Redes de Computadores

Protocolo de Ligação de Dados

Mestrado Integrado em Engenharia Informática e Computação

Gonçalo Teixeira up201806562@fe.up.pt

Gonçalo Alves up201806451@fe.up.pt

24 de outubro de 2020

Índice

Sumário

Este relatório foi elaborado no âmbito da unidade curricular de Redes e Computadores, e trata-se da implementação de um protocolo de transferência de dados. O trabalho consiste no desenvolvimento de uma aplicação capaz de transferir ficheiros de um computador para o outro através de uma porta série.

O trabalho foi concluído e a aplicação desenvolvida é capaz de transferir ficheiros sem perda de dados.

Introdução

O objetivo deste trabalho consistiu na implementação de um protocolo de ligação de dados, de acordo com o guião fornecido, e no teste do dito protocolo, através de uma aplicação de transferência de dados. Quanto ao relatório, o seu objetivo é detalhar a componente teórica do trabalho, com a estrutura descrita em baixo:

- **Arquitetura**
  - Descrição dos blocos funcionais e das interfaces implementadas
- **Estrutura do código**
  - Descrição das APIs, principais estruturas de dados e funções e as suas relações com a arquitetura
- **Casos de uso principais**
  - Identificação dos casos de uso
  - Sequências de chamada de funções
- **Protocolo de ligação lógica**
  - Identificação dos principais aspetos funcionais
  - Descrição da estratégia de implementação destes, com a exibição de extratos de código

- **Protocolo de aplicação**
    - Identificação dos principais aspetos funcionais
    - Descrição da estratégia de implementação destes, com a exibição de extratos de código
- **Validação**
    - Descrição dos testes efetuados
    - Apresentação quantificada dos resultados
- **Eficiência do protocolo de ligação de dados**
    - Caraterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido
- **Conclusão**
    - Síntese da informação apresentada nas secções anteriores
    - Reflexão sobre os objetivos de aprendizagem alcançados.

Arquitetura

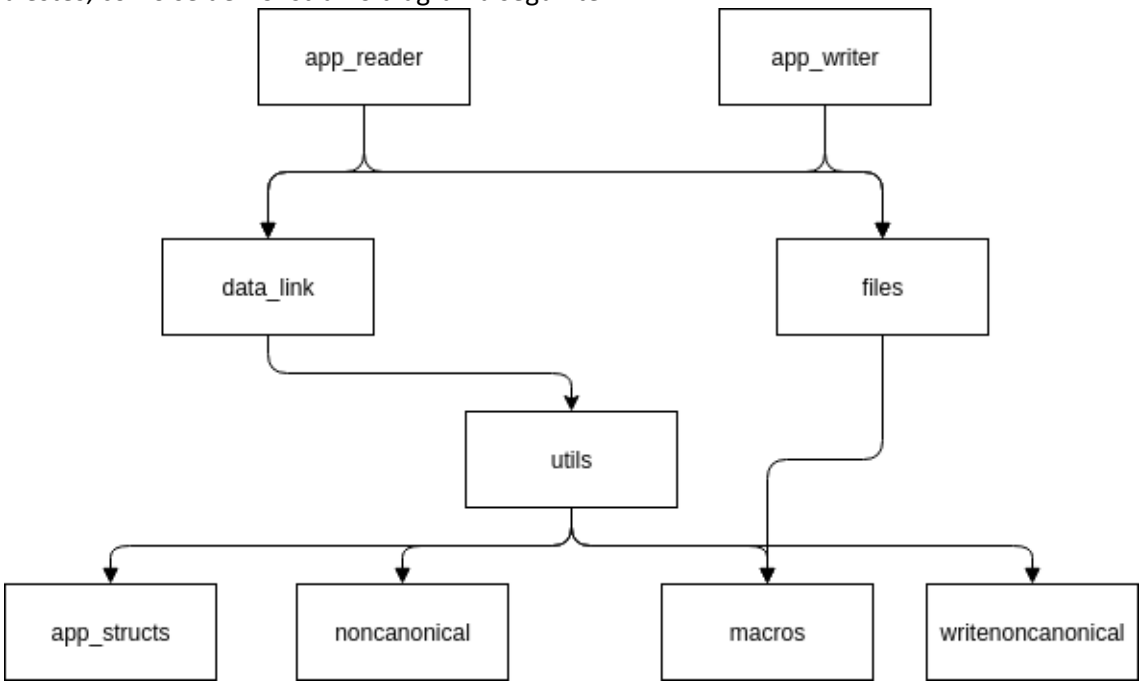
O trabalho está dividido em duas camadas: a camada do **de ligação de dados** e a camada da **aplicação**.

A camada de ligação de dados é responsável pelas interações com a porta de série, tal como: a abertura, o fecho, a leitura e a escrita desta. Além disso, a camada também é responsável pelo tratamento das tramas: delineação, *stuffing*, proteção e retransmissão destas.

A camada da aplicação é responsável pelo envio e receção de ficheiros, fazendo uso da interface da camada de ligação de dados. Além disso, a camada também é responsável pelo processamento do serviço: tratamento de cabeçalhos, distinção entre pacotes de controlo e de dados e numeração destes.

Estrutura de código

O código está dividido em diversos ficheiros de código, tendo uma hierarquia associada a estes, como se demonstra no diagrama seguinte:



Assim, a estrutura do código será feita de forma hierárquica, de baixo para cima.

