**MIEIC 3º Ano**

**Novembro 2019**

**Redes de Computadores**

**1º Trabalho Laboratorial – Ligação de Dados**

Prof. Manuel Pereira Ricardo

Prof. Rui Lopes Campos

Turma 7

João Praça - up201704748

Leonor Sousa - up201705377

Sílvia Rocha - up201704684

## Sumário

O 1º Projeto Laboratorial foi elaborado no contexto da unidade curricular Redes de Computadores e teve como tema ligação de dados. Deste modo o objetivo do projeto passava em elaborar um programa que simulava um protocolo de ligação de dados e um protocolo de uma aplicação que tinham como objetivo a transferência de dados entre dois computadores via porta de série.

## Introdução

Este projeto tinha como **objetivos**:

* Elaborar um protocolo de ligação de dados;
* Elaborar uma aplicação simples que permite a transferência de ficheiros, utilizando o protocolo de ligação de dados previamente elaborado.

O relatório adjacente ao projeto tem como principal objetivo esclarecer a arquitetura, estrutura e a aplicação dos protocolos utilizados no projeto, assim como fazer a validação e a análise da eficiência do código desenvolvido.

Seguidamente, apresentam-se, neste relatório, informações sobre a implementação, casos de uso e resultados deste projeto. Deste modo, o relatório encontra-se estruturada da seguinte forma:

* **Arquitetura e Estrutura do Código** – descrição da arquitetura utilizada e da estruturação do código (APIs, estruturas de dados utilizadas e funções), assim como a relação entre ambas.
* **Casos de Uso Principais** – identificação dos casos de uso mais importantes, assim como da sequência de chamada de funções relacionada com cada um.
* **Protocolos** - descrição dos principais aspetos funcionais e implementação respetiva dos seguintes protocolos:
  + Protocolo de Ligação de Dados
  + Protocolo de Aplicação
* **Validação e Eficiência do Protocolo de Ligação de Dados** – descrição dos testes utilizados e seus respetivos resultados, caracterização estatística da eficiência do protocolo e comparação da mesma com os resultados esperados.

## Arquitetura e Estrutura do Código

A arquitetura deste projeto consiste abrange 3 blocos funcionais:

**protocol** (protocol.h e protocol.c)

Bloco responsável por fazer a criação e processamento de tramas e por fazer a transmissão em si. É também neste bloco que são efetuados os mecanismos de stuffing e destuffing, assim como eventuais retransmissões e verificação da integridade das tramas.

A API deste bloco é constituída pelas seguintes principais funções:

|  |  |
| --- | --- |
| int open\_port(char\* porta, struct termios \*oldtio) | void close\_port(int fd, struct termios \*oldtio) |
| void send\_set(int fd); | void send\_ua\_rcv(int fd); |
| void send\_ua\_snd(int fd); | void send\_disc\_rcv(int fd); |
| void send\_disc\_snd(int fd); | void send\_resp(int fd, char c, char a); |
| int send\_msg(int fd,unsigned char\* msg, int length); | |
| int receive\_msg(int fd, unsigned char c, unsigned char a, bool data, unsigned char data\_buf[], bool data\_resp); | |
| void receive\_set(int fd); | void receive\_disc\_rcv(int fd); |
| void receive\_disc\_snd(int fd); | void receive\_ua\_rcv(int fd); |
| void receive\_ua\_snd(int fd); | bool receive\_data\_rsp(int fd); |
| int receive\_data(int fd, unsigned char data\_buf[]); | |

**application** (application.h e application.c)

Bloco responsável por fazer a divisão do ficheiro em diversas partes e por construir e processar os pacotes (de controlo e de dados).

A API deste bloco é constituída pelas seguintes principais funções:

|  |  |
| --- | --- |
| int llopen(unsigned char \*porta, bool transmitter); | int llclose(int fd); |
| int llwrite(int fd, unsigned char \* buffer, int length); | int llreadFile(int fd); |
| void llopen\_image(unsigned char \*path, int fd); | |

**nserial** (nserial.c)

Bloco responsável por fazer a interface entre o utilizador e o bloco application. É constituído pela função main e pode ser chamado, via consola, através da instrução:

.\nserial <portaDeSerie> <TRANSMITTER(1)|RECEIVER(0)> <ficheiro>

* <portaDeSerie> é uma string do tipo "/dev/ttySX", em que X é o número da porta de série
* O campo <TRANSMITTER(1)|RECEIVER(0)> deverá ser 1 se se tratar do transmissor e 0 se se tratar do receptor
* O campo <ficheiro> só deverá existir caso se trato do transmissor e representa o ficheiro que deve ser enviado.

