U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Redes de Computadores

Relatório do 1º trabalho laboratorial

Mestrado Integrado em Engenharia Informática e Computação

2020 / 2021

Afonso Maria Rebordão Caiado de Sousa up201806789@fe.up.pt

Marcelo Augusto Reis up201809566@fe.up.pt

Índice

Índice 1

Sumário 2

Introdução 2

Arquitetura 3

Estrutura do código 3

Casos de uso principais 5

Protocolo de Ligação Lógica 6

Protocolo de Aplicação 7

Validação 8

Eficiência do protocolo de ligação de dados 8

Conclusões 10

Anexo I 11

app_utils.c 11

protocol_utils.c 15

emissor/app_sender.c 17

emissor/protocol_sender.c 22

receptor/app_receiver.c 35

receptor/protocol_receiver.c 39

Anexo II 50

Anexo III 50

Anexo IV 54

Sumário

Este relatório foi realizado no âmbito da cadeira de Redes e Computadores, fazendo parte complementar do primeiro trabalho laboratorial da cadeira. Este trabalho consiste na implementação de uma aplicação capaz de efetuar a transferência de ficheiros, com a ajuda de uma Porta Série RS-232.

Após a realização deste trabalho, concluímos que foi um trabalho bem sucedido ao termos completado todos os objetivos do trabalho, obtendo uma aplicação capaz de fazer a transferência de um ficheiro com 321441 Bytes em 57.38 segundos sem perda de dados, com um tamanho de trama 65535 e uma taxa de transmissão (baudrate) 57600.

Introdução

Este trabalho tem dois principais objetivos. Implementar um protocolo de ligação de dados, que vá de acordo com o guião fornecido, assim como testar este protocolo com uma aplicação de transferência de ficheiros, igualmente especificada.

O relatório, por sua vez, tem como objetivo expor todos os detalhes do trabalho realizado, e também explicar uma componente mais teórica do trabalho, seguindo a seguinte estrutura:

- **Arquitetura**, onde são descritos os blocos funcionais e as interfaces presentes.
- **Estrutura de código**, onde apresentamos as API's, as principais estruturas de dados, as principais funções da nossa aplicação e a sua relação com arquitetura.
- **Casos de uso principais**, onde são identificados os principais casos de uso da nossa aplicação e a sua sequência de chamadas de funções.
- **Protocolo de ligação lógica**, onde identificamos os principais aspetos funcionais, descrevendo igualmente a estratégia utilizada para a implementação destes aspectos, tendo nesta parte apresentação de extratos de código.
- **Protocolo de aplicação**, onde identificamos os principais aspetos funcionais, descrevendo igualmente a estratégia utilizada para a implementação destes aspectos, tendo nesta parte apresentação de extratos de código.
- **Validação**, onde é feita a descrição dos testes efetuados.
- **Eficiência do protocolo de ligação de dados**, onde é feita uma caracterização estatística da eficiência do protocolo realizado, feita com recurso a medidas sobre o código desenvolvido.
- **Conclusões**, onde fazemos uma síntese da informação apresentada nas seções anteriores e uma reflexão sobre os objetivos de aprendizagem alcançados.

Arquitetura

O nosso trabalho está dividido em dois blocos funcionais: o bloco do emissor e o bloco do receptor. Para além destes dois principais blocos, temos dois ficheiros de código comuns a ambos, ***app_utils.c*** e ***protocol_utils.c*** que estão de fora da pasta de cada um destes blocos. O emissor é composto por duas camadas, uma responsável pela aplicação, no ficheiro ***app_sender.c***, e outra responsável pelo protocolo de ligação de dados, no ficheiro ***protocol_sender.c***. Da mesma forma, o receptor é composto pelos ficheiros ***app_receiver.c*** e ***protocol_receiver.c***.

Por outro lado, na interface da linha de comandos, é permitido ao utilizador de, após compilar a aplicação dentro do respetivo bloco com **$ make app**, e conhecendo a porta de série que quer usar (sob o formato /dev/ttySx), correr a aplicação com **./app /dev/ttySx**. No caso do emissor, o utilizador tem ainda que especificar qual o ficheiro que pretende enviar. Existem também parâmetros opcionais que permitem personalizar detalhes sobre a transferência de informação, sendo estes o tamanho de cada segmento de informação enviado e um baudrate. Desta forma, o utilizador pode correr a aplicação com valores inseridos por ele mesmo, ou com valores por defeito, **baudrate** = 38400 e **tamanho** = 1000 Bytes. Por último, implementamos também parâmetros adicionais ao receiver que estão relacionados com o teste da eficiência do protocolo de ligação. São estes **-FER**, que corresponde a uma probabilidade (%) de ocorrerem erros nas tramas de informação, e **-D**, que indica ao programa se devem existir atrasos no processamento das tramas recebidas.

Estrutura do código

Emissor

Principais funções da camada de ligação:

- `int llopen(int porta)`

Função que cria um identificador de dados.

- `int llwrite(int fd,char * buffer,int length)`

Função que escreve o conteúdo do buffer em fd.

- `int llclose(int fd)`

Função que fecha o identificador de dados.

O bloco do emissor tem ainda umas funções auxiliares para a cama de ligação, **atende -** função responsável por lidar com os SIGALARM, **changeNSS -** função que altera os valores de Ns e Nr, **byteStuffing -** função responsável pelo mecanismo de byte stuffing e **sendFrame_I -** função responsável por enviar a informação para o receptor.

Principais funções da camada da aplicação:

- `int main(int argc, char** argv)`

Função base da camada da aplicação, sendo esta responsável por chamar qualquer função utilizada da camada de ligação, assim como controlar e efetuar todas as responsabilidades da aplicação, como abrir o ficheiro, fechar o ficheiro, ler os argumentos, etc.

Receptor

Principais funções da camada de ligação:

- `int llopen(int porta)`

Função responsável por abrir a ligação série, ler a mensagem (trama) *set-up* (SET) e devolver a trama *unnumbered acknowledegment* (UA).

- `int llread(int fd,unsigned char * message)`

Função que lê as tramas de informação e faz o byte destuffing.

- `int llclose(int fd)`

Função que lê a trama de controlo *disconnect* (DISC), envia DISC para o emissor e lê a trama UA.

- `void testEfficiency(unsigned char * readed)`

Função que vai gerar erros na trama ou atrasos no processamento de cada trama recebida.

Principais funções da camada da aplicação:

- `int main(int argc, char** argv)`

Função base da camada da aplicação, sendo esta responsável por chamar qualquer função utilizada da camada de ligação, assim como controlar e efetuar todas as responsabilidades da aplicação, como criar um novo ficheiro, recolher informação sobre o tamanho do ficheiro, etc.
Esta camada contém ainda a função auxiliar **isAtEnd -** função que verifica se o pacote recebido é o pacote END (pacote que assinala o final da transmissão).

Bloco comum

Neste bloco temos funções auxiliares utilizadas por ambos os outros blocos, receptor ou emissor.

Principais funções da camada de ligação:

- **itoa -** função que converte inteiros para uma string.
- **sendFrame_S_U -** função que envia tramas de supervisão (S) e não numeradas (U)

Principais funções da camada da aplicação:

- **readArgValues -** função responsável por ler os argumentos introduzidos na linha de comandos.
- **getBaudRate -** função que verifica se o Baudrate inserido existe.

Temos ainda uma struct **baud_rate** que define um baudrate e uma struct **shell_inputs** que guarda os parâmetros passados pela shell (consultar **Anexo II**).

Casos de uso principais

Existem 2 principais casos de uso para o nosso programa: correr o programa como emissor, e correr o programa como receptor. Em ambos os casos, para utilizar o programa basta compilar, e correr utilizando o mesmo comando, apenas variando os argumentos. No caso do **emissor** o comando é:

```
./app /dev/ttySx file_name [--frame-size 1000 -B 38400]
```

E são chamadas as seguintes funções de forma sequencial:

- `readArgvValues(argc,argv,&arguments,TRANSMITTER)`, com `TRANSMITTER` = 1.
- `llopen(porta)`, para abrir a porta de série do lado do emissor.
- `llwrite(fd, controlo,tamanho_controlo)`, para enviar o pacote de controlo START ao receptor.
- `llwrite(fd,dados,tamanho_dados+bytesLidos)`, dentro de um ciclo, para enviar o pacote de dados ao receptor.
- `llwrite(fd, controlo,tamanho_controlo)`, para enviar o pacote de controlo END para o receptor.
- `llclose(fd)`, para fechar a ligação com a porta série do lado do emissor após a transferência ter sido feita.

No caso do **receptor** o comando é:

```
./app /dev/ttySx [-B 38400 -FER 0 -D 0]
```

E são chamadas as seguintes funções de forma sequencial:

- `readArgvValues(argc,argv,&arguments,RECEIVER)`, com `RECEIVER` = 0.
- `llopen(porta)`, para abrir a porta de série do lado do receptor.
- `llread(fd, start)`, para ler o pacote de controlo START enviado pelo emissor.
- `llread(fd, mensagemPronta)`, **dentro de um ciclo**, lê um pacote de informação enviado pelo emissor.
- `isAtEnd(start, sizeOfStart, mensagemPronta, sizeMessage)`, **dentro desse mesmo ciclo**, para verificar se o pacote lido é o pacote END que sinaliza o final da transmissão.
- `llclose(fd)`, para fechar a ligação com a porta série do lado do receptor após a transferência ter sido feita.