Writenoncanonical

- **atende** – *Handler* do sinal **SIGALRM**;
- **open_writer** – Abre a porta série do emissor e implementa o *handler* do alarme;
- **close_writer** – Fecha a porta série do emissor;

Noncanonical

- **open_reader** – Abre a porta série do recetor;
- **close_reader** – Fecha a porta série do recetor;

Macros

O ficheiro macros.h contém, tal como o nome indica, macros importantes para o programa, entre as quais:

- **TRIES** – número de tentativas de escrita;
- **TIMEOUT** – número de segundos de espera por resposta;

App_structs

O ficheiro app_structs.h contém as estruturas de dados relevantes para o programa:

- **information_frame_t** – *Frame* contendo: *address*, *control*, *bcc*, *data*, *data_size*, *bcc2* e *raw_bytes*;
- **data_packet_t** – Pacote de dados contendo: *control*, *sequence*, *data_field_size*, *data*, *raw_bytes* e *raw_bytes_size*;
- **control_packet_t** – Pacote de dados contendo: *control*, *file_size*, *file_name*, *filesize_size*, *raw_bytes* e *raw_bytes_size*;
- **file_t** – Ficheiro contendo: *data*, *name* e *size*;

Utils

- **parse_control_packet** – Função que faz o *parse* de bytes para a estrutura **control_packet_t**;
- **parse_data_packet** – Função que faz o *parse* de bytes para a estrutura **data_packet_t**;
- **print_control_packet –** Função que imprime um pacote de controlo;
- **print_data_packet –** Função que imprime um pacote de dados. Além disso, permite mostrar todos os bytes de dados, através do uso de uma flag;
- **print_elapsed_time –** Função que imprime o tempo passado entre uma escrita e uma leitura;
- **verify_message –** Função que verifica erros numa trama de Informação, através do bcc;
- **print_message –** Função que imprime uma trama de Informação;
- **array_to_number–** Função que transforma um *array* num número de 8 bytes;
- **number_to_array–** Função que transforma um número de 8 bytes num *array*;
- **generateErrorBCC2** – Função que gera um byte anormal e insere-o num pacote;
- **generateErrorBCC1** – Função que gera um byte anormal e substitui-o pelo byte de flag ou de endereço;
- **generateDelay** – Função que gera um atraso;
- **generate_control_packet –** Função que cria um pacote de controlo;
- **generate_data_packet –** Função que cria um pacote de dados;

Data_link

- **send_supervision_frame** – Recebe uma porta e envia-lhe uma mensagem de controlo;
- **receive_supervision_frame** – Recebe uma porta e lê uma mensagem de controlo dela;
- **receive_acknowledgment** – Recebe uma mensagem de **ACK** e retorna o seu *byte* de controlo;
- **send_acknowledgment** – Envia uma mensagem de **ACK**;
- **receive_set** – Função que espera receber uma mensagem **SET** e envia **UA** de seguida;
- **send_set** – Função que envia uma mensagem **SET** e espera receber **UA** de seguida;
- **disconnect_writer** – Função que envia uma mensagem de **DISC**, espera receber **DISC** de volta e envia **UA**;
- **disconnect_reader** - Função que recebe uma mensagem de **DISC**, espera enviar **DISC** de volta e recebe **UA**;
- **llopen** – Função que abre a porta série do emissor/recetor e inicia a ligação de dados, retornando o descritor correspondente;
- **llclose** – Função que fecha a porta série do emissor/recetor e termina a ligação de dados;
- **llwrite** – Função que faz o *stuffing* de um pacote de dados e envia para o recetor, esperando receber uma mensagem de **ACK** de volta e procede de acordo com esta;
- **llread** – Função que lê uma mensagem do emissor, faz o *destuffing*, verifica a mensagem e envia uma mensagem **ACK** adequada;

Files

- **get_file_size –** Função que calcula o tamanho de um ficheiro, em bytes;
- **read_file –** Função que lê os dados de um ficheiro;
- **split_file –** Função que obtém bytes de um ficheiro entre um índice inicial e final, inclusive;
- **join_file –** Função que concatena pacotes;
- **write_file –** Função que cria uma cópia do ficheiro recebido;

App_writer

- **main –** Função responsável pela escrita de um ficheiro;

App_reader

- **main –** Função responsável pela leitura de um ficheiro;

Casos de uso principais

Os casos de uso principais da aplicação são: a interface, que permite a escolha do ficheiro que o emissor pretende enviar, e a transferência do ficheiro, através da porta série. De modo a compilar o programa, é necessário executar `make` na pasta "src".

De modo a dar-se a transferência do ficheiro, o utilizador necessitará de introduzir o número da porta série a ser utilizada, como por exemplo **11**. Adicionalmente, caso se trate do emissor, também terá de inserir o ficheiro a ser enviado, como por exemplo **pinguim.gif.** Ex: Em duas consolas: (1ª - emissor) **"`./app_writer 11 pinguim.gif`"** / (2ª-recetor) **"`./app_reader 10`"**

A transmissão de dados dá-se pela seguinte ordem:

- Abertura da ligação entre os computadores;
- Geração dos pacotes **START** e **STOP**, para controlo;
- Escrita do pacote **START**;
- Escrita dos pacotes de dados;
- Escrita do pacote **STOP**;
- Fecho da ligação entre os computadores;

A receção de dados dá-se pela seguinte ordem:

- Abertura da ligação entre os computadores;
- Leitura e impressão do pacote **START**;
- Leitura e impressão dos pacotes de dados;
- Leitura e impressão do pacote **STOP**;
- Impressão da mensagem completa;
- Fecho da ligação entre os computadores;

Protocolo de ligação lógica

llopen

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

No caso do emissor, a porta série é aberta, é enviada uma mensagem **SET** e é esperada uma mensagem **UA**.

No caso do recetor, a porta de série é aberta, é recebida uma mensagem **SET** e é enviada uma mensagem **UA**.

llclose

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

No caso do emissor, é enviada uma mensagem **DISC**, posteriormente é recebida uma mensagem igual de volta, é enviada uma mensagem **UA** e a ligação termina.

No caso do recetor, é recebida uma mensagem **DISC**, de seguida é enviada essa mesma mensagem para o recetor, é esperada uma mensagem **UA** e a ligação é terminada.

llwrite

Esta função é responsável pelo envio de tramas.

Inicialmente, é composto o *header* da mensagem: *address*, *control* e *bcc1*. De seguida, é feito o *stuffing* da mensagem e a construção do *bcc2* (também com *stuffing*). Finalmente, a função irá enviar a mensagem completa para o emissor e esperar pela mensagem **ACK**. Dependendo desta, o emissor poderá: continuar a transmissão do ficheiro, passando para o próximo pacote; retransmitir o pacote acabado de enviar, devido a um erro. A retransmissão de um pacote também se pode dar quando o tempo de espera de uma resposta exceder o tempo máximo de espera, **TIMEOUT**.

llread
Esta é função é responsável pela receção de tramas.