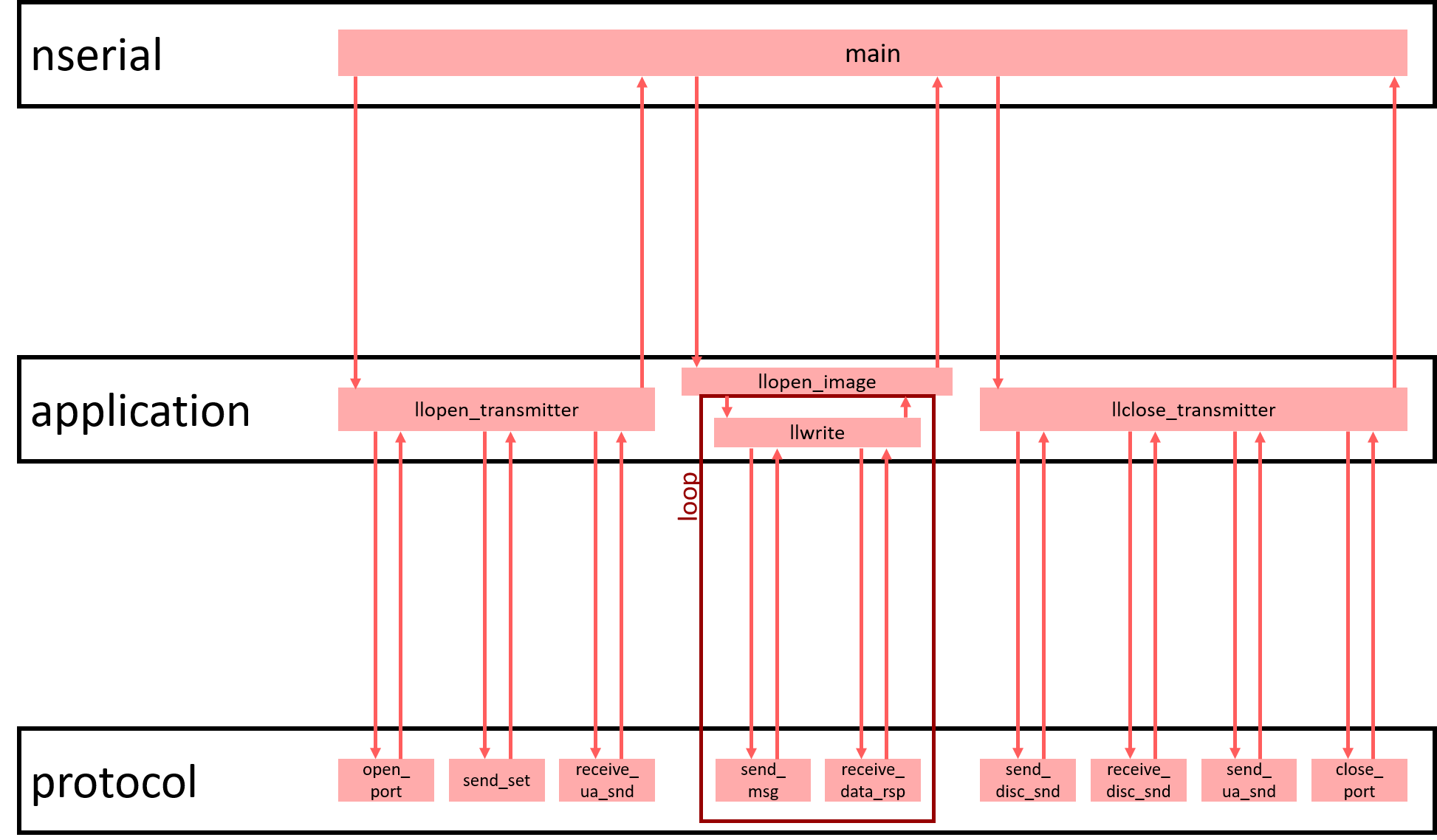
## Casos de Uso Principais

O principal caso de uso deste programa é a transmissão de ficheiros via porta de série. Desta forma, podemos subdividir este caso em dois subcasos:

**Envio de um Ficheiro**

O envio de um ficheiro pode ser feito, através da linha de comandos, usando a instrução: ./nserial <portaDeSerie> 1 <ficheiro>.

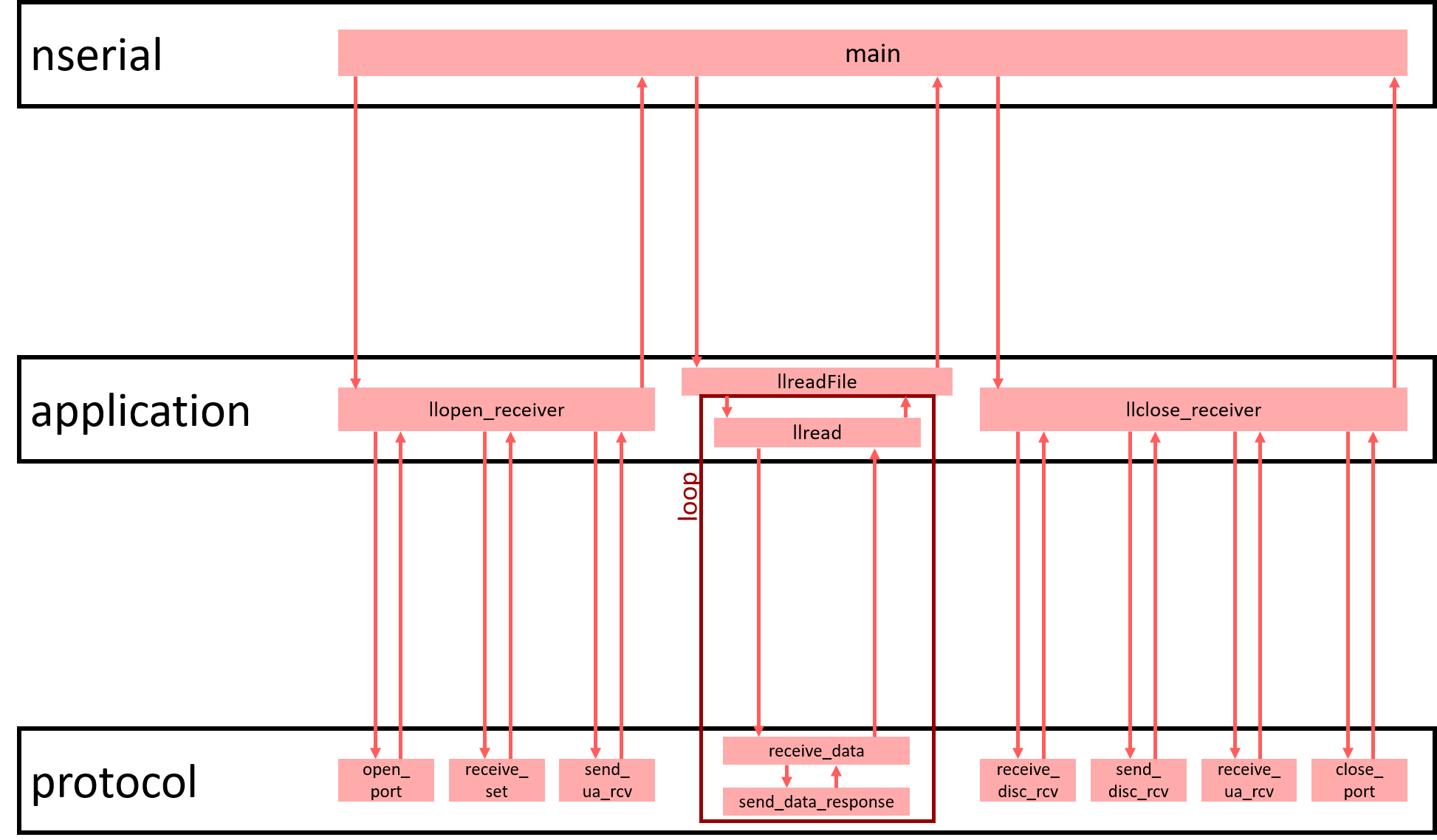
Esta instrução aciona a seguinte sequência de chamada de funções:



