigned char *endControlFrame = buildControlFrame(END_FRAME, fileStat.st_size, path, L1, L2, frameSize);
while((resp=llwrite(fd, endControlFrame, frameSize))==-1){
    usleep(STOP_AND_WAIT);
}
if(resp==-2){
    perror("Error sending END frame.\n");
    free(controlFrame);
    free(endControlFrame);
    return -1;
}
free(controlFrame);
free(endControlFrame);
return 0;
```

receiverApp

Esta função começa por ler o pacote de controlo que o emissor envia usando a função *llread()*. Daí tira o nome do ficheiro a ler, e o seu tamanho final. Depois, enquanto receber pacotes válidos que não sejam a END_FRAME, continua a escrever os bytes de informação para o ficheiro, acabando quando o receber o END_FRAME.

```
// * DATA Frames
unsigned char *fullMessage = (unsigned char*) malloc (fileSize*sizeof(unsigned char));
int index = 0;
int currSequence = -1;
int totalSize=0;
while (state == 1) {
    memset(buff, 0, sizeof(buff));
    printf("Sequence: %d\n",currSequence);
    while ((size = llread(fd, buff)) <0) {
        memset(buff, 0, sizeof(buff));
        printf("Error reading\n");
    }
    if (buff[0] == END_FRAME) {
        state = 2;
        break;
    }
    dataFrame data = parseDataFrame(buff, size);
    totalSize+=data.dataSize;
    if (data.control != DATA) {
        continue;
    }
    for (int i =0;i<data.dataSize;i++){
        fullMessage[index+i] = data.data[i];
    }
    // * caso o numero de sequencia seja diferente do anterior deve atualizar o index
    if (currSequence != data.sequence) {
        currSequence = data.sequence;
        index += data.dataSize;
    }
}

// * END Control Frame
if (state == 2) {
    controlFrame frame = parseControlFrame(buff, size);
    printControlFrame(frame);
    char* name = (char*) malloc ((frame.filenameSize +7) * sizeof(char));
    sprintf(name, "cloned_%s", frame.fileName);
    FILE *fl = fopen(name, "wb");
    if (fl != NULL) {
        fwrite(fullMessage, sizeof (unsigned char), fileSize, fl);
        fclose(fl);
    }
    free(frame.fileSize);
    free(frame.fileName);
    free(name);
}
```

Validação

Foram efetuado os seguintes testes, todos com sucesso:

- Envio do pinguim.gif pela porta de série
- Envio de ficheiros de imagem de diferentes tamanhos
- Envio de um ficheiro variando o tamanho dos pacotes de dados
- Envio de um ficheiro com interrupções da ligação ao desligar e voltar a ligar a porta de série várias vezes
- Envio de um ficheiro com variação no baudrate
- Envio de um ficheiro com uma variação simulada do tempo de propagação
- Envio de um ficheiro com erros simulados no BCC1 e BCC2

Eficiência do protocolo de ligação de dados

De forma a avaliar a eficiência do protocolo, realizamos quatro tipos de testes, variando apenas um parâmetro em cada teste. Os gráficos e as respetivas tabelas com os cálculos auxiliares estão presentes no Anexo II.

Variação da eficiência relativamente ao tamanho da trama de dados
Concluímos que quanto maior é o tamanho do pacote de dados maior é a eficiência, no entanto, como a forma do gráfico se assemelha a um gráfico logarítmico, a certo ponto o tamanho da trama de dados deixará de influenciar significativamente a eficiência.

Variação da eficiência relativamente ao tempo de propagação
Concluímos que quanto maior o tempo de propagação, menor será a eficiência, sendo um fator significativo assim que o tempo de propagação ultrapassa o valor de 0.01 segundos.

Variação da eficiência relativamente à capacidade de ligação (C)
Concluímos que quanto maior é a capacidade de ligação, menor vai ser a eficiência, diminuindo linearmente ao longo do gráfico..

Variação da eficiência relativamente ao FER
Concluímos que quanto maior é a percentagem de erros induzidos no programa menor vai ser a eficiência, diminuindo linearmente ao longo do gráfico.

Conclusões

Sintetizando o que foi apresentado no relatório, este protocolo tem duas camadas independentes entre si, a camada da aplicação e a camada da ligação de dados. Na camada da aplicação não é conhecido o funcionamento interno dos mecanismos de transmissão e proteção das tramas e na camada de ligação de dados não existe qualquer distinção entre pacotes de controlo e de dados nem é feito qualquer processamento que incida sobre os cabeçalhos dos pacotes que transportam tramas de informação.
Concluindo, a realização do protocolo foi completada com sucesso, tendo se cumprido todos os objetivos apesar da dificuldade acrescida de acesso aos laboratórios devido à situação atual do covid-19.

Anexo I - Código fonte

alarme.h

```c
#pragma once

void setAlarm();

int getAlarmFlag();

int getAlarmCounter();

void setAlarmFlag(int flag);

void setAlarmCounter(int count);

void resetAlarm();
```

alarme.c

```c
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include "alarme.h"

static int alarmCounter = 0;

static int alarmFlag = 0;



void sigalarm_handler(int signo){


    alarmFlag=1;
    alarmCounter++;
    printf("Alarm ringing\n");


}

int getAlarmFlag(){
  return alarmFlag;
}

int getAlarmCounter(){
  return alarmCounter;
}

void setAlarmFlag(int flag){
  alarmFlag=flag;
}

void setAlarmCounter(int counter){
  alarmCounter=counter;
}

void setAlarm(){
  struct sigaction act_alarm;
  act_alarm.sa_handler = sigalarm_handler;
  sigemptyset(&act_alarm.sa_mask);
  act_alarm.sa_flags = 0;

  if (sigaction(SIGALRM,&act_alarm,NULL) < 0)  {
      fprintf(stderr,"Unable to install SIGALARM handler\n");
      exit(1);
  }
}


void resetAlarm(){
  alarm(0);
  alarmFlag=0;
  alarmCounter=0;
}
```

C application.c > llwrite(int, unsigned char *, int)