Inicialmente, é feita uma leitura da porta série, caractere a caractere. De seguida, é feito o *destuffing* da mensagem e esta é guardada numa estrutura de dados (**information_frame_t**). Finalmente, é feita uma verificação de erros e, dependendo do resultado desta, é enviada a mensagem adequada para o emissor.

Protocolo de aplicação
O protoloco de aplicação implementado tem como aspetos principais:

- Envio de pacotes de controlo **START** e **STOP**. Estes contêm o nome e o tamanho do ficheiro a ser enviado;
- Divisão do ficheiro em pacotes, no emissor, e a concatenação dos pacotes recebidos, no recetor;
- Encapsulamento de cada pacote de dados com um *header* contendo o número de sequência do pacote e o tamanho do pacote;
- Leitura do ficheiro a enviar, no emissor, e criação do ficheiro, no recetor.

Estas funcionalidades foram implementadas usando funções descritas a seguir.

generate_control_packet
Esta função retorna um pacote **START** ou **STOP**, recebendo como argumento um inteiro de modo a identificar o tipo de pacote. Estes pacotes serão enviados usando a função **llwrite**, pertencente ao protocolo de ligação de dados.

generate_data_packet
Esta função retorna um pacote de dados, recebendo como argumentos: dados de um ficheiro, tamanho dos dados e o número de sequência. Estes pacotes serão enviados usando a função **llwrite**, pertencente ao protocolo de ligação de dados.

split_file
Esta função retorna dados, recebendo como argumentos: o ficheiro de onde se quer obter os dados, o índice do primeiro byte a recolher e o índice do último byte a receber. Estes dados serão usados na função acima mencionada.

get_file_size
Esta função retorna o tamanho de um ficheiro, recebendo como argumento o descritor de um ficheiro.

read_file
Esta função retorna os dados de um ficheiro, recebendo como como argumentos: o descritor de um ficheiro e o tamanho deste.

join_file
Esta função recebe como argumentos: um *array* onde se vão concatenar os pacotes, um pacote, o tamanho do pacote e o índice dos dados.

write_file
Esta função cria uma cópia do ficheiro recebido, recebendo com argumentos: o nome do ficheiro, os *bytes* deste e o seu tamanho.

Validação

Foram realizados os seguintes testes, todos com sucesso:

- Envio de ficheiros de vários tamanhos;
- Envio de um ficheiro com pacotes de vários tamanhos;
- Interrupção da ligação enquanto um ficheiro é enviado;
- Envio de um ficheiro com variação do *baudrate;*

Eficiência do protocolo de ligação de dados

Os testes não foram efetuados nos computadores da FEUP e sim nos nossos computadores pessoais, usando uma porta de série virtual. Como tal, os resultados podem variar. Além disso, os resultados dos testes encontram-se no Anexo II.

Variação do FER

Com o aumento de erros, o programa torna-se menos eficiente.

Variação do tamanho das tramas I

Com o aumento do tamanho da trama I, o programa torna-se mais eficiente.

Variação da capacidade da ligação (C)

Com o aumento da *Baudrate,* o programa torna-se menos eficiente.

Conclusões

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

Adicionalmente, foi nos dado a conhecer a **independência entre camadas**, e cada um dos blocos funcionais da arquitetura da aplicação desenvolvida cumpre esta independência.

Na camada de ligação de dados não é feito qualquer processamento que incida sobre o cabeçalho dos pacotes a transportar em tramas de Informação. Ao nível da ligação de dados não existe qualquer distinção entre pacotes de controlo e de dados, nem é relevante (nem tida em conta) a numeração dos pacotes de dados.

Na camada de aplicação não são conhecidos os detalhes do protocolo de ligação de dados, mas apenas a forma como se acede ao serviço. O protocolo de aplicação desconhece a estrutura das tramas e o respetivo mecanismo de delineação, a existência de *stuffing* (e qual a opção adotada), o mecanismo de proteção das tramas, eventuais retransmissões de tramas de Informação, etc.

Concluindo, o protocolo de ligação de dados foi realizado com sucesso, tendo-se cumprido todos os objetivos, e o desenvolvimento deste contribuiu para um aprofundamento do conhecimento, tanto teórico como prático, deste tema.

Infelizmente, devido à situação atual, as condições de desenvolvimento e teste do trabalho não foram as melhores.

app_reader.c

```
#include "data_link.h"
#include "files.h"

file_t file;

int main(int argc, char *argv[]) {
  if (argc != 2) {
    printf("Usage: %s <number for serial port>\n", argv[0]);
    printf("\nExample: %s 11\t-\tfor '/dev/ttyS11'\n", argv[0]);
    exit(ERROR);
  }

  /* opens transmiter file descriptor on second layer */
  int receiver_fd = llopen(atoi(argv[1]), RECEIVER);
  /* in case there's an error oppening the port */
  if (receiver_fd == ERROR) {
    exit(ERROR);
  }

  char buffer[2*PACKET_SIZE];
  int size;

  int state = 0;

  printf("Packet Size: %d\n", PACKET_SIZE);

  // * START Control Packet
  while (state == 0) {
    memset(buffer, 0, sizeof(buffer));
    while ((size = llread(receiver_fd, buffer)) == ERROR) {
      printf("Error reading\n");
      llclose(receiver_fd, RECEIVER);
      return ERROR;
    }
    control_packet_t packet = parse_control_packet(buffer,
size);

    file.size = array_to_number(packet.file_size,
packet.filesize_size);
    file.name = packet.file_name;

    print_control_packet(&packet);
    if (packet.control == START) {
      state = 1;
    }
  }

  // * DATA Packets
  unsigned char *full_message = (unsigned char*) malloc
(file.size);
  int index = 0;
  int current_sequence = -1;
```

```
  while (state == 1) {
    memset(buffer, 0, sizeof(buffer));
    while ((size = llread(receiver_fd, buffer)) == ERROR) {
      printf("Error reading\n");
    }
    if (buffer[0] == STOP) {
      state = 2;
      break;
    }
    data_packet_t data = parse_data_packet(buffer, size);

    if (data.control != DATA) continue;

    print_data_packet(&data, FALSE);
    join_file(full_message, data.data, data.data_field_size,
index);

    // * caso o numero de sequencia seja diferente do anterior
deve atualizar o index
    if (current_sequence != data.sequence) {
      current_sequence = data.sequence;
      index += data.data_field_size;
    }
  }

  // * STOP Control Packet
  if (state == 2) {
    control_packet_t packet = parse_control_packet(buffer,
size);
    print_control_packet(&packet);

    char* name = (char*) malloc ((strlen(file.name) + 7) *
sizeof(char));
    sprintf(name, "cloned_%s", file.name);
    write_file(name, full_message, file.size);
    printf("Received file\n");
  }

  /* resets and closes the receiver fd for the port */
  llclose(receiver_fd, RECEIVER);

  return 0;
}
```