Protocolo de Ligação Lógica

O protocolo de ligação de dados é a camada de mais baixo nível da aplicação. Esta é a camada que trata de interagir diretamente com a porta de série. As funcionalidades desta camada são então de abrir / fechar a porta série, leitura / escrita de tramas de controlo ou informação, efetuar as operações de *byte stuffing* / *destuffing*.

Para tal, o nosso programa tem implementadas as 4 funções previstas para a camada de ligação de dados, da seguinte forma:

NOTA: Sempre que o emissor aguarda uma resposta do receptor, cria um alarme, e se não receber resposta após um tempo *time-out*, reenvia a trama em questão. Ao terceiro *time-out* ocorrido, o programa termina.

- **LLOPEN**

```
int llopen(int porta)
```

Esta função é responsável por estabelecer a ligação através da porta série entre o receptor e o emissor.

Do lado do emissor, como especificado no guião, a função envia uma trama de controlo SET para o receptor, e aguarda a receção da resposta UA por parte do receptor.

Do lado do receptor, como especificado no guião, a função aguarda a receção da trama SET enviada pelo emissor e, aquando da sua chegada, responde com a mensagem de controlo UA.

Para fazer o envio da trama SET e da trama UA, o emissor e o receptor utilizam a função auxiliar **sendFrame_S_U**, que recebe como argumento o nome do tipo de trama de controlo a enviar, sendo assim capaz de enviar qualquer um dos dois. A leitura das tramas é efetuada com o auxílio de uma máquina de estados.

- **LLCLOSE**

```
int llclose(int fd)
```

Esta função é responsável por terminar a ligação através da porta série entre o receptor e o emissor.

Do lado do emissor, como especificado no guião, a função envia uma trama de controlo DISC para o receptor com **sendFrame_S_U**, aguarda a receção da resposta DISC por parte do receptor, para finalmente enviar a trama UA, novamente com a função **sendFrame_S_U**.

Do lado do receptor, a função aguarda a receção da trama DISC, responde com DISC e aguarda a receção da trama final UA, mais uma vez com recurso à função **sendFrame_S_U**, e à máquina de estados para a leitura das tramas.

- **LLWRITE**

```
int llwrite(int fd,char * buffer,int length)
```

Esta função utilizada pelo emissor é responsável por efetuar o envio das tramas de informação, assim como de efetuar a operação de *byte stuffing*.

Para a implementação desta função, começamos por chamar uma função auxiliar **byteStuffing**, que trata simultaneamente de aplicar o mecanismo de stuffing à mensagem a enviar, e de fazer a verificação do BCC após o *byte stuffing*. De seguida, construímos a trama final a enviar (com **sendFrame_I**) ao receptor ao acrescentar o cabeçalho do protocolo de ligação (*framing*).

Para além do mecanismo de *time-out* utilizado nesta função, esta função efetua o reenvia da trama de informação se, na máquina de estados, receber uma resposta negativa da parte do receptor (**REJ**).

- **LLREAD**

```
int llread(int fd,unsigned char * message)
```

Esta função utilizada pelo receptor é responsável por ler as tramas de informação, assim como de efetuar a operação de *byte destuffing*.

A leitura das tramas é efetuada graças a uma máquina de estados. A verificação do **BCC2**, assim como o mecanismo de *byte destuffing* está incorporada na máquina. Se o **BCC** recebido está correto, o receptor responde ao emissor com confirmação positiva (**RR**), senão, responde com **REJ**. A mensagem de controlo enviada depende igualmente do número de sequência da trama recebida **N(s)**. Por outro lado, esta máquina possui um estado (**ESC_STATE**) onde é efetuado o *destuffing*, quando necessário.

Protocolo de Aplicação

O protocolo de aplicação é a camada de mais alto nível do nosso programa. Esta é a parte responsável por tratar do processo de envio e de receção do ficheiro especificado, com ajuda das funções do protocolo de ligação de dados. Mais concretamente, as funcionalidades implementadas por esta camada são as seguintes:

- Envio / Recepção dos pacotes de controlo **START** e **END** que assinalam o início / fim da transmissão. Contêm o nome e tamanho do ficheiro em questão.

Esta funcionalidade foi implementada com recurso às funções anteriormente mencionadas **llwrite** e **llread**. Para fazer a leitura do pacote **END**, foi implementada uma função **isAtEnd** que verifica se a trama lida é a trama de **END**.

- Fazer a divisão do ficheiro em diferentes segmentos a serem enviados no caso do emissor, ou de reconstruir o ficheiro fonte a partir desses fragmentos, no caso do receptor.
- Acrescentar um cabeçalho contendo o número de sequência do pacote e o tamanho do fragmento.
- Efetuar a leitura / criação do ficheiro especificado.

No nosso programa, estas funcionalidades foram implementadas diretamente na função main, quer do emissor, quer do receptor, e podem ser consultadas em parte no **Anexo III**.

Validação

Para verificar o bom funcionamento e a eficácia do nosso programa, foram efetuados os seguintes testes:

- Transferência de ficheiros de diferentes tamanhos.
- Transferência de um ficheiro com variação do tamanho de trama e/ou da taxa de transmissão.
- Breve interrupção da ligação da Porta Série aquando da transferência de um ficheiro.
- Geração de ruído aquando da transferência de um ficheiro.

Estes testes foram todos concluídos com sucesso, tendo eles sido também realizados na presença do professor, no momento de avaliação. O resultado para alguns destes testes pode nomeadamente ser visto no **Anexo IV**.

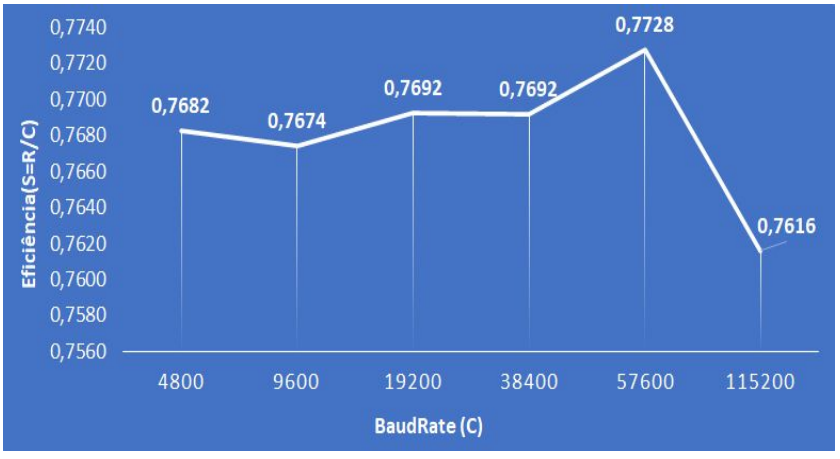
Eficiência do protocolo de ligação de dados

Com o objetivo de avaliar a eficiência do protocolo desenvolvido, foram realizados quatro testes variando parâmetros específicos e distintos, em cada um deles foi elaborado, a partir dos dados obtidos , uma tabela e um gráfico. Todas as tabelas podem ser encontradas no **Anexo IV**.

Variação da capacidade da ligação (C)

Para o gráfico seguinte foi utilizado um tamanho de trama de 5000 Bytes e uma imagem de 10968 Bytes.

Com este gráfico podemos concluir que até uma capacidade de ligação de 57600 a eficiência tende a aumentar mas a partir desse valor é visível que a eficiência diminui significativamente.



Variação do tamanho das tramas I

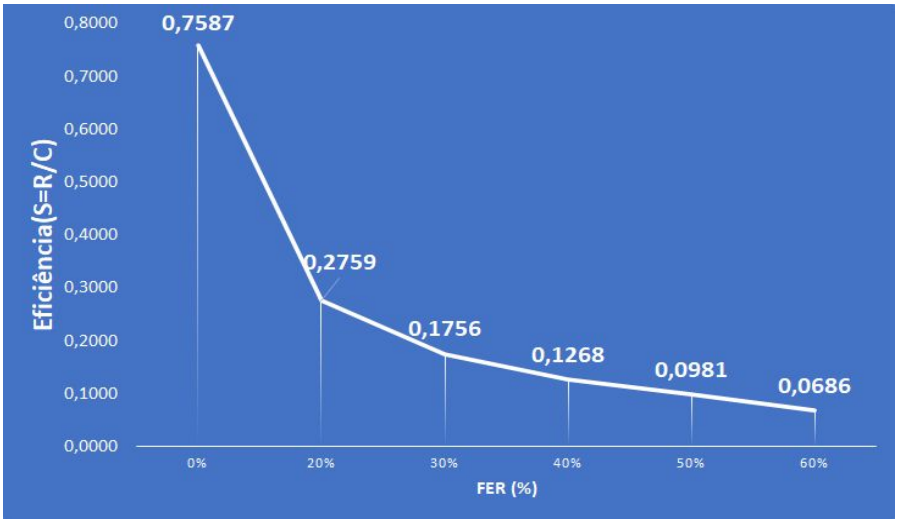
Para o gráfico seguinte foi utilizado um baudrate de 38400 e dois ficheiros de tamanhos diferentes, um com 10968 Bytes e outro com 321441 Bytes.

Com o gráfico abaixo podemos afirmar que quanto maior for o tamanho de cada pacote de dados a transmitir, maior é a eficiência do protocolo. Isto acontece porque ao aumentar o tamanho do pacote faz com que sejam enviadas menos tramas I, aumentando a velocidade de execução do programa.

Variação do FER

Para o gráfico que se encontra a seguir foi utilizado um baudrate de 38400 e um ficheiro com tamanho de 10960 Bytes.