```c
static int currFrame=0;

struct termios oldtio;



int llopen(char *port, int type){
    struct termios newtio;

    // 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 = open(port, O_RDWR | O_NOCTTY );
    if (fd <0) {
        perror(port);
        exit(-1);
    }


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

    setAlarm();

    if (type == RECEIVER){
        openReader(fd);
    }
    else if(type == TRANSMITTER){
        openWriter(fd);
    }

    return fd;
}
```

```c
int llclose(int fd, int type){


    if (type == RECEIVER){
        closeReader(fd);
    }
    else if(type == TRANSMITTER){
        closeWriter(fd);
    }

    if (tcsetattr(fd,TCSANOW,&oldtio) == -1) {
        perror("tcsetattr");
        exit(-1);
    }

    return 0;
}
```

```c
int openReader( int fd){



    if(receiveSupervisionFrame(fd,SET) >= 0){
        sendSupervisionFrame(fd,UA);
    }
    else{
        printf("Did not UA. exited program\n ");
        llclose(fd, RECEIVER);
        exit(-1);
    }

    //printf("Reader received SET frame and sent UA successfully\n");
    return 0;
}
```

application.c > llwrite(int, unsigned char *, int)

```c
int receiverApp(int fd){
    struct timespec start,end;
    clock_gettime(CLOCK_MONOTONIC_RAW, &start);

    char buff[2*MAX_SIZE+7];
    int size;

    int state = 0;

    unsigned char* fileName;

    unsigned long fileSize=0;
    //START Control Frame
    while (!state) {
        memset(buff, 0, sizeof(buff));
        while ((size = llread(fd, buff)) < 0) {
            printf("Error reading\n");
            llclose(fd, RECEIVER);
            return -1;
        }
        controlFrame frame= parseControlFrame(buff,size);

        for(int i=0; i<frame.filesizeSize;i++)
            fileSize|=frame.fileSize[i]<<(8*i);

        printf("--------FILE SIZE-------%d\n",fileSize);
        fileName = frame.fileName;

        //printf("File Name: %s\n", fileName);

        //printControlFrame(frame);
        if (frame.control == START_FRAME)
            state = 1;

        free(frame.fileSize);
        free(frame.fileName);

    }


    // * DATA Frames
    unsigned char *fullMessage = (unsigned char*) malloc (fileSize*sizeof(unsigned char));
    //unsigned char fullMessage[fileSize];
    int index = 0;
    int currSequence = -1;
    int totalSize=0;
    while (state == 1) {
        memset(buff, 0, sizeof(buff));
        //printf("done memset\n");
        printf("Sequence: %d\n",currSequence);
        while ((size = llread(fd, buff)) <0) {
            memset(buff, 0, sizeof(buff));
            printf("Error reading\n");
        }

        if (buff[0] == END_FRAME) {
            //printf("Received End Frame\n");
            state = 2;
            break;
        }
        dataFrame data = parseDataFrame(buff, size);
        totalSize+=data.dataSize;
        if (data.control != DATA) {
            continue;
        }

        //printDataFrame(data);
        for (int i =0;i<data.dataSize;i++){
            fullMessage[index+i] = data.data[i];
        }
        // * caso o numero de sequencia seja diferente do anterior deve atualizar o index
        if (currSequence != data.sequence) {
            currSequence = data.sequence;
            index += data.dataSize;
        }
    }
```

```
193     if (state == 2) {
194         controlFrame frame = parseControlFrame(buff, size);
195         //printControlFrame(frame);
196 
197         //char* name = (char*) malloc ((strlen(fileName) +7) * sizeof(char));
198         char* name = (char*) malloc ((frame.filenameSize +7) * sizeof(char));
199         //char name[frame.filenameSize +7];
200 
201         sprintf(name, "cloned_%s", frame.fileName);
202 
203         FILE *fl = fopen(name, "wb");
204         if (fl != NULL) {
205             fwrite(fullMessage, sizeof (unsigned char), fileSize, fl);
206             fclose(fl);
207         }
208         //printf("Received file\n");
209         free(frame.fileSize);
210         //printf("Freed file size\n");
211         free(frame.fileName);
212         //printf("Freed file name\n");
213         free(name);
214         //printf("Freed name\n");
215         //free(fl);
216     }
```

application.c > llwrite(int, unsigned char *, int)

```
220     free(fullMessage);
221     //printf("Freed everything\n");
222 
223     clock_gettime(CLOCK_MONOTONIC_RAW, &end);
224     long seconds = end.tv_sec - start.tv_sec;
225     long nanoseconds = end.tv_nsec - start.tv_nsec;
226     double elapsed = seconds + nanoseconds*1e-9;
227     printf("Time measured: %f seconds.\n", elapsed);
228 
229 
230     return 0;
231 }
232 
233 
234 int llread(int fd, char* buffer){
235     unsigned char bcc2=0xff;
236     infoFrame frame = messageDestuffing(buffer,fd,&bcc2);
237     //printInfoFrame(frame);
238     //printf("Exited destuffing\n");
239 
240 
241     //printf("bcc1: %x - %x address ^ control\n",frame.bcc1,frame.address ^frame.control);
242     if(frame.bcc1!=(frame.address ^frame.control) || frame.bcc2 !=bcc2){
243         sendSupervisionFrame(fd, CONTROL_RJ(currFrame));
244         for (int i= 0; i<frame.rawSize;i++){
245             if(i%10==0)
246                 printf("\n");
247             printf("RD%d:%02x ",i,frame.rawData[i]);
248         }
249         printf("Sent Negative Response\n");
250         free(frame.rawData);
251         free(frame.data);
252         return -1;
253     }
254     else{
255         memcpy(buffer, frame.data,frame.size);
256         sendSupervisionFrame(fd, CONTROL_RR(currFrame));
257         //printf("Sent Positive Response\n");
258         if(currFrame)
259             currFrame--;
260         else
261             currFrame++;
262 
263         free(frame.rawData);
264         free(frame.data);
265         return frame.size;
266     }
267 }
268 
269 
```

C application.c > llwrite(int, unsigned char *, int)