app_structs.h

```
/**
 * \addtogroup APPLICATIONLAYER
 * @{
 */

/**
 * @brief Information Frame Struct
 * This struct stores the bytes from an information frame
 */
typedef struct {
```

```
  unsigned char address;     /**< @brief The address byte */
  unsigned char control;     /**< @brief The control byte */
  unsigned char bcc1;        /**< @brief The BCC1 byte */
  unsigned char *data;       /**< @brief The data array */
  int data_size;             /**< @brief The data array size in
bytes */
  unsigned char bcc2;        /**< @brief The BCC2 byte */

  unsigned char *raw_bytes; /**< @brief The array containing
unprocessed bytes */
} information_frame_t;

/**
 * @brief Data Packet Struct
 * This struct stores the bytes from a data packet
 */
typedef struct {
  unsigned char control;     /**< @brief The control byte -
[DATA] */
  unsigned char sequence;    /**< @brief The sequence byte -
index on global data */
  int data_field_size;       /**< @brief The size of the data
array - [1..PACKET_SIZE] */
  unsigned char data[2*PACKET_SIZE]; /**< @brief The data array
*/

  unsigned char *raw_bytes; /**< @brief The array containing
unprocessed bytes */
  int raw_bytes_size;        /**< @brief The size of the
raw_bytes array */
} data_packet_t;

/**
 * @brief Control Packet Struct
 * This struct stores the bytes from a control packet
 */
typedef struct {
  unsigned char control;       /**< @brief The control byte -
[START; STOP] */
  unsigned char *file_size;    /**< @brief File size in bytes
stored in an array */
  unsigned char *file_name;    /**< @brief String for the
filename */
  unsigned int filesize_size; /**< @brief Size of the file_size
array */

  unsigned char *raw_bytes;    /**< @brief The array containing
unprocessed bytes */
  int raw_bytes_size;          /**< @brief The size of the
raw_bytes array */
} control_packet_t;

/**
 * @brief File information Struct
 * This struct stores the file's information
 */
```

```
typedef struct {
  unsigned char* data;  /**< @brief Data array, containing all
the bytes from the file */
  unsigned char* name;  /**< @brief Name of the file */
  unsigned long size;   /**< @brief Size of the file in bytes */
} file_t;

/** @} */
```

app_writer.c

```
#include "data_link.h"
#include "files.h"

extern int flag;

int main(int argc, char *argv[]) {
  if (argc != 3) {
    printf("Usage: %s <number for serial port> <filename>\n",
argv[0]);
    printf("\nExample: %s 11\t-\tfor '/dev/ttyS11'
pinguim.gif\n", argv[0]);
    return -1;
  }

  FILE *fp;
  file_t file;

  /* opens transmiter file descriptor on second layer */
  int transmiter_fd = llopen(atoi(argv[1]), TRANSMITTER);
  /* in case there's an error oppening the port */
  if (transmiter_fd == ERROR) {
    exit(ERROR);
  }

  char* filename = argv[2];

  fp = fopen(filename, "rb");
  file.name = filename;
  file.size = get_file_size(fp);
  file.data = read_file(fp, file.size);

  control_packet_t c_packet_start =
generate_control_packet(START, &file);
  control_packet_t c_packet_stop = generate_control_packet(STOP,
&file);

  // sending control packet
  struct timespec start;
  clock_gettime(CLOCK_MONOTONIC_RAW, &start);

  print_control_packet(&c_packet_start);

  int size = llwrite(transmiter_fd, c_packet_start.raw_bytes,
c_packet_start.raw_bytes_size);
  if (size == ERROR) {
```

```
    printf("Error writing START Control Packet, aborting...\n");
    llclose(transmiter_fd, TRANSMITTER);
    free(c_packet_start.file_size);
    free(c_packet_start.raw_bytes);
    free(c_packet_stop.file_size);
    free(c_packet_stop.raw_bytes);
    free(file.data);
    return ERROR;
  }
  free(c_packet_start.file_size);
  free(c_packet_start.raw_bytes);

  unsigned long bytes_left = file.size;
  int index_start;
  int index_end = -1;
  int sequence = 0;
  while (bytes_left != 0 && index_end != file.size - 1) {
    usleep(STOP_AND_WAIT);

    index_start = index_end + 1;
    if (bytes_left >= PACKET_SIZE-1) {
      index_end = index_start + PACKET_SIZE-1;
    } else {
      index_end = index_start + bytes_left - 1;
    }
    bytes_left -= (index_end - index_start) + 1;

    unsigned char* frame = split_file(file.data, index_start,
index_end);
    data_packet_t data = generate_data_packet(frame, index_end -
index_start + 1, sequence++);
    //print_data_packet(&data, TRUE);

    printProgressBar(file.size-bytes_left,file.size);
    /* caso o write não seja bem sucedido tentar de novo */
    while ((size = llwrite(transmiter_fd, data.raw_bytes,
data.raw_bytes_size)) == ERROR) {
      usleep(STOP_AND_WAIT);
    }
    clearProgressBar();

    free(frame);
    free(data.raw_bytes);
    if (size == ERROR) {
      printf("Error writing Data Packet, aborting...\n");
      llclose(transmiter_fd, TRANSMITTER);
      return ERROR;
    }
  }
  printProgressBar(1,1);

  usleep(STOP_AND_WAIT);
  print_control_packet(&c_packet_stop);
  size = llwrite(transmiter_fd, c_packet_stop.raw_bytes,
c_packet_stop.raw_bytes_size);
  if (size == ERROR) {
```

```
    printf("Error writing STOP Control Packet, aborting...\n");
    llclose(transmiter_fd, TRANSMITTER);
    return ERROR;
  }
  free(c_packet_stop.file_size);
  free(c_packet_stop.raw_bytes);

  usleep(STOP_AND_WAIT);
  print_elapsed_time(start);
  /* resets and closes the receiver fd for the port */
  llclose(transmiter_fd, TRANSMITTER);

  return OK;
}
```