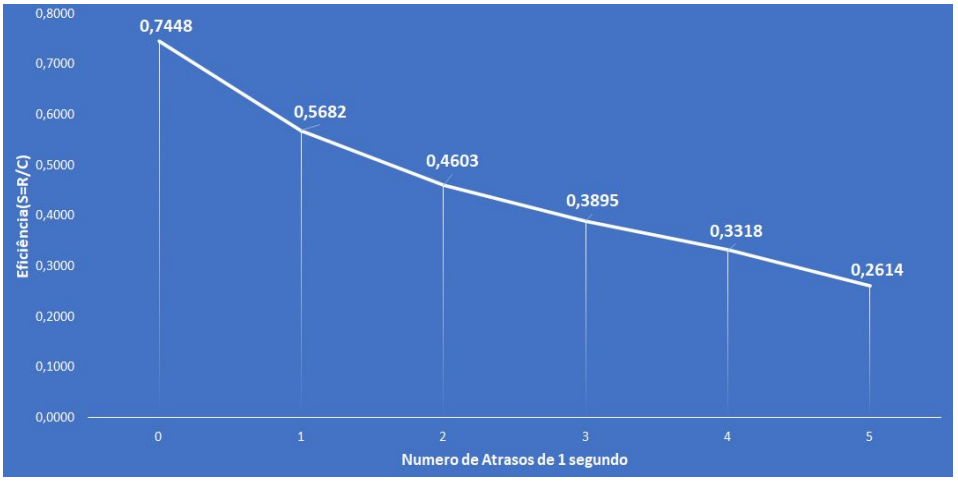
Com este gráfico podemos concluir, como previsto, que o aumento do FER prejudica e muito a eficiência do protocolo. Se os erros forem no cabeçalho o receptor ignora a trama e aguarda que o emissor a reenvie, causando *time-outs*. No caso de serem nos dados o receptor informa o emissor com um REJ, e pede que a trama seja enviada novamente atrasando a execução da aplicação.



Variação do T_prop

Para o gráfico seguinte foi utilizado um baudrate de 38400 e um ficheiro com tamanho de 10960 Bytes.

Com este gráfico é possível concluir, como era esperado, que o aumento de atraso no processamento de uma trama I diminui a eficiência do protocolo.



O protocolo *Stop and Wait,* um dos protocolos mais comuns dos esquemas **ARQ,** implementa mecanismos que solicitam automaticamente o reenvio de pacotes em falta ou com erros. Este protocolo é caracterizado pela seguinte sequência, o emissor transmite informação em tramas I, aguarda pela confirmação positiva **ACK** por parte de receptor. O receptor ao receber a trama I pode responder de duas formas diferentes, se a trama estiver correta envia uma confirmação **ACK,** caso contrário envia **NACK.** Por fim o emissor, dependendo da resposta dada pelo receptor, ou procede e transmite uma nova trama ao receber **ACK** ou retransmite novamente a trama I caso tenha recebido **NACK**.

Um dos problemas que pode acontecer neste protocolo é a perda de tramas I e de respostas **ACK** ou **NACK**. Para resolver este problema é proposto usar *time-outs,* caso o emissor não receba uma resposta por parte do receptor num intervalo de tempo definido este deve reenviar novamente a trama.

Na nossa aplicação foi usado o protocolo *Stop and Wait* para gerir a ocorrência de erros. Quando o emissor envia tramas I, com um número de sequência **Ns** que pode variar entre 0 e 1, aguarda pela resposta de receptor. Esta resposta pode ser **RR** quando a trama recebida não tem erros ou **REJ** caso contrário. Os **REJ** e **RR** são acompanhados por outro número de sequência **Nr,** que para o **REJ** indica a trama que foi rejeitada e para o RR a nova trama que deve ser enviada.

Conclusões

Em suma, o protocolo deste trabalho pode ser dividido em duas camadas **independentes entre elas**: a cama da *Ligação de Dados*, que interage diretamente com a porta série, e a camada da *Aplicação*, que faz uso da camada anterior para enviar / receber o ficheiro. Também é correto afirmar que este protocolo consiste em fornecer uma ligação de dados fiável entre dois sistemas ligados através um cabo série.

Tendo sido um trabalho desafiante e tendo implicado várias horas de empenho e trabalho, o protocolo foi implementado com sucesso e permitiu ao grupo de consolidar os conceitos teóricos presentes no trabalho, tais como o mecanismo de **stuffing**, **framing**, **stop&wait**, etc.

Anexo I

app_utils.c

```
#include <sys/types.h>
#include <termio.h>
#define NUM_BAUND_RATES 19
#define TRANSMITTER 1
#define RECEIVER 0
#define DEFAULT_BAUDRATE B38400
#define DEFAULT_FRAME_SIZE 1000

//struct que define um baudrate
struct baud_rate{
 speed_t baud; //baudrate em speed_t
 char *bauds; //baudrate em string
};

//guarda os parametros passados pela shell
struct shell_inputs
{
 char port[15]; //Dispositivo /dev/ttySx, x= 0,1,10,11
 char file_name[200]; //nome do ficheiro a enviar
 int frame_size; //tamanho maximo da quantidade de dados que devem ser enviados
de uma vez
 speed_t baudrate; // baudrate da ligação serie
 int FER; //frame error ratio
 int atraso; //boleano que indica se há ou não há atraso
};

//struct que vai guardar os baudrates para melhor acessibilidade
struct baud_rate rates[NUM_BAUND_RATES] = {
 {B0,"0"},
 {B50,"50"},
 {B75,"75"},
 {B110,"110"},
 {B134,"134"},
 {B150,"150"},
 {B200,"200"},
 {B300,"300"},
 {B600,"600"},
```

```
    {B1200,"1200"},
    {B1800,"1800"},
    {B2400,"2400"},
    {B4800,"4800"},
    {B9600,"9600"},
    {B19200,"19200"},
    {B38400,"38400"},
    {B57600,"57600"},
    {B115200,"115200"},
    {B230400,"230400"},
   };

   /**
   * vai procurar na struct de baudrates se o baudrate inserido existe
   * @param rate baudrate em string
   * @return baudrate em speed_t ou -1 em caso de erro
   **/
   speed_t getBaudRate(char * rate){
      for (size_t i = 0; i < NUM_BAUND_RATES; i++)
      {
          if (strcmp(rates[i].bauds,rate) == 0){
              return rates[i].baud;
          }
      }
      return -1;
   }

   /**
   * função responsavel por ler os argumentos introduzidos na linha de comandos
   * @param argc tamanho do array argv
   * @param argv array que contém os argumentos introduzidos
   * @param arguments struct que vai guardar os valores dos argumentos introduzidos
   * @param status flag que indica se é o emissor ou o receptor
   * @return 0 em caso de sucesso e -1 caso contrário
   **/
   int readArgvValues(int argc ,char *argv[],struct shell_inputs *arguments, int
status){
    strcpy(arguments->port,argv[1]);
    if(status){ // se for o emissor
      strcpy(arguments->file_name,argv[2]);

      //coloca os valores default de baudrate e frame size
```

```
        arguments->baudrate = DEFAULT_BAUDRATE;
        arguments->frame_size = DEFAULT_FRAME_SIZE;

        //percorre os restantes valores de argv
        for (size_t i = 3; i < argc; i++){
            if ( (strcmp(argv[i],"--frame-size") == 0 ) && ((i+1)<argc) ){ // lê o
valor da frame size introduzido
            arguments->frame_size = atoi(argv[i+1]);
             if (arguments->frame_size <= 0){ // verifica se o valor de frame size é
válido
                printf("Invalid Frame Size, the template command is:\n./app /dev/ttySx
file_name [--frame-size 1000 -B 38400]\n");
              return -1;
            }
            i++;
          }else if ( (strcmp(argv[i],"-B") == 0) && ((i+1)<argc) ){ // lê o valor da
baudrate introduzido
            arguments->baudrate = getBaudRate(argv[i+1]);
            if (arguments->baudrate == -1){ // se o valor de baudrate estiver errado
                printf("Invalid Baudrate, the template command is:\n./app /dev/ttySx
file_name [--frame-size 1000 -B 38400]\n");
              return -1;
            }
            i++;
          }else{ // se existir um argumento invalido
                printf("Invalid inputs, the template command is:\n./app /dev/ttySx
file_name [--frame-size 1000 -B 38400]\n");
            return -1;
          }
        }
      }else{ // se for o receptor
        //coloca o valor default do baudrate
        arguments->baudrate = DEFAULT_BAUDRATE;
        arguments->FER = 0; //valor default de FER
        arguments->atraso = 0; //valor default de atraso

        //percorre os restantes valores de argv
        for (size_t i = 2; i < argc; i++)
        {
            if ( (strcmp(argv[i],"-B") == 0) && ((i+1)<argc) ){ // lê o valor da
baudrate introduzido
            arguments->baudrate = getBaudRate(argv[i+1]);
```

```
          if (arguments->baudrate == -1){ // se o valor de baudrate estiver errado
              printf("Invalid Baudrate, the template command is:\n./app /dev/ttySx
[-B 38400 -FER 0 -D 0]\n");
            return -1;
          }
          i++;
         }else if ( (strcmp(argv[i],"-FER") == 0) && ((i+1)<argc) ){ // lê o valor
de FER introduzido
          arguments->FER = atoi(argv[i+1]);
           if ( (arguments->FER > 100) || (arguments->FER < 0) ){ // se o valor de
FER estiver errado
              printf("Invalid FER, the template command is:\n./app /dev/ttySx [-B
38400 -FER 0 -D 0]\n");
            return -1;
          }
          i++;
         }else if ( (strcmp(argv[i],"-D") == 0) && ((i+1)<argc) ){ // lê o valor de
atraso introduzido
          arguments->atraso = 1;
          i++;
        }else{ // se existir um argumento invalido
            printf("Invalid inputs, the template command is:\n./app /dev/ttySx [-B
38400 -FER 0 -D 0]\n");
          return -1;
        }
      }
    }

    return 0;
   }
```