```c
infoFrame messageDestuffing(unsigned char* buff,int fd ,unsigned char *bcc2){

    infoFrame frame;
    frame.rawData=(unsigned char*) malloc (sizeof(unsigned char)*(MAX_SIZE*2+7));
    frame.data = (unsigned char*) malloc(sizeof(unsigned char)*(MAX_SIZE*2));
    int flags = 0;
    unsigned char msg;
    int size=0;
    int lastEsc=0;
    while(flags!=2){
        //printf("read --");
        read(fd, &msg, 1);


        if (msg == FLAG && flags == 0) {
            flags = 1;
            //printf("Flag 1\n");
            continue;
        }
        else if (msg == FLAG && flags == 1) {
            flags = 2;
            //printf("Flag 2 - size %d\n",size);
            if (size>1 && frame.rawData[size-2] == ESC) {
                if (frame.rawData[size-1] == ESC_FLAG)
                    frame.rawData[--size-1]=FLAG;
                else if (frame.rawData[size-1] == ESC_ESC)
                  size--;
            }
            break;
        }
        //printf("msg- 0x%02x size- %d rawData- 0x%02x",msg,size,frame.rawData);
        frame.rawData[size] = msg;

        //printf(" --r");
        if (size>0 && frame.rawData[size-1] == ESC && !lastEsc) {
            if (frame.rawData[size] == ESC_FLAG){
                frame.rawData[--size]=FLAG;
                lastEsc=1;
            }
            else if (frame.rawData[size] == ESC_ESC){
                  size--;
                  lastEsc=1;
              }

        }
        else
         lastEsc=0;

        /*if(size>3){
            frame.data[size-3] = frame.rawData[size];
            //*bcc2^=frame.data[size-4];
        }
        else if(size==3)
            frame.data[size-3] = frame.rawData[size];*/

        size++;
        //printf("--new size-%d\n",size);
    }
```

```
328         frame.rawData = (unsigned char*) realloc (frame.rawData, (size));
329         frame.data = (unsigned char*) realloc (frame.data, (size-4));
330         if(PROBABILITY_BCC1)
331             generateErrorBCC1(frame.rawData);
332         if(PROBABILITY_BCC2)
333         generateErrorBCC2(frame.rawData, size);
334         //frame.rawData = (unsigned char*) realloc (frame.rawData, (size));
335         //printf("exits\n");
336         frame.flag=FLAG;
337         frame.address = frame.rawData[0];
338         frame.control = frame.rawData[1];
339         frame.bcc1 = frame.rawData[2];
340 
341         //printf("------ RAW SIZE------%d\n",size);
342         for (int i = 0; i < size - 4; i++) {
343             frame.data[i] = frame.rawData[i+3];
344             *bcc2^=frame.data[i];
345         }
346 
347 
348         frame.bcc2=frame.rawData[size-1];
349         frame.size=size-4;
350         frame.rawSize=size;
351         return frame;
352 
353     }
354 
355 
```

```
356     int closeReader(int fd){
357 
358         printf("Closing reader...\n");
359 
360 
361 
362         do{
363             alarm(ALARM_TIME);
364             setAlarmFlag(0);
365             int resp;
366             while((resp=receiveSupervisionFrame(fd, DISC)) <-0){
367                 printf("Error recieving DISC Frame...\n");
368             }
369 
370 
371             if(sendSupervisionFrame(fd,DISC)!=5){
372                 printf("Error Sending DISC\n");
373                 continue;
374             }
375 
376             printf("%d\n",getAlarmCounter());
377 
378             if(receiveSupervisionFrame(fd, UA)==TRUE){
379                 resetAlarm();
380                 printf("\nClosed Reader\n");
381                 return 0;
382             }
383 
384             if(getAlarmFlag()){
385                 printf("Timed Out\n");
386             }
387         }while(getAlarmCounter()<3);
388         resetAlarm();
389 
390 
391         printf("Error recieving UA Frame...\n");
392         return -1;
393 
394 
395 
396     }
397 
```

```
405  int openWriter(int fd){
406
407
408
409
410      do{
411          sendSupervisionFrame(fd,SET);
412          alarm(ALARM_TIME);
413          setAlarmFlag(0);
414          //printf("%d\n",getAlarmCounter());
415
416          if(receiveSupervisionFrame(fd, UA)>=0){
417              resetAlarm();
418              return 0;
419          }
420
421          if(getAlarmFlag()){
422              printf("Timed Out\n");
423          }
424      }while(getAlarmCounter()<3);
425      resetAlarm();
426
427      perror("Error retriving supervision frame\n");
428
429
430      llclose(fd,TRANSMITTER);
431
432
433      exit(-1);
434  }
435
436
437  int transmitterApp(char *path, int fd){
438      struct timespec start,end;
439      clock_gettime(CLOCK_MONOTONIC_RAW, &start);
440
441
442      int fileFd;
443      struct stat fileStat;
444
445      //printf("Before lstat\n");
446
447      if (lstat(path, &fileStat)<0){
448          perror("Error getting file information.\n");
449          return -1;
450      }
451
452      //printf("lstat successful\n");
453
454      if ((fileFd = open(path, O_RDONLY)) < 0){
455          perror("Error opening file.\n");
456          return -1;
457      }
458
```