data_link.c

```
#include "data_link.h"

int reetransmit = 1;
extern int flag;
extern int conta;

int send_supervision_frame(int fd, unsigned char msg, unsigned
char address) {
  unsigned char mesh[5];
  mesh[0] = FLAG;
  mesh[1] = address;
  mesh[2] = msg;
  mesh[3] = mesh[1] ^ mesh[2];
  mesh[4] = FLAG;
  int err = write(fd, mesh, 5);
  if (!(err == 5))
    return ERROR;
  return OK;
}

unsigned char receive_acknowledgement(int fd) {
  // ! Remove comments if you want to debug the data being read
  int part = 0;
  unsigned char rcv_msg;
  unsigned char ctrl;
  unsigned char address;
  //printf("Reading ACK supervision frame...\n");
  while (part != 5) {
    int rd = read(fd, &rcv_msg, 1);
    if (rd == -1 && errno == EINTR) {
      //printf("READ failed\n");
      return 2;
    }
    switch (part) {
    case 0:
      if (rcv_msg == FLAG) {
        part = 1;
        // printf("FLAG: 0x%x\n",rcv_msg);
      }
      break;
```

```
    case 1:
      if (rcv_msg == A_1 || rcv_msg == A_3) {
        address = rcv_msg;
        part = 2;
        // printf("A: 0x%x\n",rcv_msg);
      } else {
        if (rcv_msg == FLAG)
          part = 1;
        else
          part = 0;
      }
      break;
    case 2:
      if ((rcv_msg == C_RR0) || (rcv_msg == C_RR1) || (rcv_msg
== C_REJ0) ||
          (rcv_msg == C_REJ1)) {
        part = 3;
        // printf("Control: 0x%x\n",rcv_msg);
        ctrl = rcv_msg;
      } else
        part = 0;
      break;
    case 3:
      if (rcv_msg == (address ^ ctrl)) {
        part = 4;
        // printf("Control BCC: 0x%x\n",rcv_msg);
      } else
        part = 0;
      break;
    case 4:
      if (rcv_msg == FLAG) {
        part = 5;
        // printf("FINAL FLAG: 0x%x\nReceived
Control\n",rcv_msg);
      } else
        part = 0;
      break;
    default:
      break;
    }
  }
  return ctrl;
}

int receive_supervision_frame(int fd, unsigned char msg) {
  // ! Remove comments if you want to debug the data being read
  int part = 0;
  unsigned char rcv_msg;
  unsigned char address;

  printf("Reading supervision frame...\n");
  while (part != 5) {
    int rd = read(fd, &rcv_msg,1);
    if (rd == -1 && errno == EINTR) {
      printf("READ failed\n");
      return 2;
```

```
    }
    switch (part) {
    case 0:
      if (rcv_msg == FLAG) {
        part = 1;
        // printf("FLAG: 0x%x\n",rcv_msg);
      }
      break;
    case 1:
      if (rcv_msg == A_1 || rcv_msg == A_3) {
        address = rcv_msg;
        part = 2;
        // printf("A: 0x%x\n",rcv_msg);
      } else {
        if (rcv_msg == FLAG)
          part = 1;
        else
          part = 0;
      }
      break;
    case 2:
      if (rcv_msg == msg) {
        part = 3;
        // printf("Control: 0x%x\n",rcv_msg);
      } else
        part = 0;
      break;
    case 3:
      if (rcv_msg == (address ^ msg)) {
        part = 4;
        // printf("Control BCC: 0x%x\n",rcv_msg);
      } else
        part = 0;
      break;
    case 4:
      if (rcv_msg == FLAG) {
        part = 5;
        // printf("FINAL FLAG: 0x%x\nReceived
Control\n",rcv_msg);
      } else
        part = 0;
      break;
    default:
      break;
    }
  }
  return (part == 5) ? 0 : -1;
}

int receive_set(int fd) {
  if (receive_supervision_frame(fd, SET) == 0) {
    printf("Sending UA reply...\n");
    send_supervision_frame(fd, UA, A_3);
  }
  return 0;
}
```

```c
int send_set(int fd) {
  //struct timespec start;
  do {
    //clock_gettime(CLOCK_MONOTONIC_RAW, &start);
    if (send_supervision_frame(fd, SET, A_3) == -1) {
      printf("Error writing SET\n");
      continue;
    }
    alarm(TIMEOUT); // activa alarme de 3s
    printf("Sent SET frame\n");
    flag = 0;
    printf("Receiving UA response...\n");
    while (!flag) {
      if (receive_supervision_frame(fd, UA) == 0) {
        reetransmit = 0;
        break;
      }
    }

    if (flag)
      printf("Timed Out - Retrying\n");
    //print_elapsed_time(start);

  } while (conta < 4 && flag);

  alarm(RESET_ALARM);

  if (conta == 4) {
    reetransmit = 2;
    printf("Gave up\n");
    return -1;
  }

  return 0;
}

int disconnect_writer(int fd) {
  //struct timespec start;
  do {
    //clock_gettime(CLOCK_MONOTONIC_RAW, &start);
    if (send_supervision_frame(fd, DISC, A_3) == -1) {
      printf("Error writing DISC\n");
      continue;
    }
    alarm(TIMEOUT); // activa alarme de 3s
    printf("Sent DISC frame\n");
    flag = 0;
    printf("Receiving DISC response...\n");
    while (!flag) {
      if (receive_supervision_frame(fd, DISC) == 0) {
        reetransmit = 0;
        break;
      }
    }
```

```
    if (flag)
      printf("Timed Out - Retrying\n");
    //print_elapsed_time(start);

  } while (conta < 4 && flag);

  alarm(RESET_ALARM);

  if (conta == 4) {
    reetransmit = 2;
    printf("Gave up\n");
    return -1;
  }

  printf("Sending UA ACK...\n");
  return send_supervision_frame(fd, UA, A_1);
}

int disconnect_reader(int fd) {
  //struct timespec start;
  do {
    //clock_gettime(CLOCK_MONOTONIC_RAW, &start);
    alarm(TIMEOUT); // activa alarme de 3s
    flag = 0;
    printf("Receiving DISC from writer...\n");
    while (!flag) {
      if (receive_supervision_frame(fd, DISC) == 0) {
        reetransmit = 0;
        break;
      }
    }
    alarm(RESET_ALARM);
    printf("DISC received, sending DISC..\n");
    if (send_supervision_frame(fd, DISC, A_3) == -1) {
      printf("Error writing DISC\n");
      continue;
    }

    if (flag)
      printf("Timed Out - Retrying\n");
    //print_elapsed_time(start);

  } while (conta < 4 && flag);

  printf("Receiving UA...\n");

  return receive_supervision_frame(fd, UA);
}

int send_acknowledgement(int fd, int frame, int accept) {
  //printf("Sending acknowledgement...\n");
  if (frame == 0) {
    if (accept == 1) {
      // caso seja o frame 0 e seja aceite então pede o frame 1
      send_supervision_frame(fd, C_RR1, A_3);
    } else {
```

```
      send_supervision_frame(fd, C_REJ0, A_3);
    }
  } else {
    if (accept == 1) {
      send_supervision_frame(fd, C_RR0, A_3);
    } else {
      send_supervision_frame(fd, C_REJ1, A_3);
    }
  }
  return 0;
}

int llopen(int port, int type) {
  char file[48];
  sprintf(file, "/dev/ttyS%d", port);

  int fd;
  if (type == TRANSMITTER) {
    if ((fd = open_writer(file)) == ERROR) {
      perror("llopen: error on open_writer");
      return ERROR;
    }
    if (send_set(fd) == ERROR) {
      perror("llopen: error sending SET");
      return ERROR;
    }
    return fd;
  }
  else if (type == RECEIVER) {
    if ((fd = open_reader(file)) == ERROR) {
      perror("llopen: error on open_reader");
      return ERROR;
    }
    if (receive_set(fd) == ERROR) {
      perror("llopen: error receiving SET");
      return ERROR;
    }
    return fd;
  }
  perror("llopen: type not valid");
  return ERROR;
}

int llclose(int fd, int type) {
  printf("\nDisconnecting...\n");
  if (type == TRANSMITTER) {
    if (disconnect_writer(fd) == ERROR) {
      perror("llclose: error disconnecting writer: ");
      return ERROR;
    }
    if (close_writer(fd) != ERROR) {
      printf("Writer Successfully Closed!\n");
      return OK;
    } else {
      perror("llclose: writer not closed successfully: ");
      return ERROR;
```

```
    }
  }

  else if (type == RECEIVER) {
    if (disconnect_reader(fd) == ERROR) {
      perror("llclose: error disconnecting reader: ");
      return ERROR;
    }
    if (close_reader(fd) != ERROR) {
      printf("Reader Successfully Closed!\n");
      return OK;
    } else {
      perror("llclose: reader not closed successfully: ");
      return ERROR;
    }
  }
  return ERROR;
}

int llwrite(int fd, char *buffer, int length) {
  //printf("Sending data...\n");
  // printf("Message: %s\n", buffer);
  //printf("Coding message...\n");
  information_frame_t frame; // to keep everything organized

  frame.address = A_3;

  /* C byte - Controls package, alternating between 0 and 1*/
  frame.control = (current_frame == 0) ? C_I0 : C_I1;

  frame.bcc1 = frame.address ^ frame.control;

  int size_info = length;
  //printf("SENT DATA_SIZE: %d\n",size_info);
  unsigned char *information_frame = (unsigned char *) malloc
(size_info *sizeof(unsigned char));
  unsigned char bcc = 0x00;
  int i = 0,n=0;
  for (int j = 0; j < length; j++) {
    /* Data stuffing and buffer size adjusting*/
    if (buffer[j] == ESCAPE) {
      n++;
      information_frame =
          (unsigned char *)realloc(information_frame,
++size_info);
      information_frame[i++] = ESCAPE;
      information_frame[i++] = ESCAPE_ESC;
    } else if (buffer[j] == FLAG) {
      n++;
      information_frame =
          (unsigned char *)realloc(information_frame,
++size_info);
      information_frame[i++] = ESCAPE;
      information_frame[i++] = ESCAPE_FLAG;
    } else
      information_frame[i++] = buffer[j];
```

```
    bcc = buffer[j] ^ bcc;
  }
  frame.data = information_frame; /* saves the stuffed data-
buffer on the struct */
  frame.data_size = i;            /* size of the suffed data-
buffer */
  frame.bcc2 = bcc;               /* this BCC2 is not stuffed
yet and it will be displayed *unstuffed* */

  /* Saving all data to be transmitted to .raw_bytes */
  frame.raw_bytes = (unsigned char *)malloc((frame.data_size +
10));
  int j = 0;
  frame.raw_bytes[j++] = FLAG;
  frame.raw_bytes[j++] = frame.address;
  frame.raw_bytes[j++] = frame.control;
  frame.raw_bytes[j++] = frame.bcc1;
  for (int k = 0; k < frame.data_size; k++) {
    frame.raw_bytes[j++] = frame.data[k];
  }
  /* BCC2 stuffing*/
  if (bcc == ESCAPE) {
    n++;
    frame.raw_bytes[j++] = ESCAPE;
    frame.raw_bytes[j++] = ESCAPE_ESC;
    //printf("STUFFED BCC\n");
  } else if (bcc == FLAG) {
    n++;
    frame.raw_bytes[j++] = ESCAPE;
    frame.raw_bytes[j++] = ESCAPE_FLAG;
    //printf("STUFFED BCC\n");
  } else
    frame.raw_bytes[j++] = bcc;

  frame.raw_bytes[j++] = FLAG;

  // ! remove next comment if you want to see the coded message
being written
  //print_message(&frame, TRUE);
  conta = 1;
  int count = -1;
  //printf("SENT DATA_SIZE AFTER STUFFING: %d\n",j);
  //printf("STUFFED BYTES: %d\n",n);

  do {
    if ((count = write(fd, frame.raw_bytes, j)) != ERROR) {
      //printf("Message sent! Waiting for ACK\n");
    } else {
      //printf("Message not sent!\n");
      free(information_frame);
      free(frame.raw_bytes);
      return ERROR;
      // adicionei esta linha, pq caso não escreva corretamente
      // deve retornar -1 para escrever de novo
    }
```

```
    alarm(TIMEOUT);
    flag = 0;

    unsigned char ack = receive_acknowledgement(fd);
    if (ack == C_REJ0 || ack == C_REJ1) {
      //printf("Received negative ACK\n");
      alarm(RESET_ALARM);
      free(information_frame);
      free(frame.raw_bytes);
      return ERROR;
    }
    // Retransmition
    if ((ack == C_RR0 && current_frame == 0) ||
        (ack == C_RR1 && current_frame == 1)) {
      //printf("Received positive ACK (retransmition)\n");
      alarm(RESET_ALARM);
      // returns error but to the application only means it has
to
      // send the same frame again
      free(information_frame);
      free(frame.raw_bytes);
      return ERROR;
    }

    if ((ack == C_RR0 && current_frame == 1) ||
        (ack == C_RR1 && current_frame == 0)) {
      //printf("Received positive ACK\n");
      alarm(RESET_ALARM);
      current_frame = (current_frame == 0) ? 1 : 0; // changes
the current frame
      free(information_frame);
      free(frame.raw_bytes);
      return count;
    } else {
      //printf("Timed out\nTrying again\n");
      alarm(RESET_ALARM);
    }
    //printf("Couldn't receive ACK in time\n");
  } while (flag && conta < 4);
  free(information_frame);
  free(frame.raw_bytes);
  return ERROR;
}

int llread(int fd, char *buffer) {
  information_frame_t information_frame;
  information_frame.raw_bytes = (unsigned char
*)malloc(sizeof(unsigned char));

  int i = 0;
  int part = 0;
  unsigned char rcv_msg;
  //printf("Reading...\n");

  // * lógica: processar os dados todos em raw bytes, depois
fazer o unstuffing,
```

```
  // * e depois fazer o tratamento dos dados

  /*
    part 0 - before first flag
    part 1 - between flag start and flag stop
    part 2 - after flag stop */
  while (part != 2) {
    read(fd, &rcv_msg, 1);

    if (rcv_msg == FLAG && part == 0) {
      part = 1;
      continue;
    } else if (rcv_msg == FLAG && part == 1) {
      part = 2;
      break;
    }
    information_frame.raw_bytes[i++] = rcv_msg;
    information_frame.raw_bytes = (unsigned char
*)realloc(information_frame.raw_bytes, (i + 1));
  }

  int data_size = i;
  //printf("RECEIVED DATA_SIZE: %d\n",data_size);
  if(PROBABILITY_BCC1)
    generateErrorBCC1(information_frame.raw_bytes);
  if(PROBABILITY_BCC2)
    generateErrorBCC2(information_frame.raw_bytes,data_size);

  if(DELAY>0)
    generateDelay();

  /* UNSTUFFING BYTES */
  int j = 0, p = 0, n=0;
  for (; j < i && p < i; j++) {
    if (information_frame.raw_bytes[p] == ESCAPE) {
      n++;
      --data_size;
      if (information_frame.raw_bytes[p + 1] == ESCAPE_ESC)
        information_frame.raw_bytes[j] = ESCAPE;
      else if (information_frame.raw_bytes[p + 1] ==
ESCAPE_FLAG)
        information_frame.raw_bytes[j] = FLAG;
      p += 2;
    } else {
      information_frame.raw_bytes[j] =
information_frame.raw_bytes[p];
      p++;
    }
  }
  information_frame.raw_bytes =
          (unsigned char *)realloc(information_frame.raw_bytes,
data_size);
  //printf("AHHHHHHHHHHH: %d\n",n);
  information_frame.data =
      (unsigned char *)malloc((data_size - 4) * sizeof(unsigned
char));
```

```
  information_frame.address = information_frame.raw_bytes[0];
  information_frame.control = information_frame.raw_bytes[1];
  information_frame.bcc1 = information_frame.raw_bytes[2];
  p = 0;
  for (int byte = 3; byte < data_size - 1; byte++) {
    information_frame.data[p++] =
information_frame.raw_bytes[byte];
  }
  information_frame.bcc2 = information_frame.raw_bytes[data_size
- 1];
  information_frame.data_size = data_size - 4;
  //printf("DATA_SIZE: %d\n",data_size);

  // ! remove *sleep* comments if you want to check what happens
when ACK is not received in time
  // ! remove print_message comment if you want to see the data
byte-by-byte
  int bccError = verify_message(&information_frame);
  if (bccError == ERROR) {
    // sleep(15);
    send_acknowledgement(fd, current_frame, FALSE);
    //print_message(&information_frame, TRUE);
  } else {
    // sleep(4);
    send_acknowledgement(fd, current_frame, TRUE);
    current_frame = (current_frame == 0) ? 1 : 0;
    //print_message(&information_frame, TRUE);
  }

  for (i = 0; i < information_frame.data_size; i++) {
    buffer[i] = information_frame.data[i];
  }

  free(information_frame.raw_bytes);
  free(information_frame.data);
  return (bccError == OK) ? information_frame.data_size : ERROR;
}
```