protocol_utils.c

```
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1
#define MODEMDEVICE "/dev/ttyS"

#define ESC 0x7D
#define ESCFLAG 0x5E
#define ESCESC 0x5D
#define NEUTROXOR 0x00  // 0x00 é o elemento neutro do XOR

enum State { START, FLAG_RCV, A_RCV, C_RCV, BCC_OK, ESC_STATE, STOP}; // estados possiveis da maquina de estados usada
enum C { SET = 0x03, UA = 0x07, DISC = 0x0b, I0 = 0x00, I1 = 0x40, RR0 = 0x05, RR1 = 0x85, REJ0 = 0x01, REJ1 = 0x81}; // valores possiveis do campo C na trama

const unsigned char FLAG = 0x7e; // flag de inicio e fim de uma trama

/**
* converte inteiros para string
* @param i inteiro a ser convertido
* @param b array de caracteres destino
**/
void itoa(int i, char b[]){
    char const digit[] = "0123456789";
    char* p = b;
    if(i<0){
        *p++ = '-';
        i *= -1;
    }
    int shifter = i;
    do{ //Move to where representation ends
        ++p;
        shifter = shifter/10;
    }while(shifter);
    *p = '\0';
    do{ //Move back, inserting digits as u go
        *--p = digit[i%10];
        i = i/10;
    }while(i);
```

```
    }

    /**
    * função que envia tramas de supervisão (S) e não numeradas (U)
    * @param fd identificador da ligacao de dados
    * @param campo_endereco campo de endereco da trama a enviar
    * @param campo_controlo campo de controlo da trama a enviar
    **/
    void sendFrame_S_U(int fd, unsigned char campo_endereco, unsigned char
campo_controlo){
     unsigned char buffer[5]; // array que vai guardar a trama a ser enviada
     buffer[0] = FLAG;
     buffer[1] = campo_endereco;
     buffer[2] = campo_controlo;
     buffer[3] = campo_endereco ^ campo_controlo;
     buffer[4] = FLAG;

     write(fd,buffer,sizeof(buffer));
    }
```

emissor/app_sender.c

```
#include "protocol_sender.c"
#include "../app_utils.c"

#define DADOS 1
#define START 2
#define END 3

//indentificadores de informação
enum T { TAMANHO = 0 , NOME = 1};

int main(int argc, char** argv)
{
    int fd, porta , num_sequencia = 0; // (fd) - descritor da ligacao de dados, (porta) - porta serie, (num_sequencia) - numero de sequencia do pacote de dados
     int frame_size; //tamanho máximo em bytes dos dados contidos no pacote de dados
    unsigned int tamanho; //tamanho do ficheiro em blocos de 1 byte
    FILE * ficheiro; // ficheiro que irá ser enviado
    char * nome_ficheiro; // nome do fiheiro a ser enviado
    int tamanho_dados = 4; //tamanho do pacote de dados inicialmente
    int tamanho_controlo = 5; //tamanho do pacote de controlo inicialmente
     char * controlo = (char *) malloc(tamanho_controlo); // guarda a informação que irá conter um pacote de controlo
    char * dados = (char * )malloc(tamanho_dados); // guarda a informação que irá conter um pacote de dados
      struct shell_inputs arguments; // struct que vai guardar os argumentos introduzidos

    //verifica se foi introduzido pelo menos 3 argumentos, e se segundo argumento começa por /dev/ttyS
    if ( (argc < 3) || (strncmp("/dev/ttyS",argv[1],9) != 0) ) {
    printf("Invalid inputs, the template command is:\n./app /dev/ttySx file_name (--frame-size 1000 -B 38400)\n");
    exit(1);
    }

    //vai buscar ao array argv os argumentos introduzidos
    if (readArgvValues(argc,argv,&arguments,TRANSMITTER) == -1){
        exit(-1);
```

```
    }

    //atribuição dos argumentos lidos
    frame_size = arguments.frame_size;
    if (frame_size > 65535){ // caso o frame_size exceda o tamanho máximo
        printf("Frame size exceeded!\n");
        exit(-1);
    }
    nome_ficheiro = arguments.file_name;
    BAUDRATE = arguments.baudrate;
    porta = atoi(&arguments.port[9]);

    fd = llopen(porta);

    // verificar se a porta serie foi aberta com sucesso
    if (fd == -1){
        free(controlo);
        free(dados);
        exit(-1);
    }

    //abre o ficheiro para leitura
    ficheiro = fopen(nome_ficheiro,"r");

    //verificar se o ficheiro foi aberto com sucesso
    if (ficheiro !=NULL){

        //calcular o tamanho do ficheiro
        fseek(ficheiro,0,SEEK_END);
        tamanho = ftell(ficheiro);

        rewind(ficheiro); //voltar ao inicio do ficheiro

        //construção do pacote de controlo de START
        controlo[0] = START;
        controlo[1] = TAMANHO;

          //passar o tamanho do ficheiro para string e dps adicionar ao array
controlo
        char size[10];
        sprintf(size,"%d",tamanho);
         int num_blocos_tamanho = strlen(size); //guarda o numero de blocos de 1
```

```
byte que são necessários para guardar a informação do tamanho do ficheiro
            controlo[2] = num_blocos_tamanho;
            tamanho_controlo += num_blocos_tamanho;

               controlo = (char * )realloc(controlo,tamanho_controlo); //alocar mais
memoria para ser possivel guardar os blocos que correspondem ao tamanho do ficheiro

            memcpy(controlo+3,size,num_blocos_tamanho);

            //adicionar ao pacote de controlo a informação do nome do ficheiro
            controlo[3+num_blocos_tamanho] = NOME;
            controlo[4+num_blocos_tamanho] = strlen(nome_ficheiro);

            tamanho_controlo += strlen(nome_ficheiro);
               controlo= (char * ) realloc (controlo,tamanho_controlo);//necessário
alocar mais memória para guardar o nome do ficheiro

memcpy(controlo+5+num_blocos_tamanho,nome_ficheiro,strlen(nome_ficheiro));

            //envia o pacote de controlo START ao receptor
            int bytesEscritos = llwrite(fd, controlo,tamanho_controlo);

            if ( bytesEscritos < 0) // se ocorreu um erro no llwrite
                exit(-1);

            //envio de dados
            char buffer[frame_size];
            int bytesLidos = fread(buffer,1,frame_size,ficheiro);
            dados[0] = DADOS;
            while (bytesLidos > 0){ //enquanto houver dados para enviar
                //construção do pacote de dados
                dados[1] = num_sequencia;
                dados[2] = bytesLidos/255;
                dados[3] = bytesLidos%255;

                      dados = (char * )realloc(dados,tamanho_dados + bytesLidos);
//adiciona mais espaço ao array para poder guardar a informação do ficheiro

                for (size_t i = 0; i < bytesLidos; i++)
                {
                    dados[4+i] = buffer[i];
                }
```

```
            //envia o pacote de dados para o receptor
            bytesEscritos = llwrite(fd,dados,tamanho_dados+bytesLidos);

            if ( bytesEscritos < 0){ // se ocorreu um erro no llwrite
                free(controlo);
                free(dados);
                exit(-1);
            }

               if(num_sequencia == 255) // caso o numero de sequencia já tenha
chegado ao máximo
                num_sequencia = 0;
            else
                num_sequencia++; //incrementar o numero de sequência

            bytesLidos = fread(buffer,1,frame_size,ficheiro);
        }

        //contrução do pacote de controlo END
        controlo[0] = END;

        //envia o pacote de controlo END para o receptor
        bytesEscritos = llwrite(fd, controlo,tamanho_controlo);

        if ( bytesEscritos < 0) // se ocorreu um erro no llwrite
            exit(-1);


    }else{ // caso do ficheiro indicado pelo utilizador não existir
        printf("O ficheiro %s não existe.\n",nome_ficheiro);
        free(controlo);
        free(dados);
        llclose(fd);
        exit(-1);
    }

    //fecha o ficheiro lido
    fclose(ficheiro);

    //verificar se a porta serie foi fechada com sucesso
    if (llclose(fd) == -1){
```

```
            free(controlo);
            free(dados);
            exit(-1);
        }

        //liberta a memoria utilizada
        free(controlo);
        free(dados);

        return 0;
    }
```

emissor/protocol_sender.c

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

#include "../protocol_utils.c"

#define DESLIGAR_ALARME 0
#define TEMPO_ESPERA 5
#define MAX_TENTATIVAS 3

enum A { AC = 0x03 , AR = 0x01 }; // valores possiveis do campo A na trama

speed_t BAUDRATE; // baudrate que irá ser utilizado na ligação série

int Ns = 0; // numero de sequencia da trama do emissor
int Nr = 1; // numero de sequencia da trama do receptor

int time_out =FALSE; // flag que indica se ocorreu um time out
int num_tentativas = 0; // numero de tentativas de retransmissão

struct termios oldtio,newtio; //variaveis utilizadas para guardar a configuração
da ligação pelo cabo serie

/**
* função responsavel por lidar com os SIGALARM
**/
void atende()
{
 num_tentativas++;
 if (num_tentativas<=MAX_TENTATIVAS)
   printf("Time Out...%i\n",num_tentativas);
 time_out = TRUE;
}
```