```c
int transmitterApp(char *path, int fd){
    struct timespec start,end;
    clock_gettime(CLOCK_MONOTONIC_RAW, &start);


    int fileFd;
    struct stat fileStat;

    //printf("Before lstat\n");

    if (lstat(path, &fileStat)<0){
        perror("Error getting file information.\n");
        return -1;
    }

    //printf("lstat successful\n");

    if ((fileFd = open(path, O_RDONLY)) < 0){
        perror("Error opening file.\n");
        return -1;
    }

   // printf("gif opened successfully\n");

    //Generates and sends START control frame
    unsigned int L1 = sizeof(fileStat.st_size);  //Size of file

    //printf("L1\n");

    unsigned int L2 = strlen(path);  //Length of file name
    //printf("L2\n");
    unsigned int frameSize = 5 + L1 + L2;

    unsigned char *controlFrame = buildControlFrame(START_FRAME, fileStat.st_size, path, L1, L2, frameSize);

    //printf("built control frame\n");
    int resp;
    while((resp=llwrite(fd, controlFrame, frameSize))==-1){
        usleep(STOP_AND_WAIT);
    }
    if(resp==-2){
        perror("Error sending START frame.\n");
        free(controlFrame);
        return -1;
    }

    //printf("wrote start frame sucessfully\n");

    usleep(STOP_AND_WAIT);
    //Generates and sends data packets
    char *buf=(char*)malloc(sizeof(char)*MAX_SIZE);
    unsigned int bytesToSend, noBytes;
    unsigned int sequenceNumber = 0;
```

```c
    while((noBytes = read(fileFd, buf, MAX_SIZE-4))){
        bytesToSend = noBytes + 4;
        unsigned char *data=(unsigned char *)malloc(sizeof(unsigned char)*bytesToSend);
        //printf(" ------Data size------ %d\n",noBytes);
        data[0] = DATA;
        data[1] = sequenceNumber % 255;
        data[2] = noBytes /256;
        data[3] = noBytes % 256;
        memcpy(&data[4], buf, noBytes);
        while((resp=llwrite(fd, data, bytesToSend))==-1){
            usleep(STOP_AND_WAIT);
        }
        if(resp==-2){
            perror("Error sending Data frames\n");
            free(data);
            free(controlFrame);
            return -1;
        }
        free(data);
        sequenceNumber++;
        usleep(STOP_AND_WAIT);
    }
    free(buf);
    printf("Number of data frames sent: %d\n", sequenceNumber);

    usleep(STOP_AND_WAIT);
    //Generates and sends END control frame
    unsigned char *endControlFrame = buildControlFrame(END_FRAME, fileStat.st_size, path, L1, L2, frameSize);
    //printf("frame size- %d\n",frameSize);

    while((resp=llwrite(fd, endControlFrame, frameSize))==-1){
        usleep(STOP_AND_WAIT);
    }
    if(resp==-2){
        perror("Error sending END frame.\n");
        free(controlFrame);
        free(endControlFrame);
        return -1;
    }


    free(controlFrame);
    free(endControlFrame);


    clock_gettime(CLOCK_MONOTONIC_RAW, &end);
    long seconds = end.tv_sec - start.tv_sec;
    long nanoseconds = end.tv_nsec - start.tv_nsec;
    double elapsed = seconds + nanoseconds*1e-9;
    printf("Time measured: %f seconds.\n", elapsed);
    return 0;
}
```

```c
void generateErrorBCC2(unsigned char *frame, int frameSize){
  int prob = (rand() % 100) + 1;
    printf("bcc2 - prob: %d", prob);
  if (prob <= PROBABILITY_BCC2){
    int i = (rand() % (frameSize - 5)) + 4; /* only considering data and BCC2*/
    unsigned char randomAscii = (unsigned char)((rand() % 177));
    frame[i] = randomAscii;
    printf("\nGenerate BCC2 with errors.\n");
  }

}

void generateErrorBCC1(unsigned int *checkBuffer){
  int prob = (rand() % 100) + 1;
  printf("bcc1 - prob: %dss", prob);
  if (prob <= PROBABILITY_BCC1)
  {
    int i = (rand() % 2);
    unsigned char randomAscii = (unsigned char)((rand() % 177));
    checkBuffer[i] = randomAscii;
    printf("\nGenerate BCC1 with errors.");
  }

}

int llwrite(int fd, unsigned char* buffer,int length){

    //Init
    printf("\nMessage: %x\n", buffer);

    infoFrame frame = messageStuffing(buffer, length);
    //printInfoFrame(frame);
    //printf("raw data %hhn\n",frame.rawData);
    int size;
    //printf("raw size %d\n",frame.rawSize);
    alarm(ALARM_TIME);
    //printf("Alarm counter %d\n",getAlarmCounter());
```