data_link.h

```
#include "utils.h"

/** \addtogroup MACROS
 * @{
 */
#define TRANSMITTER 0
#define RECEIVER 1
#define STOP_AND_WAIT 1
/** @} */

/**
 * \defgroup DATALINKLAYER
 * @{
 */
```

```
/**
 * current I-Frame
 */
static int current_frame = 0;

/**
* @brief takes file descriptor (port) and sends a code msg in a
supervision frame
*/
int send_supervision_frame(int fd, unsigned char msg, unsigned
char address);

/**
* receives a supervision frame with controll as msg
*/
int receive_supervision_frame(int fd, unsigned char msg);

/**
* @brief Receives a ACK frame and returns it's control byte
*/
unsigned char receive_acknowledgement(int fd);

/**
 * @brief Method to send an ACK message
 * Takes a frame number (0, 1) and a flag (0, 1) and sends the
ACK value accordingly
 */
int send_acknowledgement(int fd, int frame, int accept);

/**
 * Reading Fucntion
 * @param fd Serial Port to be read
 */
int receive_set(int fd);

/**
 * @brief This function sends a SET Control frame and expects an
UA
 */
int send_set(int fd);

/**
 * @brief Disconnect method for writer
 * Sends DISC, expects DISC, sends UA to receiver
 */
int disconnect_writer(int fd);

/**
 * @brief Disconnect method for reader
 * Expects DISC, sends DISC, expects UA from emissor
 */
int disconnect_reader(int fd);

/**
 * @brief This function opens a port and starts the connection
 * @param port port number to be open
```

```
 * @param type RECEIVER or TRANSMITTOR
 * @return file descriptor for the port
 */
int llopen(int port, int type);

/**
 * @brief Same as llopen but this one disconnects writer or
reader and closes the descriptor
 */
int llclose(int fd, int type);

/**
 * @brief Function to write a buffer to a file descriptor
 * This function takes the buffer and sends it trought the port,
 * after the byte-stuffing
 * @returns -1 if error or number of bytes written for success
 */
int llwrite(int fd, char *buffer, int length);

/**
 * @brief Function to read a buffer from a file descriptor
 * This function reads a buffer from the port and returns the
number
 * @returns -1 if error or number of bytes read for success
 */
int llread(int fd, char *buffer);

/** @} */
```