**Receção de um Ficheiro**

A receção de um ficheiro pode ser feito, através da linha de comandos, usando a instrução: .\nserial <portaDeSerie> 0.

Esta instrução aciona a seguinte sequência de chamada de funções:



## Protocolos

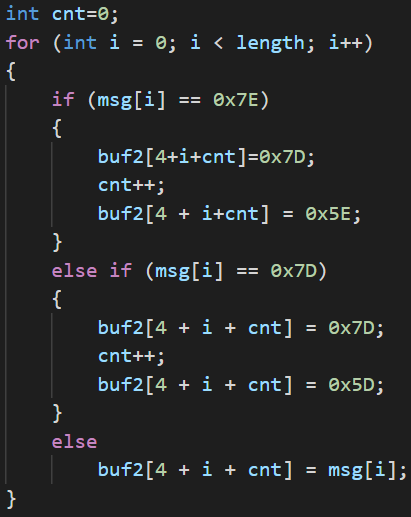
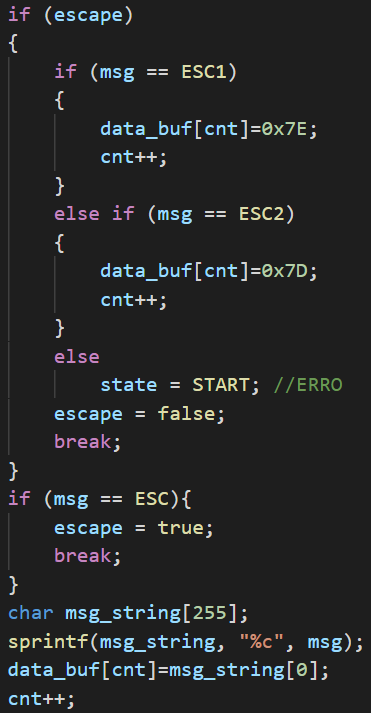
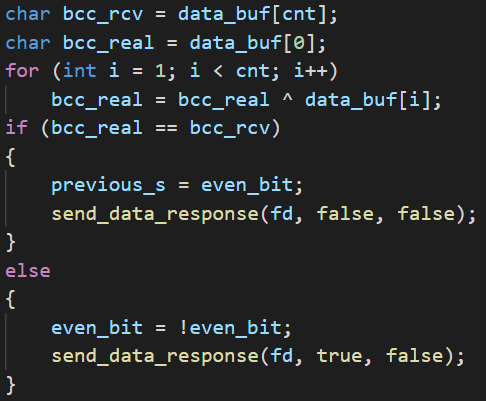
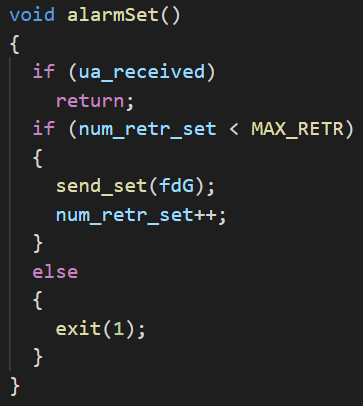
Protocolo de Ligação Lógica

Uma das principais funcionalidades do protocolo de ligação lógico é a criação e o processamento de tramas do tipo I e de tramas de dados. Estas tramas têm um formato já especificado à priori. O seu processamento é efetuado recorrendo à função receive\_msg(), onde é implementada uma máquina de estados. A sua criação é efetuada nas funções send\_msg() (tramas de dados) e send\_resp (tramas do tipo I).

A estas tramas é aplicado byte stuffing (no envio de dados) e byte destuffing (na receção de dados). Este mecanismo é aplicado recorrendo ao octeto escape (0x7D). Desta forma, sempre que se quer enviar o byte 0x7E, são enviados os bytes 0x7D e 0x5E e, sempre que se quer enviar o byte 0x7D, são enviados os bytes 0x7D e 0x5D. Na leitura faz-se o processamento inverso.

O protocolo de ligação de dados faz também a verificação da integridade das tramas, a partir da análise do BCC e do BCC2, no caso de tramas de dados.

No caso de ocorrer um timeout na receção da resposta ao envio da trama, a trama é reenviada. Este mecanismo é efetuado recorrendo a sinais do tipo ALARM.