```c
    do{
        //usleep(100);
        if((size=write(fd, frame.rawData, frame.rawSize))>=0){
            printf("Message sent\n");
        }
        else{
            alarm(0);
            setAlarmFlag(0);
            printf("Message not sent\n");
            free(frame.rawData);
            free(frame.data);
            return -1;
        }

        if(getAlarmFlag()){
            alarm(ALARM_TIME);
            setAlarmFlag(0);
            //printf("Alarm counter %d\n",getAlarmCounter());
        }


        unsigned char response = readSupervisionFrame(fd);
        if(response==0xff)
            continue;

        if(response==CONTROL_RJ(1)||response==CONTROL_RJ(0)){
            alarm(0);
            setAlarmFlag(0);
            printf("Negative response\n");
            free(frame.rawData);
            free(frame.data);
            return -1;
        }
        else if (response==CONTROL_RR(currFrame)){
            resetAlarm();
            if(currFrame)
                currFrame--;
            else
                currFrame++;

            free(frame.rawData);
            free(frame.data);
            return size;
        }
    }while (getAlarmCounter()<RT_ATTEMPTS);
    resetAlarm();
    free(frame.rawData);
    free(frame.data);
    return -2;
}
```

application.c > llwrite(int, unsigned char *, int)

```
634     infoFrame messageStuffing(unsigned char* buff, int length){
635         infoFrame frame;
636         memset(&frame,0, sizeof( infoFrame));
637         frame.flag=FLAG;
638         frame.address=A;
639         frame.control=CONTROL_I(currFrame);
640         frame.bcc1=frame.address ^ frame.control;
641 
642         frame.bcc2=0xff;
643         int size=length;
644         frame.data=(unsigned char*)malloc((size+1)*sizeof(unsigned char));
645         frame.size=0;
646         for(int i =0; i<length;i++){
647             if(buff[i]==ESC){
648                 frame.data=(unsigned char*)realloc(frame.data, ++size);
649                 frame.data[frame.size++]=ESC;
650                 frame.data[frame.size++]=ESC_ESC;
651             }
652             else if(buff[i]==FLAG){
653                 frame.data=(unsigned char*)realloc(frame.data, ++size);
654                 //printf("got Flag\n");
655                 frame.data[frame.size++]=ESC;
656                 frame.data[frame.size++]=ESC_FLAG;
657             }
658             else
659                 frame.data[frame.size++]=buff[i];
660 
661             frame.bcc2=buff[i]^frame.bcc2;
662         }
663         frame.rawData=(unsigned char*)malloc((frame.size+7)*sizeof(unsigned char));
664         frame.rawData[0]=frame.flag;
665         frame.rawData[1]=frame.address;
666         frame.rawData[2]=frame.control;
667         frame.rawData[3]=frame.bcc1;
668 
669         for (int i=0; i< frame.size;i++){
670             frame.rawData[i+4]=frame.data[i];
671             //printf("%x\n",frame.data[i]);
672         }
673 
674         if(frame.bcc2==ESC){
675             frame.rawData[frame.size+4]=frame.bcc2;
676             frame.rawData[frame.size+5]=ESC_ESC;
677             frame.rawData[frame.size+6]=frame.flag;
678             frame.rawSize=frame.size+7;
679         }
680         else if (frame.bcc2==FLAG){
681             frame.rawData[frame.size+4]=ESC;
682             frame.rawData[frame.size+5]=ESC_FLAG;
683             frame.rawData[frame.size+6]=frame.flag;
684             frame.rawSize=frame.size+7;
685         }
686         else{
687             frame.rawData[frame.size+4]=frame.bcc2;
688             frame.rawData[frame.size+5]=frame.flag;
689             frame.rawSize=frame.size+6;
690         }
691 >       //printf("----Frame Size -----%d\n",frame.size);···
695 
696         return frame;
697     }
698 
```

```c
int closeWriter(int fd){

    //printf("Closing writer...\n");


    do{
        sendSupervisionFrame(fd,DISC);
        //printf("sent DISC frame\n")
        alarm(ALARM_TIME);
        setAlarmFlag(0);
        //printf("%d\n",getAlarmCounter());

        if(receiveSupervisionFrame(fd, DISC)){
            sendSupervisionFrame(fd,UA);
            alarm(0);
            setAlarmCounter(0);
            printf("\nClosed Writer\n");
            return 0;
        }

        if(getAlarmFlag()){
            printf("Timed Out\n");
        }
    }while(getAlarmCounter()<3);
    setAlarmCounter(0);



    printf("Error recieving DISC Frame...\n");

    return -1;
}
```

interface.c > ...

```c
#include "interface.h"


int main(int argc, char **argv){
    //set_alarm();
    applicationLayer app;

    //parses arguments
    if (argc != 5){
        printf("Usage: ./application -p <port> -r/-w <file_path>\n");
    }
    else{
        for (int i = 0; i < argc; i++){
            switch(i){
                case 1:
                    if (strcmp(argv[1], "-p")!=0){
                        printf("Usage: ./application -p <port> -r/-w <file_path>\n");
                        printf("-p tag is missing\n");
                        return -1;
                    }
                    break;
                case 2:
                    if ((strcmp("/dev/ttyS0", argv[2])!=0) && (strcmp("/dev/ttyS1", argv[2])!=0) &&
                        (strcmp("/dev/ttyS10", argv[2])!=0) && (strcmp("/dev/ttyS11", argv[2])!=0) ){
                        printf("Usage: ./application -p <port> -r/-w <file_path>\n");
                        printf("port is missing\n");
                        return -1;
                    }
                    else{
                        app.port = argv[2];
                    }
                    break;
                case 3:
                    if (strcmp(argv[3], "-r")!=0 && strcmp(argv[3], "-w")!=0){
                        printf("Usage: ./application -p <port> -r/-w <file_path>\n");
                        printf("tag -r/-w is missing\n");
                        return -1;
                    }
                    else{
                        if (strcmp(argv[3], "-r") == 0){
                            app.status = RECEIVER;
                            app.path = argv[4];
                        }
                        if (strcmp(argv[3], "-w") == 0){
                            app.status = TRANSMITTER;
                            app.path = argv[4];
                        }
                    }
                    break;
            }
        }
    }
```

C interface.c > ...