files.c

```
#include "files.h"

unsigned long get_file_size(FILE* f) {
  fseek(f, 0, SEEK_END); // seek to end of file
  unsigned long size = ftell(f); // get current file pointer
  fseek(f, 0, SEEK_SET); // seek back to beginning of file
  // proceed with allocating memory and reading the file
  return size;
}

unsigned char* read_file(FILE* f, unsigned long filesize) {
  unsigned char* data = (unsigned char*) malloc (filesize);
  fread(data, sizeof(unsigned char), filesize, f);
  return data;
}

unsigned char* split_file(unsigned char* data, unsigned long
index_start, unsigned long index_end) {
  int range = index_end - index_start + 1;
  unsigned char* frame = (unsigned char*) malloc (range);

  for (int k = 0; k < range; k++) {
    frame[k] = data[index_start + k];
  }

  return frame;
```

```
}

void join_file(unsigned char* data, unsigned char* frame,
unsigned long size, int index) {
  for (int j = 0; j < size; j++) {
    data[index + j] = frame[j];
  }
}

void write_file(char* name, unsigned char* bytes, unsigned long
size) {
  FILE *fh = fopen (name, "wb");
  if (fh != NULL) {
      fwrite (bytes, sizeof (unsigned char), size, fh);
      fclose (fh);
  }
}
```