```
    /**
    * altera os valores de Ns e Nr
    **/
    void changeNSS(){
     if (Ns){
       Ns=0;
       Nr=1;
     }
     else{
       Ns=1;
       Nr=0;
     }
    }

    /**
    * mecanismo de byte stuffing
    * @param buffer array de caracteres que vai sofrer byte stuffing
    * @param length tamanho do array buffer
    * @param new_buffer array que irá receber o conteudo do buffer após o byte stuffing
    * @param BCC array que contem o(s) valore(s) do parametro BCC que protege os dados
    * @return tamanho do new_buffer
    **/
    int byteStuffing(char * buffer,int length,char new_buffer[],char BCC[]){
     int j=0; // variavel que vai conter o tamanho de new_buffer, mas também irá funcionar de indice do array new_buffer

     //byte stuffing do buffer
     for (int i = 0; i < length; i++)
     {
       if (buffer[i] == FLAG){
         BCC[0] = BCC[0] ^ FLAG;
         new_buffer[j] = ESC;
         new_buffer[j+1] = ESCFLAG;
         j += 2;
       }else if (buffer[i] == ESC){
         BCC[0] = BCC[0]^ ESC;
         new_buffer[j] = ESC;
         new_buffer[j+1] = ESCESC;
         j += 2;
       }else{
```

```
        BCC[0] = BCC[0] ^ buffer[i];
        new_buffer[j] = buffer[i];
        j++;
      }
    }

    //verifica se o BCC é afetado pelo byte stuffing
    if(BCC[0] == FLAG){
      BCC[0] = ESC;
      BCC[1] = ESCFLAG;
    }else if (BCC[0] == ESC){
      BCC[0] = ESC;
      BCC[1] = ESCESC;
    }
    return j;
   }

   /**
   * envia a informação para o receptor (tramas de informação (I))
   * @param fd identificador da ligação de dados
   * @param trama array de caracteres a enviar
   * @param trama_length indica o tamanho da trama a enviar
   **/
   void sendFrame_I(int fd, char * trama, int trama_length){

    int bytesSended = 0; // bytes que já foram enviados
     int indice = 0; // indica o indice da trama onde deve começar a enviar a
informação
    int remaning = trama_length; // indica a quantidade de bytes que faltam enviar
    while (bytesSended != remaning )
    {
      bytesSended = write(fd,&trama[indice],remaning); // envia a informação para o
receiver
      if ( (bytesSended != remaning) && (bytesSended != -1) ){
        indice += bytesSended; //atualiza o indice do array
          remaning -=bytesSended; // decrementa a quantidade de bytes que faltam
enviar
      }
    }
   }

   /**
```

```
* cria um identificador de ligação de dados
* @param  porta porta de comunicação
* @return identificador de ligação em caso de sucesso e -1 em caso de erro
**/
int llopen(int porta){
 int fd; // identificador da ligação de dados
 char temp[3]; // guarda a porta no tipo string
 itoa(porta,temp);
 char modemDevise [9 + strlen(temp)]; // nome do modem devise p.e /dev/ttyS10
  //construção do string que contem o path do modem device
 strcpy(modemDevise,MODEMDEVICE);
 strcat(modemDevise,temp);

 fd = open(modemDevise, O_RDWR | O_NOCTTY );
 if (fd <0) {perror(modemDevise); 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]     = 0;   /* blocking read until 0 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");

     (void) signal(SIGALRM, atende); // redireciona todos os SIGALRM para a função
atende

    // envia trama SET
    sendFrame_S_U(fd,AC,SET);
    printf("Sended: SET\n");
    alarm(TEMPO_ESPERA); // cria um alarm para gerar os time outs
      unsigned char readed; // variavel que vai guardando a informação byte a byte
da trama recebida

    enum State actualState = START; // estado inical da maquina de estados

    // Logical Connection
    while (actualState != STOP)
    {
      read(fd,&readed,1); //lê um byte da informação recebida

        if (num_tentativas > MAX_TENTATIVAS){ // se excedeu o limite maximo de
tentativas
        //repor os valores standart para não afetar outras funções
        num_tentativas = 0;
        time_out = FALSE;
        printf("Error llopen: No answer from receiver, closing connection.\n");
        if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
          perror("tcsetattr");
          exit(-1);
        }
        close(fd);
        return -1;
      }
      // envia outra vez a informação de SET caso tenha ocorrido time out
      if (time_out){
        sendFrame_S_U(fd,AC,SET);
        printf("Sended: SET\n");
        alarm(TEMPO_ESPERA);
        time_out = FALSE;
      }

      // maquina de estados que valida a informação recebida
```

```
      switch (actualState)
      {
      case START:{ // recebe byte FLAG
        if (readed == FLAG)
          actualState = FLAG_RCV;
        break;
      }
      case FLAG_RCV:{ // recebe byte AC
        if (readed== AC)
          actualState = A_RCV;
        else if (readed != FLAG)
          actualState = START;
        break;
      }
      case A_RCV:{ // recebe byte UA
        if(readed == UA)
          actualState = C_RCV;
        else if(readed == FLAG)
          actualState = FLAG_RCV;
        else
          actualState = START;
        break;
      }
      case C_RCV:{ // recebe byte BCC
        if(readed == (AC ^ UA))
          actualState = BCC_OK;
        else if(readed == FLAG)
          actualState = FLAG_RCV;
        else
          actualState = START;
        break;
      }
      case BCC_OK:{ // recebe byte FLAG
        if (readed == FLAG)
          actualState = STOP;
        else
          actualState = START;
        break;
      }
      default:
        break;
      }
```

```
    }
    //repor o numero de tentativas para não afetar outras funções;
    num_tentativas = 0;
    printf("Received: UA\n");
     alarm(DESLIGAR_ALARME); //desliga o alarme porque já foi recebida a resposta pretendida
     printf("Logical Connection...Done\n");

    return fd;
   }

   /**
   * escreve o conteudo do buffer em fd
   * @param fd identificador da ligação de dados
   * @param buffer array de caracteres a transmitir
   * @param length tamanho do array buffer
   * @return numero de caracteres escritos ou -1 em caso de erro
   **/
   int llwrite(int fd,char * buffer,int length){
    int trama_length = 6; // tamanho da trama em bytes, não inclui o tamanho dos dados

    char new_buffer[length*2]; //guarda o conteudo de buffer após o mecanismo de byte stuffing
      char BCC_Dados[2]; // array que irá conter os valores de BCC (o array tem tamanho 2 pois o BCC pode ser afetado pelo byte stuffing)
    BCC_Dados[0] = NEUTROXOR;
    BCC_Dados[1] = NEUTROXOR;

    int new_buffer_size = byteStuffing(buffer,length,new_buffer,BCC_Dados);
    trama_length += new_buffer_size;

    if (BCC_Dados[1]!=NEUTROXOR){ // se o BCC foi afetado pelo byte stuffing então o tamanho dele aumenta em um byte
      trama_length++;
    }

    char trama[trama_length]; //array que vai guardar a trama a ser enviada pelo emissor

    //contrução da trama I a ser enviado pelo emissor
    trama[0] = FLAG;
```

```
    trama[1] = AC;
    if (Ns)
      trama[2] = I1;
    else
      trama[2] = I0;
    trama[3] = AC ^ trama[2];

    for (int i = 4,t=0; t < new_buffer_size; i++,t++){
      trama[i] = new_buffer[t];
    }

    // se o BCC foi afetado pelo byte stuffing então é necessário colocar os dois bytes dele na trama
    if (BCC_Dados[1]!=NEUTROXOR){
      trama[4+new_buffer_size] = BCC_Dados[0];
      trama[5+new_buffer_size] = BCC_Dados[1];
      trama[6+new_buffer_size] = FLAG;
    }else{
      trama[4+new_buffer_size] = BCC_Dados[0];
      trama[5+new_buffer_size] = FLAG;
    }
     sendFrame_I(fd,trama,trama_length); //envia a informação para o receptor
    printf("Sended: Ns=%i\n",Ns);
    alarm(TEMPO_ESPERA); // cria um alarme para gerar os time outs
      unsigned char readed; // variavel que vai guardar a informação envia pelo receiver byte a byte

    enum State actualState = START; // estado inicial da maquina de estados

    enum C answer ; // indica qual o tipo de resposta que o receiver enviou

    //aguardar por uma resposta do receiver
    while (actualState != STOP)
    {
      read(fd,&readed,1); // lê um byte da informação enviada pelo receiver

        if (num_tentativas > MAX_TENTATIVAS){ // se excedeu o numero maximo de tentativas
        //repor os valores standart para não afetar outras funções
        num_tentativas = 0;
        time_out = FALSE;
        printf("Error llwrite: No answer from receiver.\n");
```

```
        return -1;
      }

       if (time_out){ // se ocorer um time out o sender irá enviar novamente a
informação
        sendFrame_I(fd,trama,trama_length);
        printf("Sended: Ns=%i\n",Ns);
        alarm(TEMPO_ESPERA);
        time_out = FALSE;
      }