1. Retransmissão
2. Destuffing
3. Stuffing
4. Verificação da Integridade

Protocolo de Aplicação

O protocolo de aplicação é responsável por fazer a construção e o processamento dos pacotes (de dados e de controlo), não tendo qualquer conhecimento da estrutura das tramas e de todos os outros pormenores que fazem parte do protocolo de ligação de lógica. Na construção e desconstrução dos pacotes deve ser efetuada uma numeração correta dos pacotes.

## Validação

Para garantir a validade do protocolo desenvolvido foram efetuados os seguintes testes:

* Envio de ficheiros de vários tamanhos e tipo.
  + Todos os ficheiros
* Interrupção da ligação múltiplas vezes durante o envio de um ficheiro.
  + Em cada uma das medições, do lado do *sender,* o alarme instalado para o envio de dados foi acionado e os dados foram reenviados.
* Geração de interferência durante o envio de um ficheiro.
  + Do lado do *receiver,* os dados incorretos foram descartados .
  + Do lado do *sender*, os dados foram reenviados.
* Variação na percentagem de erros simulados no envio de um ficheiro.
* Variação do tamanho de pacotes de dados no envio de um ficheiro.
* Variação da capacidade de ligação (*baudrate*) no envio de um ficheiro.

Todos estes testes foram bem sucedidos. Os três últimos testes serão abordados com maior pormenor na secção seguinte.

## Eficiência do protocolo de ligação de dados

Todas as medições de eficiência apresentadas nesta secção foram efetuadas usando como ficheiro de envio o que nos foi fornecido (pinguim.gif) e recorrendo a duas máquinas virtuais. Os valores obtidos em todas as medições e que originaram os gráficos que se seguem encontram-se na secção correspondente aos anexos.

### **Variação da Capacidade de Ligação (*baudrate*)**

Ao testar o comportamento do protocolo aquando da variação da capacidade de ligação, as alterações do tempo de execução detetadas foram muito reduzidas. Deste modo, recorrendo à fórmula em que S=R/C e R=<fileSize>/<tempo de execução>, as variações no valor de S são bastante acentuadas e os valores são, em geral, desadequados.

Apesar disto, conseguimos concluir que a eficiência do protocolo diminui com o aumento da *baudrate*.

### **Variação do tamanho dos Pacotes de Dados**

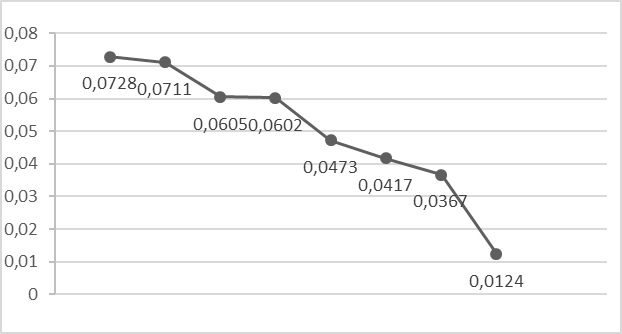
Ao testar o comportamento do protocolo aquando da variação do tamanho dos pacotes de dados, verifica-se que a eficiência do mesmo aumenta com o aumento desse tamanho.

### 

### **Variação do FER**

Ao testar o comportamento do protocolo aquando da variação da probabilidade de erros simulados e, consequentemente, o número de erros gerados, é possível concluir que os erros quer ao nível do cabeçalho quer ao nível de dados afetam a eficiência do protocolo. É ainda visível que os erros associados ao bcc1 têm um impacto maior que o bcc2. Este facto é facilmente explicável pelo facto de que aquando de um erro no bcc1, o *sender* irá recorrer ao alarm instalado pelo que será necessário atingir o TIMEOUT para que volte a enviar a mesma trama. Já no bcc2, aquando de um erro, as informações são simplesmente descartadas e a nova trama é enviada imediatamente.

No gráfico seguinte <num>+<num> representa o número de erros no bcc1 e bcc2, respetivamente.



0+0

0+12

0+14

0+17

0+26

0+46

9+0

4+56

Este protocolo recorre ao mecanismo *Stop & Wait* para controlo de erros. Este mecanismo, permite a retransmissão dos dados em caso de erro.

Após a transmissão de um *packet* de informação, o emissor espera por uma confirmação positiva por parte do recetor, denominada por *acknowledgment*, **ACK**. Quando o recetor recebe o *packet*, caso não tenha nenhum erro, confirma com **ACK**, caso tenha erro, envia **NACK**. Assim que o emissor recebe a resposta do recetor, no caso de **ACK**, continua e envia um novo *packet*, mas no caso de **NACK**, volta a enviar o mesmo *packet*.

Na nossa aplicação, quando o emissor manda qualquer tipo de tramas (U, S ou I) espera uma resposta. Essa resposta é **RR** caso o recetor receba os dados sem erros, e **REJ** caso contrário. Assim, o emissor sabe se deve mandar uma nova trama ou reenviar a mesma. Para que seja possível identificar se uma trama é nova ou duplicada, varia-se o *Nr* destas tramas de resposta de acordo com a trama de *Ns* 0 ou 1 enviada pelo emissor.

## Conclusões

Consideramos que os objetivos principais do projeto foram alcançados com sucesso: o programa lê corretamente um ficheiro, divide-o em pacotes numerados, cria tramas, envia-as através da porta de série (transmissão série assíncrona), recebe-as, escreve o ficheiro no computador recetor, e deteta e corrige eventuais erros nas tramas transmitidas. O código está organizado em camadas independentes (nserial, application e protocol), de forma a que haja abstração.

Para além disso, este projeto ajudou-nos a aprofundar os conhecimentos abordados nas aulas teóricas, nomeadamente a transmissão assíncrona de dados e a sua eficiência, a consolidar conhecimentos em C e na coordenação enquanto grupo.