```c
        }
    }

    //clock_t start = clock();

    if((app.fileDescriptor=llopen(app.port, app.status)) < 0){
        printf("Error opening file descriptor\n");
        exit(1);
    }


    //clock_t mid1 = clock();

    if(app.status== TRANSMITTER){
        transmitterApp(app.path,app.fileDescriptor);

    }

    else if(app.status==RECEIVER){
        if(receiverApp(app.fileDescriptor)<0){
            perror("Error on receiver\n");
            exit(1);
        }

    }

    //clock_t mid2 = clock();
    //printf("About to close\n");

    if(llclose(app.fileDescriptor,app.status)){
        printf("Error closing file descriptor");
        exit(1);

    }

    //clock_t end = clock();
    //double cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    //printf("START: %f\n", (double)start / CLOCKS_PER_SEC);
    //printf("MID1: %f\n", (double)mid1 / CLOCKS_PER_SEC);
    //printf("MID2: %f\n", (double)mid2 / CLOCKS_PER_SEC);
    //printf("END: %f\n", (double)end / CLOCKS_PER_SEC);
    //printf("CPU TIME USED: %f\n", cpu_time_used);


    return 0;
}
```

C interface.h > ...

```c
#include "macros.h"
#include "application.h"

typedef struct{
    char *port;  /*Path da porta de série passado pelo user*/
    int fileDescriptor;/*Descritor correspondente à porta série*/
    int status;/*TRANSMITTER | RECEIVER*/
    char *path;
}applicationLayer;
```

```c
#include "parseNbuild.h"



unsigned char *buildControlFrame(char ctrlField, unsigned long fileSize, char* fileName, unsigned int L1, unsigned int L2, unsigned int frameSize) {
    unsigned char *frame=(unsigned char*) malloc(sizeof(unsigned char)*frameSize);
    //printf(" ----Frame Size -------%d\n",frameSize);
    frame[0] = ctrlField;
    frame[1] = FILE_SIZE;
    frame[2] = L1;
    memcpy(&frame[3], &fileSize, L1);
    /*for(int i=0; i<L1;i++)
        printf("j = 0x%02x\n",frame[3+i]);*/
    frame[3+L1] = FILE_NAME;
    frame[4+L1] = L2;
    memcpy(&frame[5+L1], fileName, L2);
    //printf(" -----File Size -------%d\n",fileSize);
    //printf(" -----Size Size -------%d\n",L1);
    //printf(" -----File Name -------%s\n",fileName);
    //printf(" -----Name Size -------%d\n",L2);

    return frame;
}



dataFrame parseDataFrame(unsigned char *rawBytes, int size) {
    dataFrame frame;
    memset(&frame, 0, sizeof( dataFrame));
    frame.rawBytes = rawBytes;
    frame.rawSize = size;
    frame.control = rawBytes[0];
    frame.sequence = rawBytes[1];

    frame.dataSize = (rawBytes[2] << 8) | rawBytes[3];
    for (int i = 0; i < frame.dataSize; i++) {
        frame.data[i] = rawBytes[i+4];
    }

    return frame;
}
```

```c
controlFrame parseControlFrame(unsigned char *rawBytes, int size) {
    controlFrame frame;
    memset(&frame, 0, sizeof(controlFrame));
    frame.control = rawBytes[0];

    frame.filenameSize = 0;

    frame.filesizeSize = 0;
    int len;
    int fileSizeFlag=1;
    for (int i = 1; i < size;i++) {
        if (rawBytes[i] == FILE_SIZE && fileSizeFlag) {
            fileSizeFlag=0;
            // printf("Parsing file size\n");
        len = rawBytes[++i];
        //printf("len %d\n",len);
        frame.fileSize = (unsigned char*) malloc(len*sizeof(unsigned char));

        for (int j = 0; j <len;j++) {
            frame.fileSize[j] = rawBytes[++i];
            //printf("j %d - byte 0x%02x\n",j,frame.fileSize[j]);

        }
        frame.filesizeSize=len;

        }
        else if (rawBytes[i] == FILE_NAME) {
            //printf("Parsing file name\n ");
        len = rawBytes[++i];
        //printf("len %d\n",len);
        frame.fileName = (unsigned char *) malloc ((len+1)*sizeof(unsigned char));

        for (int j = 0; j < len;j++) {
            frame.fileName[j] = rawBytes[++i];
            //printf("j %d - byte 0x%02x\n",j,frame.fileName[j]);
        }
        frame.filenameSize=len;
        }

    }
    /*for (int i=1;i<size;i++)
        printf("RawData byte %d- 0x%02x\n",i,rawBytes[i]);*/

    frame.rawSize=size;
    frame.fileName[frame.filenameSize] = '\0';

    frame.rawBytes=rawBytes;



    return frame;
}
```

parseNbuild.h > parseDataFrame(unsigned char *, int)

```c
#pragma once
#include "macros.h"
#include "frameStructs.h"


unsigned char *buildControlFrame(char ctrlField, unsigned long fileSize, char* fileName, unsigned int L1, unsigned int L2, unsigned int frameSize);

controlFrame parseControlFrame(unsigned char *rawBytes, int size);

dataFrame parseDataFrame(unsigned char *rawBytes, int size);
```

printers.h > ...

```c
#pragma once
#include "frameStructs.h"
#include "macros.h"


void printInfoFrame( infoFrame frame);

void printDataFrame( dataFrame frame);

void printControlFrame( controlFrame frame);
```

printers.c > ...