      // maquina de estados que valida a informação recebida
      switch (actualState)
      {
      case START:{ // recebe byte FLAG
        if (readed == FLAG)
          actualState = FLAG_RCV;
        break;
      }
      case FLAG_RCV:{ // recebe byte AC
        if (readed== AC)
          actualState = A_RCV;
        else if (readed != FLAG)
          actualState = START;
        break;
      }
      case A_RCV:{ //recebe byte RR1, RR0, REJ0, REJ1
         answer = readed; // coloca o valor de readed em answer para ser utilizada
nos estados seguintes
         if ( (readed == RR1) || (readed == RR0) || (readed == REJ0) || (readed ==
REJ1) )
          actualState = C_RCV;
        else if(readed == FLAG)
          actualState = FLAG_RCV;
        else
          actualState = START;
        break;
      }
      case C_RCV:{ // recebe byte BCC
        if (readed == (AC^answer))
          actualState = BCC_OK;
        else if(readed == FLAG)
```

```
          actualState = FLAG_RCV;
        else
          actualState = START;
        break;
      }
      case BCC_OK:{ // recebe byte FLAG
        if (readed == FLAG){
          if (Nr){
            if(answer == RR1)
              actualState = STOP;
            else if (answer == REJ0){
              num_tentativas = 0; //repor o numero de tentativas
              printf("Received: REJ0\n");
              sendFrame_I(fd,trama,trama_length);
              printf("Sended: Ns=%i\n",Ns);
              alarm(DESLIGAR_ALARME);
              alarm(TEMPO_ESPERA);
              actualState = START;
            }
          }else{
            if(answer == RR0)
              actualState = STOP;
            else if (answer == REJ1){
              num_tentativas = 0; //repor o numero de tentativas
              printf("Received: REJ1\n");
              sendFrame_I(fd,trama,trama_length);
              printf("Sended: Ns=%i\n",Ns);
              alarm(DESLIGAR_ALARME);
              alarm(TEMPO_ESPERA);
              actualState = START;
            }
          }
        }
        else
          actualState = START;
        break;
      }
      default:
        break;
      }
    }
    printf("Received: Nr=%i\n",Nr);
```

```
    //repor o numero de tentativas para não afetar outras funções;
    num_tentativas = 0;
     alarm(DESLIGAR_ALARME); //desliga o alarme porque já foi recebida a resposta
pretendida

    changeNSS(); //altera os valores de Ns e Nr
     return trama_length;
   }

   /**
   * fecha o identificador de ligação de dados
   * @param fd identificador da ligação de dados
   * @return valor 1 em caso de sucesso e -1 caso contrário
   **/
   int llclose(int fd){

    //envia DISC ao receptor
    sendFrame_S_U(fd,AC,DISC);
    printf("Sended: DISC\n");
    alarm(TEMPO_ESPERA); // cria um alarme gera os time outs

    unsigned char readed; // guarda a informação da trama recebida pelo receiver
byte a byte

    enum State actualState = START; // estado inicial da maquina de estados

    //aguardar por uma respota válida do receptor
    while (actualState != STOP)
    {
      read(fd,&readed,1); // lê um byte da informação recebida

        if (num_tentativas > MAX_TENTATIVAS){ // caso exceda o limite maximo de
tentativas
        //repor os valores standart para não afetar outras funções
        num_tentativas = 0;
        time_out = FALSE;
        printf("Error llclose: No answer from receiver, closing connection.\n");
        if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
          perror("tcsetattr");
          exit(-1);
        }
        close(fd);
```

```
        return -1;
      }

       if (time_out){ // se ocorer um time out o sender irá enviar novamente a
informação
        sendFrame_S_U(fd,AC,DISC);
        printf("Sended: DISC\n");
        alarm(TEMPO_ESPERA);
        time_out = FALSE;
      }

      // maquina de estados que valida a informação recebida
      switch (actualState)
      {
      case START:{ // recebe byte FLAG
        if (readed == FLAG)
          actualState = FLAG_RCV;
        break;
      }
      case FLAG_RCV:{ // recebe byte AR
        if (readed== AR)
          actualState = A_RCV;
        else if (readed != FLAG)
          actualState = START;
        break;
      }
      case A_RCV:{ // recebe byte DISC
        if(readed == DISC)
          actualState = C_RCV;
        else if(readed == FLAG)
          actualState = FLAG_RCV;
        else
          actualState = START;
        break;
      }
      case C_RCV:{ // recebe byte BCC
        if(readed == (AR ^ DISC))
          actualState = BCC_OK;
        else if(readed == FLAG)
          actualState = FLAG_RCV;
        else
          actualState = START;
```

```
        break;
      }
      case BCC_OK:{ // recebe byte FLAG
        if (readed == FLAG)
          actualState = STOP;
        else
          actualState = START;
        break;
      }
      default:
        break;
      }
    }
    printf("Received: DISC\n");
    alarm(DESLIGAR_ALARME); // como não ocorreu nenhum erro é necessário desligar o
alarme

    // envia a trama com a resposta UA
    sendFrame_S_U(fd,AR,UA);
    printf("Sended: UA\n");

    sleep(3); // importante para garantir que o receptor leia a trama UA antes que
o emissor fechar a ligação

    if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
      perror("tcsetattr");
      exit(-1);
    }
     printf("Closing Connection...\n");
     close(fd);

    return 1;
   }
```

receptor/app_receiver.c

```
#include "protocol_receiver.c"
#include "../app_utils.c"
#include <math.h>
/**
* funcao que verifica se o pacote recebido é o pacote END
* @param start pacote START recebida anteriormente
* @param sizeStart tamanho da pacote START
* @param end mensagem lida a verificar se é END
* @param sizeEnd tamanho dessa mesma mensagem
* @return TRUE se for END, FALSE caso contrario
**/
int isAtEnd(unsigned char *start, int sizeStart, unsigned char *end, int sizeEnd)
{
 int s = 1;
 int e = 1;
 if (sizeStart != sizeEnd) // se o tamanho do pacote a analisar e do pacote de START nao for o mesmo
   return FALSE;
 else
 {
   if (end[0] == 0x03) // se o campo de controlo do pacote atual for 3 (end)
   {
     for (; s < sizeStart; s++, e++) // percorre o pacote atual e o pacote START simultaneamente, para os comparar
     {
       if (start[s] != end[e]) // se o pacote a analisar nao for igual ao pacote de START em qualquer um dos bytes
         return FALSE;
     }
     return TRUE;
   }
   else
   {
     return FALSE;
   }
 }
}
```

```
int main(int argc, char** argv)
{
 int fd,porta; // descritor da ligacao de dados (fd) e porta serie (porta)
  unsigned char mensagemPronta[65540]; // variavel onde cada pacote vai ser guardado
 int sizeMessage = 0; // tamanho do pacote
 int sizeOfStart = 0; // tamanho do pacote START
 unsigned char start[100]; // variavel que guarda o pacote START
 struct timespec before , after; // before vai guardar o tempo de quando começa a receber informação, e after vai guardar o tempo de quando acabar de transferir a informação
  struct shell_inputs arguments; // struct que vai guardar os argumentos introduzidos
 u_int64_t before_ns,after_ns; // vai guardar os tempos com grande precisão
  //verifica se pelo menos tem 2 argumentos, e se o segundo começa com /dev/ttyS
 if ( (argc < 2) || (strncmp("/dev/ttyS",argv[1],9) != 0) ){
     printf("Invalid inputs, the template command is:\n./app /dev/ttySx [-B38400]\n");
   exit(1);
 }

 //vai buscar ao array argv os argumentos introduzidos
 if (readArgvValues(argc,argv,&arguments,RECEIVER) == -1){
   exit(-1);
 }

 //atribuição dos argumentos lidos
 BAUDRATE = arguments.baudrate;
 Delay = arguments.atraso;
 FER = arguments.FER;
 porta = atoi(&arguments.port[9]);

 fd = llopen(porta);

 // verificar se a porta serie foi aberta com sucesso
 if (fd == -1)
   exit(-1);

 sizeOfStart = llread(fd, start); // lê o pacote de controlo START

 if (sizeOfStart == -1) //se ocorreu algum erro na leitura do pacote START
   exit(-1);
```

```
    //recolher a informação do tamanho do ficheiro
    int num_blocos_tamanho = start[2];
    char tamanho_str[num_blocos_tamanho];
    for (size_t i = 0; i < num_blocos_tamanho; i++)
    {
      tamanho_str[i] = start[3+i];
    }
     int  tamanho_int  =  atoi(tamanho_str);  //converter  o  tamanho  de  string  para
inteiro

    //recolher a informação do nome do ficheiro
    int num_blocos_nome = start[3+num_blocos_tamanho+1];
    char nome_ficheiro[num_blocos_nome+1];
    for (size_t i = 0; i < num_blocos_nome; i++)
    {
      nome_ficheiro[i] = start[5 + num_blocos_tamanho + i];
    }
    nome_ficheiro[num_blocos_nome]= '\0';

    // cria ficheiro com o nome obtido no pacote START
    FILE *file = fopen(nome_ficheiro, "wb+");
     clock_gettime(CLOCK_MONOTONIC,&before);//guarda o tempo atual em before

    //enquanto houver informação para ler
    while (TRUE)
    {
      sizeMessage = llread(fd, mensagemPronta); //lê de fd um pacote de informação

      if(sizeMessage == -1) // se ocorreu algum erro no llread
        exit(-1);
      if (sizeMessage == 0)
        continue;

       if  (isAtEnd(start,  sizeOfStart,  mensagemPronta,  sizeMessage))  // se leu a
trama de END
      {
        clock_gettime(CLOCK_MONOTONIC,&after);//guarda o tempo atual em after
        printf("End message received\n");
        break;
      }
```

```
        int sizeWithoutHeader = sizeMessage - 4; //guarda o tamanho do pacote de
dados sem o cabeçalho