Anexo I

nserial.c

#include "application.h"

int main(int argc, char \*\*argv)

{

    int fd;

    struct termios oldtio;

    if ((argc != 3 && argc != 4) ||

        ((strcmp("/dev/ttyS0", argv[1]) != 0) &&

         (strcmp("/dev/ttyS1", argv[1]) != 0) &&

         (strcmp("/dev/ttyS2", argv[1]) != 0)))

    {

        printf("Usage:\tnserial SerialPort TRANSMITTER(1)|RECEIVER(0)\n\tex: nserial /dev/ttyS1 1\n");

        exit(1);

    }

    if(atoi(argv[2]) == RECEIVER){

        fd = llopen(argv[1], RECEIVER);

        int size;

        size = llreadFile(fd);

        llclose(fd);

    }

    else if (atoi(argv[2]) == TRANSMITTER){

        fd = llopen(argv[1], TRANSMITTER);

        llopen\_image(argv[3], fd);

        llclose(fd);

    }

    else{

        printf("Usage:\tnserial SerialPort TRANSMITTER(1)|RECEIVER(0)\n\tex: nserial /dev/ttyS1 1\n");

        exit(1);

    }

    return 0;

}

Application.h

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <termios.h>

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <time.h>

#define TRANSMITTER 1

#define RECEIVER 0

void send\_file();

int llopen(unsigned char \*porta, bool transmitter);

int llreadFile(int fd);

int llwrite(int fd, unsigned char \* buffer, int length);

int llclose(int fd);

void llopen\_image(unsigned char \*path, int fd);

application.c

#include "application.h"

#include "protocol.h"

#define PACKAGE\_SIZE 256

struct termios oldtio;

bool trans;

void createDataPackage(unsigned char \*package, int indice, int num)

{

    unsigned char\* buffer;

    buffer = malloc(sizeof(unsigned char)\*(num + 5));

    unsigned char aux[2];

    buffer[0]='1';

    sprintf(aux, "%1x", indice%255);

    buffer[1]=aux[0];

    sprintf(aux, "%d", num/8);

    buffer[2]=aux[0];

    sprintf(aux, "%d", num%8);

    buffer[3]=aux[0];

    for (int i=0; i<num; i++)

        buffer[4+i]=package[i];

    for (int i=0; i<num+4; i++)

        package[i]=buffer[i];

    free(buffer);

}

void createControlPackage(unsigned char \*buffer, int controlCamp, int fileSize, unsigned char \*path)

{

    unsigned char aux[2];

    sprintf(aux, "%d",controlCamp);

    buffer[0]=aux[0];

    buffer[1]='0';

    buffer[2]='1';

    buffer[3]=fileSize;

    buffer[4]='1';

    sprintf(aux, "%d",strlen(path));

    buffer[5]=aux[0];

    buffer[6]='\0';

    strcat(buffer, path);

}

void llopen\_image(unsigned char \*path, int fd)

{

    FILE\* file;

    file = fopen(path, "r");

    if(file==NULL){

        perror("File opening failed ");

        exit(1);

    }

    fseek(file, 0L, SEEK\_END);

    int size = ftell(file);

    fseek(file,0L, SEEK\_SET);

    unsigned char packageControl[5 + strlen(path)];

    createControlPackage(packageControl, 0x02, size, path);

    llwrite(fd,packageControl, strlen(packageControl));

    int num, i=0;

    do{

        unsigned char \*packageBuf;

        packageBuf = malloc(sizeof(unsigned char) \* (PACKAGE\_SIZE + 4));

        num = fread(packageBuf, sizeof(unsigned char), PACKAGE\_SIZE, file);

        packageBuf[num] = '\0';

        createDataPackage(packageBuf, i, num);

        llwrite(fd, packageBuf, num+4);

        i++;

        free(packageBuf);

    } while(num == PACKAGE\_SIZE);

    createControlPackage(packageControl,3, size, path);

    llwrite(fd,packageControl, strlen(packageControl));

}

int llopen\_transmitter(unsigned char \*porta)

{

    int fd=open\_port(porta, &oldtio);

    send\_set(fd);

    receive\_ua\_snd(fd);

    return fd;

}

int llopen\_receiver(unsigned char \*porta)

{

    int fd=open\_port(porta, &oldtio);

    receive\_set(fd);

    send\_ua\_rcv(fd);

    return fd;

}

int llopen(unsigned char \*porta, bool transmitter)

{

    trans=transmitter;

    if (transmitter)

        llopen\_transmitter(porta);

    else

        llopen\_receiver(porta);

}

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

    int num;

    do{

        num=send\_msg(fd, buffer, length);

    } while(!receive\_data\_rsp(fd));

    return num;

}

int processPackage(unsigned char \*buffer, unsigned char \*filename, int sequenceNumber)

{

    switch(buffer[0]){

        case '1':{

            unsigned char aux[2];

            sprintf(aux, "%x", sequenceNumber);

            if (buffer[1] == aux[0])

            {

                int k = (int)buffer[2] + (int)buffer[3] \* 8;

                unsigned char aux2[PACKAGE\_SIZE+4];

                for (int i=0; i < PACKAGE\_SIZE+4; i++)

                    aux2[i] = buffer[i];

                for (int i=0; i<k; i++)

                    buffer[i] = aux2[i+4];

            }

            return 1;

        }

        case '2':

            if(buffer[4]=='1'){

                strncpy(filename, &buffer[6], (int)buffer[5]);

            }

            return 2;

        case '3':

            return 3;

        default:

            return -1;

    }

}

int llread(int fd, unsigned char \*buffer){

    int num = receive\_data(fd, buffer);

    return num;

}

int llreadFile(int fd ){

    unsigned char filename[128];

    unsigned char buf[1024];

    llread(fd, buf);

    processPackage(buf, filename, -1);