```c
#include "printers.h"



void printDataFrame( dataFrame frame) {
  printf("\n     DATA     \n");
  printf("Control: -0x%02x\n", frame.control);
  printf("Data size: %d -0x%02x\n", frame.dataSize,
         frame.dataSize);
  printf("Sequence: %d -0x%02x\n", frame.sequence, frame.sequence);

  if (PRINT_ALL) {
    for (int i = 0; i < frame.dataSize; i++) {
      printf("DATA[%d]: %x -0x%02x\n", i, frame.data[i], frame.data[i]);
    }
  }
}


void printControlFrame( controlFrame frame){
    printf("\n       CONTROL      \n");
    printf("Control: %x\n", frame.control);
    printf("File size: %x\n", frame.fileSize);
    printf("File name: %s\n", frame.fileName);
    printf("Filesize size: %d\n", frame.filesizeSize);
    printf("Filename size: %d\n", frame.filenameSize);
    printf("Raw bytes: %x\n", frame.rawBytes);
    printf("Raw size: %d\n", frame.rawSize);

}

void printInfoFrame( infoFrame frame){
    printf("\n     Info     \n");
    printf("FLAG- 0x%x\n",frame.flag);
    printf("Address - 0x%x\n",frame.address);
    printf("Control - 0x%x\n",frame.control);
    printf("BCC1 - 0x%x",frame.bcc1);
    if(PRINT_ALL){
        for(int i =0;i<frame.size;i++){
            if(i%8==0){
                printf("\nData - ");
            }
            printf(" 0x%x,",frame.data[i]);
        }
    }
    printf("\nData Size - %d",frame.size);
    printf("\nBCC2 - 0x%x\n",frame.bcc2);
    printf("Second FLAG- 0x%x",frame.flag);
    if(PRINT_ALL){
        for(int i =0;i<frame.rawSize;i++){
            if(i%15==0){
                printf("\nRaw Data - ");
            }
            printf(" 0x%x,",frame.rawData[i]);
        }
    }
    printf("\nRaw Size - %d\n",frame.rawSize);
}
```

C supervision.c > readSupervisionFrame(int)

```c
#include "supervision.h"



int receiveSupervisionFrame(int fd, unsigned char control) {
  int times=0;
  unsigned char msg;
  printf("Reading response...\n");
  while (times!=5 && !getAlarmFlag()) {
    if(read(fd,&msg,1)<0)
      return -1;
    if(times==0){
        if(msg==FLAG){
          times++;
          //printf("FLAG- 0x%02x\n",msg);
        }
    }
    else if(times==1){
        if(msg==A){
          times++;
          //printf("A- 0x%02x\n",msg);
        }
        else {
          if(msg==FLAG)
            times--;
          else
            times=0;
        }
    }
    else if (times==2){
        if(msg==control){
          times++;
          //printf("C- 0x%02x\n",msg);
          control = msg;
        }
        else
          times=0;
    }
    else if(times==3){
        if(msg==(A^control)){
          times++;
          //printf("BCC- 0x%02x\n",msg);
        }
        else
          times=0;
    }
    else if(times==4){
        if(msg==FLAG) {
          times++;
          //printf("SECOND FLAG- 0x%02x\n",msg);
          return TRUE;
        }
        else
          times=0;
    }
  }
  return FALSE;
}

```

```c
unsigned char readSupervisionFrame(int fd){
  int times=0;
  unsigned char msg, control;
  //printf("Reading response...\n");
  while (times!=5) {
    int resp =read(fd,&msg,1);
    if (resp<0){
      return 0xff;
    }
    if(times==0){
        if(msg==FLAG){
          times++;
          //printf("FLAG- 0x%02x\n",msg);
        }
    }
    else if(times==1){
        if(msg==A){
          times++;
          //printf("A- 0x%02x\n",msg);
        }
        else {
          if(msg==FLAG)
            times--;
          else
            times=0;
        }
    }
    else if (times==2){
        if(msg==CONTROL_RJ(1) || msg==CONTROL_RJ(0)
        || msg==CONTROL_RR(1) || msg==CONTROL_RR(0)){
          times++;
          //printf("C- 0x%02x\n",msg);
          control = msg;
        }
        else
          times=0;
    }
    else if(times==3){
        if(msg==(A^control)){
          times++;
          //printf("BCC- 0x%02x\n",msg);
        }
        else
          times=0;
    }
    else if(times==4){
        if(msg==FLAG) {
          times++;
          //printf("SECOND FLAG- 0x%02x\n",msg);
        }
        else
          times=0;
    }
  }
  return control;
}


int sendSupervisionFrame(int fd, unsigned char msg) {
  unsigned char mesh[5];
  mesh[0]=FLAG;
  mesh[1]=A;
  mesh[2]=msg;
  mesh[3]=mesh[1]^mesh[2];
  mesh[4]=FLAG;
  int size=write(fd,mesh,5);
  printf("Sending Supervision Frame - 0x%02x\n",msg);
  return size;
}

```

frameStructs.h > ...