       // remove o cabeçalho do nível de aplicação dos pacotes de dados
       int i = 0;
       int j = 4;
          unsigned char messageRemovedHeader[sizeWithoutHeader]; //array que vai
guardar os dados recebidos
       for (; i < sizeMessage; i++, j++)
       {
         messageRemovedHeader[i] = mensagemPronta[j];
       }

       //escreve no ficheiro os dados recebidos
       fwrite(messageRemovedHeader, 1, sizeWithoutHeader, file);
     }
      //pré calculo do tempo decorrido com maior precisão
     before_ns = (before.tv_sec * 1000000000) + before.tv_nsec;
     after_ns = (after.tv_sec * 1000000000) + after.tv_nsec;

     //tempo decorrido em segundos
     double elapsedT = (after_ns - before_ns)*pow(10,-9);

     //imprime o tamanho do ficheiro e o tempo que demorou a ser transferido
     printf("File size: %d bytes\nTransfer time: %.4f sec\n", tamanho_int,elapsedT);

     //fecha o ficheiro escrito
     fclose(file);

     //fecha a ligação com a porta série
     if(llclose(fd) == -1)
       exit(-1);

     return 0;
    }
```

receptor/protocol_receiver.c

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

#include "../protocol_utils.c"

enum A { AR = 0x03 , AC = 0x01 }; // valores possiveis do campo A na trama

speed_t BAUDRATE; // baudrate que irá ser utilizado na ligação série
int esperado = 0; //numero de sequencia esperado
int Delay; //atraso introduzido no processamento dos dados no llread
int FER; //frame error ratio
struct termios oldtio,newtio; //variaveis utilizadas para guardar a configuracao
da ligacao pelo cabo serie



/**
* função que vai gerar erros na trama ou atrasos no processamento de cada trama
recebida
* @param readed variavel que contem um byte lido da trama que irá sofrer
alteração forçada para gerar erros
*/
void testEfficiency(unsigned char * readed){
 //gerar um numero de 0 a 100
 srand(time(NULL));
 int number = rand() %100;

 if (FER != 0){// se o FER não for 0
   //o readed só vai ver alterado se number tiver um valor menor que FER*100
   if (number < FER){
     *readed = 0x00;
   }
 }
```

```
    if (Delay != 0){ // se o delay não for 0
      if (number <30){
        sleep(1); //atraso de 1 segundo
      }
    }
   }

   /**
   * funcao responsavel por abir a ligação série, ler a trama SET e devolver a
trama UA
   * @param porta porta série p.e 0, 1, 10, 11
   * @return retorna o identificador de ligação
   **/
   int llopen(int porta) {
    int fd; //identificador de ligação
    char temp[3]; // guarda a porta no tipo string
    itoa(porta,temp);
    char modemDevise [9 + strlen(temp)]; // nome do modem devise p.e /dev/ttyS10
     //construção do string que contem o path do modem device
    strcpy(modemDevise,MODEMDEVICE);
    strcat(modemDevise,temp);

    fd = open(modemDevise, O_RDWR | O_NOCTTY);
    if (fd <0) {perror(modemDevise); 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");

unsigned char readed; // variavel que guarda a informacao recebida byte a byte
enum State actualState = START; // estado inicial da maquina de estados

// Logical Connection
while (actualState != STOP)
{
  read(fd,&readed,1); //le um byte da informacao recebida
  switch (actualState) // maquina de estados que valida a informacao recebida
  {
  case START:{ // recebe byte FLAG
    if (readed == FLAG)
      actualState = FLAG_RCV;
    break;
  }
  case FLAG_RCV:{ // recebe byte AR
    if (readed == AR)
      actualState = A_RCV;
    else if (readed != FLAG)
      actualState = START;
    break;
  }
  case A_RCV:{ // recebe byte SET
    if(readed == SET)
      actualState = C_RCV;
    else if(readed == FLAG)
      actualState = FLAG_RCV;
    else
      actualState = START;
    break;
  }
```

```
      case C_RCV:{ // recebe byte BCC
        if(readed == (AR ^ SET))
          actualState = BCC_OK;
        else if(readed == FLAG)
          actualState = FLAG_RCV;
        else
          actualState = START;
        break;
      }
      case BCC_OK:{ // recebe byte FLAG final
        if (readed == FLAG)
          actualState = STOP;
        else
          actualState = START;
        break;
      }
      default:
        break;
      }
    }
    printf("Received: SET\n");

    //responde ao emissor com UA
    sendFrame_S_U(fd,AR,UA);
    printf("Sended: UA\n");

    return fd;
   }

   /**
   * funcao que le as tramas de informacao e faz destuffing
   * @param fd identificador da ligacao de dados
   * @param message array que irá receber a informação transmitida pelo emissor
   * @return retorna o numero de bytes recebidos ou -1 em caso de erro
   **/
   int llread(int fd,unsigned char * message) {
    int size = 0; //tamanho da mensagem que está a ser recebida
    unsigned char c_read; // variavel para guardar o byte do campo de controlo
    int trama = 0; // variavel que varia consoante o valor de N(s) recebido
     int  mandarDados  =  FALSE;  //  variavel  que  esta  a  TRUE  quando  o  BCC  foi
corretamente recebido no final da trama
    char BCC_DADOS = NEUTROXOR; //guarda o valor do XOR entre todos os dados mais o
```

```
BCC dos dados

    unsigned char readed; // variavel que guarda a informacao recebida byte a byte
    enum State actualState = START; // estado inicial da maquina de estados

    // Recebe a trama de I(0) ou I(1)
    while (actualState != STOP)
    {
      read(fd,&readed,1); //le um byte da informacao recebida
      testEfficiency(&readed);
      switch (actualState) // maquina de estados que valida a informacao recebida
      {
      case START:{ // recebe byte FLAG
        if (readed == FLAG)
          actualState = FLAG_RCV;
        break;
      }
      case FLAG_RCV:{ // recebe byte AR
        if (readed == AR)
          actualState = A_RCV;
        else if (readed != FLAG)
          actualState = START;
        break;
      }
      case A_RCV:{ // recebe byte I0 ou I1
        if(readed == I0){ // se o numero de sequencia N(s) é 0
          actualState = C_RCV;
          c_read = readed;
          trama = 0;
        }
        else if(readed == I1){ // se o numero de sequencia N(s) é 1
          actualState = C_RCV;
          c_read = readed;
          trama = 1;
        }
        else if(readed == FLAG)
          actualState = FLAG_RCV;
        else
          actualState = START;
        break;
      }
      case C_RCV:{ // recebe byte BCC
```

```
        if(readed == (AR ^ c_read))
          actualState = BCC_OK;
        else if(readed == FLAG)
          actualState = FLAG_RCV;
        else
          actualState = START;
        break;
      }
      case BCC_OK:{
        if (readed == FLAG){ // se recebe FLAG final
           if (BCC_DADOS == 0x00) // se o BCC recebido esta correto (0x00 significa
que o XOR dos dados é igual ao BCC)
          {
            if (trama == 0){ // se N(s) foi 0
                sendFrame_S_U(fd, AR ,RR1); // responde ao emissor com confirmacao
positiva e com N(r) = 1
              printf("Sended RR, T: %d\n", trama^1);
            }else { // se N(s) foi 1
                sendFrame_S_U(fd, AR, RR0); // responde ao emissor com confirmacao
positiva e com N(r) = 0
              printf("Sended RR, T: %d\n", trama^1);
            }
            actualState = STOP;
            mandarDados = TRUE;
          }
          else // se o BCC recebido nao esta correto
          {
            if (trama == 0) // se N(s) foi 0
                sendFrame_S_U(fd, AR, REJ0); // responde ao emissor com confirmacao
negativa e com N(r) = 0
            else // se N(s) foi 1
                sendFrame_S_U(fd, AR, REJ1); // responde ao emissor com confirmacao
negativa e com N(r) = 1

            actualState = STOP;
            mandarDados = FALSE;
            printf("Sended REJ, T: %d\n", trama);
          }
        }
        else if (readed == ESC) // se recebe o octeto de escape
          actualState = ESC_STATE;
         else { // se não for nenhum dos anteriores significa que se trata de um
```

```
byte de dados
          BCC_DADOS = BCC_DADOS ^ readed;
          message[size] = readed;
          size++;
        }
        break;
      }
      case ESC_STATE:{ // recebeu octeto de escape
        if (readed == ESCFLAG) // se apos o octeto de escape, a sequencia se seguir
com 0x5e
        {
          BCC_DADOS = BCC_DADOS ^ FLAG;
          message[size] = FLAG;
          size++;
        }
        else
        {
            if (readed == ESCESC) // se apos o octeto de escape, a sequencia se
seguir com 0x5d
          {
            BCC_DADOS = BCC_DADOS ^ ESC;
            message[size] = ESC;
            size++;
          }
          else // neste caso a sequencia apos o octeto de escape nao e valida
          {
            perror("Non valid character after escape character\n");
            exit(-1);
          }
        }
         actualState = BCC_OK; // volta para o estado em que espera pela leitura da
FLAG final
        break;
      }
      default:
        break;
      }
    }
     //message tem BCC2 no fim
    size = size - 1;
    printf("Message size: %d\n", size);
     if (mandarDados) // se o BCC foi valido
```

```
    {
      if (trama == esperado)
      {
        esperado ^= 1;
      }
      else
        size = 0;
    }
    else
      size = 0;

    return size;
   }

   /**
   * funcao que le as tramas de controlo DISC, responde ao emissor com DISC, e
recebe a trama UA
   * @param fd identificador da ligacao de dados
   * @return retorna 1 em caso de sucesso e -1 caso contrário
   **/
   int llclose(int fd) {
    unsigned char readed; // variavel que guarda a informacao recebida byte a byte
    enum State actualState = START; // estado inicial da maquina de estados

    // Recebe a trama de DISC
    while (actualState != STOP)
    {
      read(fd,&readed,1); //le um byte da informacao recebida
      switch (actualState) // maquina de estados que valida a informacao recebida
      {
      case START:{ // recebe byte FLAG
        if (readed == FLAG)
          actualState = FLAG_RCV;
        break;
      }
      case FLAG_RCV:{ // recebe byte AR
        if (readed == AR)
          actualState = A_RCV;
        else if (readed != FLAG)
          actualState = START;
        break;
      }
```

```
      case A_RCV:{ // recebe byte DISC
        if(readed == DISC)
          actualState = C_RCV;
        else if(readed == FLAG)
          actualState = FLAG_RCV;
        else
          actualState = START;
        break;
      }
      case C_RCV:{ // recebe byte BCC
        if(readed == (AR ^ DISC))
          actualState = BCC_OK;
        else if(readed == FLAG)
          actualState = FLAG_RCV;
        else
          actualState = START;
        break;
      }
      case BCC_OK:{ // recebe byte FLAG final
        if (readed == FLAG)
          actualState = STOP;
        else
          actualState = START;
        break;
      }
      default:
        break;
      }
    }
    printf("Received: DISC\n");

    // responde DISC ao emissor
    sendFrame_S_U(fd,AC,DISC);
    printf("Sended: DISC\n");

    // efetua a leitura da trama UA enviada de volta pelo emissor
    actualState = START; // estado inicial

    // Recebe a trama de UA
    while (actualState != STOP)
    {
      read(fd,&readed,1); //le um byte da informacao recebida
```

```
switch (actualState) // maquina de estados que valida a informacao recebida
{
case START:{ // recebe byte FLAG
  if (readed == FLAG)
    actualState = FLAG_RCV;
  break;
}
case FLAG_RCV:{ // recebe byte AC
  if (readed == AC)
    actualState = A_RCV;
  else if (readed != FLAG)
    actualState = START;
  break;
}
case A_RCV:{ // recebe byte UA
  if(readed == UA)
    actualState = C_RCV;
  else if(readed == FLAG)
    actualState = FLAG_RCV;
  else
    actualState = START;
  break;
}
case C_RCV:{ // recebe byte BCC
  if(readed == (AC ^ UA))
    actualState = BCC_OK;
  else if(readed == FLAG)
    actualState = FLAG_RCV;
  else
    actualState = START;
  break;
}
case BCC_OK:{ // recebe byte FLAG final
  if (readed == FLAG)
    actualState = STOP;
  else
    actualState = START;
  break;
}
default:
  break;
```

```
    }
  }

  printf("Received: UA\n");
   if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
    perror("tcsetattr");
    exit(-1);
  }
   printf("Receiver terminated\n");
   close(fd);

  return 1;
}
```