    FILE \*file;

    filename[0]='z';

    file = fopen(filename, "w");

    unsigned char \*aux;

    int num, sequenceNumber = 0;

    int ret;

    do{

        unsigned char buf[1024];

        num = llread(fd, buf);

        ret = processPackage(buf, filename, sequenceNumber);

        if(ret==1){

            fwrite(buf, sizeof(unsigned char), num-4, file);

            sequenceNumber++;

            sequenceNumber %= 255;

        }

    }  while(ret != 3);

    fclose(file);

    return num;

}

int llclose\_transmitter(int fd){

    send\_disc\_snd(fd);

    receive\_disc\_snd(fd);

    send\_ua\_snd(fd);

    close\_port(fd, &oldtio);

    return 0;

}

int llclose\_receiver(int fd){

    receive\_disc\_rcv(fd);

    send\_disc\_rcv(fd);

    receive\_ua\_rcv(fd);

    close\_port(fd, &oldtio);

    return 0;

}

int llclose(int fd){

    if (trans)

        return llclose\_transmitter(fd);

    else

        return llclose\_receiver(fd);

}

Protocol.h

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <termios.h>

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <signal.h>

#include <time.h>

int open\_port(char\* porta, struct termios \*oldtio);

void close\_port(int fd, struct termios \*oldtio);

void send\_set(int fd);

void send\_ua\_rcv(int fd);

void send\_ua\_snd(int fd);

void send\_disc\_rcv(int fd);

void send\_disc\_snd(int fd);

void send\_resp(int fd, char c, char a);

int send\_msg(int fd,unsigned char\* msg, int length);

int receive\_msg(int fd, unsigned char c, unsigned char a, bool data, unsigned char data\_buf[], bool data\_resp);

void receive\_set(int fd);

void receive\_disc\_rcv(int fd);

void receive\_disc\_snd(int fd);

void receive\_ua\_rcv(int fd);

void receive\_ua\_snd(int fd);

int receive\_data(int fd, unsigned char data\_buf[]);

bool receive\_data\_rsp(int fd);

protocol.c

/\*Non-Canonical Input Processing\*/

#include "protocol.h"

#define BAUDRATE B38400

#define \_POSIX\_SOURCE 1 /\* POSIX compliant source \*/

#define FALSE 0

#define TRUE 1

#define START 0

#define FLAG\_RCV 1

#define A\_RCV 2

#define C\_RCV 3

#define BCC\_OK 4

#define RCV\_DATA 5

#define BCC2\_OK 6

#define STOPS 7

#define FLAG 0x7E

#define A\_RCV\_RSP 0x03

#define A\_RCV\_CMD 0x01

#define A\_SND\_CMD 0x03

#define A\_SND\_RSP 0x01

#define SET 0x03

#define UA 0x07

#define DISC 0x0B

#define ESC 0x7D

#define ESC1 0x5E

#define ESC2 0x5D

#define RR\_R1 0x85

#define RR\_R0 0x05

#define REJ\_R1 0x81

#define REJ\_R0 0x01

#define C\_DATA\_S0 0x00

#define C\_DATA\_S1 0x40

#define MAX\_RETR 6

#define TIMEOUT 3

volatile int STOP = FALSE;

bool even\_bit = 0;

bool previous\_s = 0;

static int num\_retr\_set = 0;

static int num\_retr\_disc = 0;

static int num\_retr\_data = 0;

int fdG;

unsigned char msgG[1024];

int lengthG;

int state;

bool ua\_received=false;

bool data\_received=false;

bool disc\_received=false;

static int erro\_cab = 0;

static int erro\_dados = 0;

void alarmSet()

{

  if (ua\_received)

    return;

  if (num\_retr\_set < MAX\_RETR)

  {

    printf("alarmSet %d \n", num\_retr\_set);

    send\_set(fdG);

    num\_retr\_set++;

  }

  else

  {

    exit(1);

  }

}

void alarmDisc()

{

 if (disc\_received)

    return;

  if (num\_retr\_disc < MAX\_RETR)

  {

    printf("alarmDisc %d \n", num\_retr\_disc);

    send\_disc\_snd(fdG);

    num\_retr\_disc++;

  }

  else

  {

    exit(1);

  }

}

void alarmData()

{

  if (data\_received)

    return;

  if (num\_retr\_data < MAX\_RETR)

  {

    printf("alarmData %d \n", num\_retr\_data);

    send\_msg(fdG, msgG, lengthG);

    num\_retr\_data++;

  }

  else

  {

    exit(1);

  }

}

int open\_port(char\* porta, struct termios \*oldtio)

{

    int fd;

    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.

    \*/

    fd = open(porta, O\_RDWR | O\_NOCTTY);

    if (fd < 0)

    {

        perror(porta);

        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 1 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);

    }

    printf("New termios structure set\n");

    return fd;

}

void close\_port(int fd, struct termios \*oldtio)

{

    sleep(1);

    tcsetattr(fd, TCSANOW, oldtio);

    close(fd);

    printf("erros cabeçalho: %d\n", erro\_cab);

    printf("erros dados: %d\n", erro\_dados);

}

void send\_resp(int fd, char c, char a)

{

    unsigned char buf2[5];

    buf2[0] = FLAG;

    buf2[1] = a;

    buf2[2] = c;

    buf2[3] = a ^ c;

    buf2[4] = FLAG;

    write(fd, buf2, 5);

}

void send\_set(int fd)

{

    if(num\_retr\_set == 0)

        signal(SIGALRM, alarmSet);

    alarm(TIMEOUT);

    send\_resp(fd, SET, A\_SND\_CMD);

}

void send\_ua\_rcv(int fd)

{

    send\_resp(fd, UA, A\_RCV\_RSP);