```c
#pragma once
#include "macros.h"



typedef struct {
    unsigned char flag;
    unsigned char control;
    unsigned char address;
    unsigned char bcc1;
    unsigned char bcc2;
    unsigned char * data; //after stuffing
    unsigned char * rawData;  //before stuffing
    int size;
    int rawSize;
}infoFrame;

typedef struct {
  unsigned char control;
  unsigned char *fileSize;
  unsigned char *fileName;
  int filesizeSize;
  int filenameSize;

  unsigned char *rawBytes;
  int rawSize;
}controlFrame;


typedef struct {
  unsigned char control;    /**< @brief The control byte - [DATA] */
  unsigned char sequence;   /**< @brief The sequence byte - index on global data */
  int dataSize;      /**< @brief The size of the data array - [1..PACKET_SIZE] */
  unsigned char data[2*MAX_SIZE]; /**< @brief The data array */

  unsigned char *rawBytes; /**< @brief The array containing unprocessed bytes */
  int rawSize;       /**< @brief The size of the raw_bytes array */
  unsigned char bcc2;
}dataFrame;
```

```c
#pragma once

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

#include <time.h>


#define BAUDRATE B38400
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1
#define MAX_SIZE 512

#define FLAG 0x7E
#define A 0x03
#define SET 0x03
#define UA 0x07
#define DISC 0x0B
#define SET_BCC A ^ SET
#define UA_BCC A ^ UA
```

```c
#define ESC 0x7d
#define ESC_ESC 0x5d
#define ESC_FLAG 0x5e

#define CONTROL_I(r) ((r == 0) ? 0x00 : 0x40)
#define CONTROL_RR(r) ((r == 0) ? 0x05 : 0x85)
#define CONTROL_RJ(r) ((r == 0) ? 0x01 : 0x81)

#define STOP_AND_WAIT 50000
#define ALARM_TIME 3
#define RT_ATTEMPTS 3

#define DATA 0x1
#define START_FRAME 0x2
#define END_FRAME 0x3


#define FILE_SIZE 0x00
#define FILE_NAME 0x01

#define TRANSMITTER 1234
#define RECEIVER 4321

#define PROBABILITY_BCC1 50
#define PROBABILITY_BCC2 50


#define PRINT_ALL 0
```

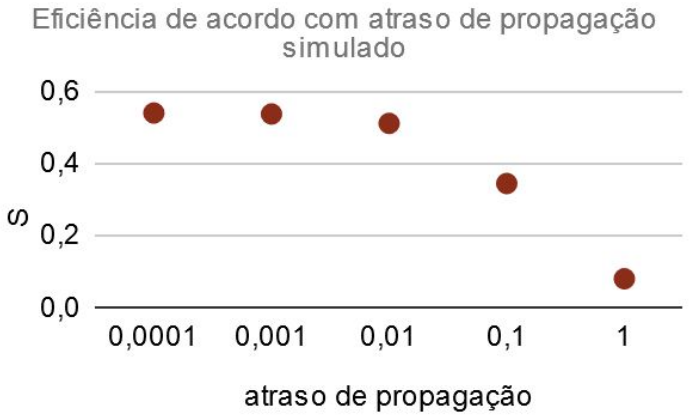
Anexo II - tabelas de cálculos auxiliares

Variação da eficiência relativamente ao tamanho da trama de dados
Baudrate = 38400
Número de bytes = 10960

Tamanho da trama	64	128	256	512	1024	2048
Tempos de transmissão (s)	13,291535	7,996446	5,464092	4,225035	3,606664	3,607207
	13,290487	7,995224	5,464867	4,226116	3,606728	3,606753
	13,291769	7,993265	5,464838	4,225693	3,607564	3,607295
Média dos tempos de transmissão (s)	13,291535	7,995224	5,464838	4,225693	3,606728	3,607207
R (bits/s)	6601,494861	10974,55181	16056,10267	20764,4048	24327,86725	24324,63676
S (R/C)	0,1719139287	0,28579562	0,4181276737	0,5407397083	0,6335382097	0,6334540823

Variação da eficiência relativamente ao tempo de propagação
Baudrate = 38400
Número de bytes = 10960
Tamanho da trama = 512

Atraso de propagação simulado	0,0001	0,001	0,01	0,1	1
Tempos de transmissão (s)	4,228927	4,250476	4,46651	6,626385	28,226656
	4,229571	4,250382	4,466739	6,626694	28,226537
	4,229732	4,250244	4,467188	6,626725	28,226645
Média dos tempos de transmissão (s)	4,22941	4,250367333	4,466812333	6,626601333	28,22661267
R (bits/s)	20746,15608	20643,86278	19643,53849	13241,17682	3108,555782
S (R/C)	0,5402644813	0,5376005933	0,5115504815	0,3448223131	0,08095197348



Variação da eficiência relativamente à capacidade de ligação (C)

Número de bytes = 10960

Tamanho da trama = 512

Baudrate	2400	4800	9600	19200	38400	57600
Tempos de transmissão (s)	49,475165	25,34182	13,274934	7,242727	4,225218	3,220044
	49,475711	25,34206	13,275285	7,241802	4,225113	3,219356
	49,475679	25,341461	13,27528	7,241567	4,22548	3,21962
Média dos tempos de transmissão (s)	49,4755183	25,3417803	13,2751663	7,242032	4,225270333	3,219673333
R (bits/s)	1773,483188	3462,424457	6609,634697	12115,93652	20766,48192	27252,45418
S (R/C)	0,738951328 5	0,721338428 5	0,688503614 2	0,631038360 5	0,540793800 1	0,473132885 1

Variação da eficiência relativamente ao FER
Baudrate = 38400
Número de bytes = 10960
Tamanho da trama = 512

Probabilidade de erro (em %)	10	25	50
Tempos de transmissão (s)	4,414674	8,329147	15,949221
	4,413774	8,329114	15,950955
	4,413622	8,327714	15,948779
Média dos tempos de transmissão (s)	4,414023333	8,328658333	15,94965167
R (bits/s)	19878,46311	10535,19024	5501,311366
S (R/C)	0,5176683102	0,2743539125	0,1432633168