Anexo II

```
   //guarda os parametros passados pela shell
   struct shell_inputs
   {
    char port[15]; //Dispositivo /dev/ttySx, x= 0,1,10,11
    char file_name[200]; //nome do ficheiro a enviar
    int frame_size; //tamanho maximo da quantidade de dados que devem ser enviados de uma vez
    speed_t baudrate; // baudrate da ligação serie
   };
```

Anexo III

Envio do pacote START

```
   //construção do pacote de controlo de START
          controlo[0] = START;
          controlo[1] = TAMANHO;

             //passar o tamanho do ficheiro para string e dps adicionar ao array controlo
          char size[10];
          sprintf(size,"%d",tamanho);
           int num_blocos_tamanho = strlen(size); //guarda o numero de blocos de 1 byte que são necessários para guardar a informação do tamanho do ficheiro
          controlo[2] = num_blocos_tamanho;
          tamanho_controlo += num_blocos_tamanho;

            controlo = (char * )realloc(controlo,tamanho_controlo); //alocar mais memoria para ser possivel guardar os blocos que correspondem ao tamanho do ficheiro

          memcpy(controlo+3,size,num_blocos_tamanho);

          //adicionar ao pacote de controlo a informação do nome do ficheiro
          controlo[3+num_blocos_tamanho] = NOME;
          controlo[4+num_blocos_tamanho] = strlen(nome_ficheiro);
```

```
            tamanho_controlo += strlen(nome_ficheiro);
               controlo=  (char  *  )  realloc  (controlo,tamanho_controlo);//necessário
alocar mais memória para guardar o nome do ficheiro

memcpy(controlo+5+num_blocos_tamanho,nome_ficheiro,strlen(nome_ficheiro));

            //envia o pacote de controlo START ao receptor
            int bytesEscritos = llwrite(fd, controlo,tamanho_controlo);
```

isAtEnd

```
    /**
    * funcao que verifica se o pacote recebido é o pacote END
    * @param start pacote START recebida anteriormente
    * @param sizeStart tamanho da pacote START
    * @param end mensagem lida a verificar se é END
    * @param sizeEnd tamanho dessa mesma mensagem
    * @return TRUE se for END, FALSE caso contrario
    **/
    int isAtEnd(unsigned char *start, int sizeStart, unsigned char *end, int
sizeEnd)
    {
     int s = 1;
     int e = 1;
     if (sizeStart != sizeEnd) // se o tamanho do pacote a analisar e do pacote de
START nao for o mesmo
       return FALSE;
     else
     {
       if (end[0] == 0x03) // se o campo de controlo do pacote atual for 3 (end)
       {
         for (; s < sizeStart; s++, e++) // percorre o pacote atual e o pacote START
simultaneamente, para os comparar
         {
           if (start[s] != end[e]) // se o pacote a analisar nao for igual ao pacote
de START em qualquer um dos bytes
             return FALSE;
         }
         return TRUE;
       }
       else
```

```
      {
        return FALSE;
      }
    }
  }
```

Tratamento dos pacotes a enviar

```
//construção do pacote de dados
          dados[1] = num_sequencia;
          dados[2] = bytesLidos/255;
          dados[3] = bytesLidos%255;

            dados = (char * )realloc(dados,tamanho_dados + bytesLidos); //adiciona
mais espaço ao array para poder guardar a informação do ficheiro

          for (size_t i = 0; i < bytesLidos; i++)
          {
              dados[4+i] = buffer[i];
          }
```

Criação do ficheiro do lado do receptor, a partir do pacote de controlo START lido

```
  //recolher a informação do tamanho do ficheiro
    int num_blocos_tamanho = start[2];
    char tamanho_str[num_blocos_tamanho];
    for (size_t i = 0; i < num_blocos_tamanho; i++)
    {
      tamanho_str[i] = start[3+i];
    }
     int  tamanho_int  =  atoi(tamanho_str);  //converter  o  tamanho  de  string  para
inteiro

    //recolher a informação do nome do ficheiro
    int num_blocos_nome = start[3+num_blocos_tamanho+1];
    char nome_ficheiro[num_blocos_nome+1];
    for (size_t i = 0; i < num_blocos_nome; i++)
    {
      nome_ficheiro[i] = start[5 + num_blocos_tamanho + i];
```

```
    }
    nome_ficheiro[num_blocos_nome]= '\0';

    // cria ficheiro com o nome obtido no pacote START
    FILE *file = fopen(nome_ficheiro, "wb+");
```

Anexo IV

Variação da capacidade da ligação				
S =(R/C)	BaudRate	R	Tempo Decorrido (sec)	Tamanho do Ficheiro (bits)
0,7682	4800	3687,559	23,7946	87744
0,7674	9600	7367,254	11,91	87744
0,7692	19200	14769,48	5,9409	87744
0,7692	38400	29536,47	2,9707	87744
0,7728	57600	44510,73	1,9713	87744
0,7616	115200	87735,23	1,0001	87744

Variação do tamanho das tramas I					
S =(R/C)	BaudRate	R	Tempo Decorrido (sec)	Tamanho do Ficheiro (bits)	Tamanho do Pacote s/ cabeçalho (bytes)
0,75873	38400	29135,3	3,0116	87744	1024
0,76918	38400	29536,5	2,9707	87744	5000
0,77048	38400	29586,3	2,9657	87744	10000
0,77386	38400	29716,1	86,5365	2571528	20000
0,77413	38400	29726,7	86,5056	2571528	30000
0,77422	38400	29730,2	86,4955	2571528	40000
0,77432	38400	29733,7	86,4853	2571528	50000
0,77436	38400	29735,5	86,48	2571528	60000
0,77440	38400	29737,1	86,4755	2571528	65535

Variação do FER					
S =(R/C)	BaudRate	R	Tempo Decorrido (sec)	Tamanho do Ficheiro (bits)	FER (%)
0,7587	38400	29134,4	3,0117	87744	0%
0,2759	38400	10595,3	8,2814	87744	20%
0,1756	38400	6742,07	13,0144	87744	30%
0,1268	38400	4870,77	18,0144	87744	40%
0,0981	38400	3768,25	23,2851	87744	50%
0,0686	38400	2635,93	33,2877	87744	60%

Variação do T_prop					
S =(R/C)	BaudRate	R	Tempo Decorrido (sec)	Tamanho do Ficheiro (bits)	T_prop(sec)
0,7448	38400	28601,6	3,0678	87744	0
0,5682	38400	21819,81	4,0213	87744	1
0,4603	38400	17673,93	4,9646	87744	2
0,3895	38400	14957,3	5,8663	87744	3
0,3318	38400	12742,93	6,8857	87744	4
0,2614	38400	10038,21	8,741	87744	5