}

void send\_ua\_snd(int fd)

{

    send\_resp(fd, UA, A\_SND\_RSP);

}

void send\_disc\_rcv(int fd)

{

    send\_resp(fd, DISC, A\_RCV\_CMD);

}

void send\_disc\_snd(int fd)

{

    if(num\_retr\_disc == 0)

        signal(SIGALRM, alarmDisc);

    alarm(TIMEOUT);

    send\_resp(fd, DISC, A\_SND\_CMD);

}

void send\_data\_response(int fd, bool reject, bool duplicated)

{

    if (reject && duplicated && even\_bit)

        send\_resp(fd, RR\_R1, A\_RCV\_RSP);

    else if (reject && duplicated && !even\_bit)

        send\_resp(fd, RR\_R0, A\_RCV\_RSP);

    else if (reject && even\_bit)

        send\_resp(fd, REJ\_R1, A\_RCV\_RSP);

    else if (reject && !even\_bit)

        send\_resp(fd, REJ\_R0, A\_RCV\_RSP);

    else if (!reject && even\_bit)

        send\_resp(fd, RR\_R1, A\_RCV\_RSP);

    else

        send\_resp(fd, RR\_R0, A\_RCV\_RSP);

}

int send\_msg(int fd, unsigned char\* msg, int length)

{

    if(num\_retr\_data==0){

        signal(SIGALRM, alarmData);

        fdG=fd;

        for(int i=0; i<length;i++){

            msgG[i]=msg[i];

        }

        lengthG=length;

    }

    alarm(TIMEOUT);

    unsigned char buf2[7 + length\*2];

    buf2[0] = FLAG;

    buf2[1] = A\_SND\_CMD;

    if (even\_bit)

        buf2[2] = C\_DATA\_S0;

    else

        buf2[2] = C\_DATA\_S1;

    buf2[3] = (A\_SND\_CMD ^ buf2[2]);

    unsigned char bcc2 = msg[0];

    for (int i = 1; i < length; i++)

        bcc2 = (bcc2 ^ msg[i]);

    int cnt=0;

    for (int i = 0; i < length; i++)

    {

        if (msg[i] == 0x7E)

        {

            buf2[4+i+cnt]=0x7D;

            cnt++;

            buf2[4 + i+cnt] = 0x5E;

        }

        else if (msg[i] == 0x7D)

        {

            buf2[4 + i + cnt] = 0x7D;

            cnt++;

            buf2[4 + i + cnt] = 0x5D;

        }

        else

            buf2[4 + i + cnt] = msg[i];

    }

    if (bcc2 == 0x7E)

    {

        buf2[4 + length + cnt] = 0x7D;

        cnt++;

        buf2[4 + length + cnt] = 0x5E;

    }

    else if (bcc2 == 0x7D)

    {

        buf2[4 + length + cnt] = 0x7D;

        cnt++;

        buf2[4 + length + cnt] = 0x5D;

    }

    else

        buf2[4 + length + cnt] = bcc2;

    buf2[5 + length +cnt] = FLAG;

    buf2[6 + length +cnt] = '\0';

    return write(fd, buf2, 6 + length+cnt);

}

int receive\_msg(int fd, unsigned char c, unsigned char a, bool data, unsigned char\* data\_buf, bool data\_resp)

{

    state = START;

    int res;

    bool escape = false;

    int cnt = 0;

    int random1, random2;

    while (state != STOPS)

    { /\* loop for input \*/

        unsigned char buf[2];

        res = read(fd, buf, 1); /\* returns after 1 char have been input \*/

        buf[1]='\0';

        unsigned char msg = buf[0];

        srand(time(0));

        random1 = (rand() % (10 - 1 + 1)) + 1;

        srand(time(0));

        random2 = (rand() % (1000 - 1 + 1)) + 1;

        switch (state)

        {

        case START:

            if (msg == FLAG){

                state = FLAG\_RCV;

            }

            break;

        case FLAG\_RCV:

            if (random1 == 1 && data)

            {

                msg = 0x00;

                a = 0x00;

            }

            if (msg == a)

                state = A\_RCV;

            else if (msg != FLAG){

               state = START;

                break;

            }

            break;

        case A\_RCV:

            if (!data && !data\_resp && msg == c)

                state = C\_RCV;

            else if (!data && !data\_resp)

                state = START;

            else if (data\_resp)

            {

                if ((int)msg == RR\_R1 || (int)msg == RR\_R0 || (int)msg == REJ\_R1 || (int)msg == REJ\_R0)

                {

                    data\_buf[0] = msg;

                    state = C\_RCV;

                    c=msg;

                }

                else

                    state = START;

            }

            else if (msg == C\_DATA\_S0 || msg == C\_DATA\_S1)

            {

                if (((bool)msg) != previous\_s)

                {

                    even\_bit = !(bool)msg;

                    state = C\_RCV;

                    c=msg;

                }

                else

                {

                    even\_bit = (bool)msg;

                    send\_data\_response(fd, true, (((bool)msg) == previous\_s));

                    state = START;

                }

            }

            else

            {

                state=START;

            }

            break;

        case C\_RCV:

            if (msg == (a^c)){

                state = BCC\_OK;

            }

            else

            {

                state = START;

                erro\_cab++;

                send\_data\_response(fd, true, false);

                return 0;

            }

            break;

        case BCC\_OK:

            if (random2 == 1 && data)

            {

                msg = 0;

            }

            if (msg == FLAG)

            {

                state = STOPS;

            }

            else if (data)

            {

                state = RCV\_DATA;

                if (msg == ESC)

                    escape = true;

                else {

                    char msg\_string[255];

                    sprintf(msg\_string, "%c", msg);

                    data\_buf[cnt]=msg\_string[0];

                    cnt++;

                }

            }

            else{

                state = START;

            }

            break;

        case RCV\_DATA:

            if (random2 == 1 && data)

            {

                msg = 0;

            }

            if (msg == FLAG)

            {

                cnt--;

                char bcc\_rcv = data\_buf[cnt];

                char bcc\_real = data\_buf[0];

                for (int i = 1; i < cnt; i++)

                    bcc\_real = bcc\_real ^ data\_buf[i];

                if (bcc\_real == bcc\_rcv)

                {

                    previous\_s = !even\_bit;

                    send\_data\_response(fd, false, false);

                }

                else

                {

                    data\_buf[0] = '\0';

                    even\_bit = !even\_bit;

                    erro\_dados++;

                    send\_data\_response(fd, true, false);

                }

                data\_buf[cnt] = '\0';

                state = STOPS;

                break;

            }

            if (escape)

            {

                if (msg == ESC1)

                {

                    data\_buf[cnt]=0x7E;

                    cnt++;

                }

                else if (msg == ESC2)

                {

                    data\_buf[cnt]=0x7D;

                    cnt++;

                }

                else

                    state = START; //ERRO

                escape = false;

                break;

            }

            if (msg == ESC){

                escape = true;

                break;

            }

            char msg\_string[255];

            sprintf(msg\_string, "%c", msg);

            data\_buf[cnt]=msg\_string[0];

            cnt++;

            break;

        default:

            state = START;

            break;

        }

    }

    return cnt;

}

void receive\_set(int fd)

{

    ua\_received=false;

    num\_retr\_set=0;

    receive\_msg(fd, SET, A\_RCV\_RSP, false, NULL, false);

}

void receive\_disc\_rcv(int fd)

{

    receive\_msg(fd, DISC, A\_SND\_CMD, false, NULL, false);

}

void receive\_disc\_snd(int fd)

{

    receive\_msg(fd, DISC, A\_RCV\_CMD, false, NULL, false);

    disc\_received = true;

}

void receive\_ua\_rcv(int fd)

{

    receive\_msg(fd, UA, A\_SND\_RSP, false, NULL, false);

}

void receive\_ua\_snd(int fd)

{

    receive\_msg(fd, UA, A\_RCV\_RSP, false, NULL, false);

    ua\_received=true;

}

int receive\_data(int fd, unsigned char\* data\_buf)

{

    int size;

    if (previous\_s == 0)

        size= receive\_msg(fd, C\_DATA\_S1, A\_SND\_CMD, true, data\_buf, false);

    else

        size = receive\_msg(fd, C\_DATA\_S0, A\_SND\_CMD, true, data\_buf, false);

    return size;

}

bool receive\_data\_rsp(int fd)

{

    data\_received=false;

    num\_retr\_data=0;

    unsigned char buf[1024];

    receive\_msg(fd, RR\_R0, A\_RCV\_RSP, false, buf, true);

    if(buf[0]==REJ\_R0 || buf[0] == REJ\_R1)

        return false;

    data\_received = true;

    even\_bit = !even\_bit;

  return true;

}

Anexo 2

## Medição tempo de execução

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Baudrate | Tempo | | | | | Tempo Médio | R | S |
| 300 | 1,135 | 1,133 | 1,131 | 1,125 | 1,117 | 1,1282 | 9721,68 | 32,4 |
| 600 | 1,132 | 1,132 | 1,116 | 1,120 | 1,139 | 1,1278 | 9725,13 | 16,2 |
| 1200 | 1,121 | 1,125 | 1,118 | 1,127 | 1,136 | 1,1254 | 9745,87 | 8,12 |
| 2400 | 1,126 | 1,124 | 1,125 | 1,126 | 1,122 | 1,1246 | 9752,80 | 4,06 |
| 4800 | 1,116 | 1,128 | 1,124 | 1,138 | 1,127 | 1,1266 | 9735,49 | 2,03 |
| 9600 | 1,126 | 1,135 | 1,151 | 1,124 | 1,119 | 1,131 | 9697,61 | 1,01 |
| 19200 | 1,126 | 1,128 | 1,122 | 1,128 | 1,126 | 1,126 | 9740,67 | 0,51 |
| 38400 | 1,134 | 1,133 | 1,121 | 1,127 | 1,132 | 1,1294 | 9711,35 | 0,25 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Tamanho | Tempo | | | Tempo Médio | R | S |
| 8 | 1,675 | 1,625 | 1,564 | 1,621 | 6764,803 | 0,176 |
| 16 | 1,373 | 1,353 | 1,385 | 1,370 | 8003,892 | 0,208 |
| 32 | 1,25 | 1,243 | 1,233 | 1,242 | 8830,918 | 0,230 |
| 64 | 1,182 | 1,179 | 1,209 | 1,190 | 9216,807 | 0,240 |
| 128 | 1,138 | 1,129 | 1,158 | 1,142 | 9607,007 | 0,250 |
| 256 | 1,125 | 1,125 | 1,123 | 1,124 | 9755,114 | 0,254 |
| 512 | 1,116 | 1,116 | 1,102 | 1,111 | 9869,226 | 0,257 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Erros BCC1 | Erros BCC2 | Tempo | R | S |
| 0 | 0 | 3,922 | 2796,532 | 0,072826 |
| 0 | 12 | 4,017 | 2730,396 | 0,071104 |
| 0 | 14 | 4,721 | 2323,237 | 0,060501 |
| 0 | 17 | 4,741 | 2313,436 | 0,060246 |
| 0 | 26 | 6,037 | 1816,796 | 0,047312 |
| 0 | 46 | 6,856 | 1599,767 | 0,041661 |
| 9 | 0 | 7,79 | 1407,959 | 0,036667 |
| 4 | 56 | 23,111 | 474,5792 | 0,012359 |