files.h

```
#include "macros.h"

/**
 * \addtogroup APPLICATIONLAYER
 * @{
 */

/**
 * @brief Method to calculate file's size in bytes
 */
unsigned long get_file_size(FILE* f);

/**
 * @brief Method to read all the bytes from a file
 */
unsigned char* read_file(FILE* f, unsigned long filesize);

/**
 * @brief Method to split a file's data
 * This method takes the data array, containing the file's data
and returns the data from
 * index_start to index_end
 */
unsigned char* split_file(unsigned char* data, unsigned long
index_start, unsigned long index_end);

/**
 * @brief Method to join a frame to the data buffer
 * This method joins the frame to the data, starting on the
position given by index
 */
void join_file(unsigned char* data, unsigned char* frame,
unsigned long size, int index);

/**
 * @brief Method to write bytes to a file with a given name
 */
```

```
void write_file(char* name, unsigned char* bytes, unsigned long
size);

/** @} */
```

macros.h

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

/** \addtogroup MACROS
 * @{
 */

#define BAUDRATE        B38400
#define MODEMDEVICE     "/dev/ttyS1"
#define _POSIX_SOURCE   1 /**< @brief POSIX compliant source */
#define FALSE           0
#define TRUE            1
#define OK              0
#define ERROR           -1
#define TRIES           3
#define TIMEOUT         3
#define RESET_ALARM     0

// Mesh Macros
#define FLAG            0x7E
#define A_3             0x03  /**< @brief Address for commands
given by the emissor and responses by the receiver */
#define A_1             0x01  /**< @brief Address for commands
given by the receiver and responses by the emissor */
#define SET             0x03
#define UA              0x07
#define DISC            0x0B
#define SET_BCC         A ^ SET
#define UA_BCC          A ^ UA

// Control Macros (there is another way of defining them)
#define C_I0            0x00  /**< @brief Control Flag for
Information Frame 0 */
#define C_I1            0x40  /**< @brief Control Flag for
Information Frame 1 */
#define C_RR0           0x05  /**< @brief Control Flag for ACK
Frame 0 */
#define C_RR1           0x85  /**< @brief Control Flag for ACK
Frame 1 */
```

```
#define C_REJ0          0x01  /**< @brief Control Flag for NACK
Frame 0 */
#define C_REJ1          0x81  /**< @brief Control Flag for NACK
Frame 1 */

//Byte Stuffing
#define ESCAPE          0x7D
#define ESCAPE_ESC      0x5D
#define ESCAPE_FLAG     0x5E

// packet Macros
#define DATA            0x1
#define START           0x2
#define STOP            0x3
#define FILE_SIZE       0x0
#define FILE_NAME       0x1
#define PACKET_SIZE     1024 /**< @brief Maximum packet size in
bytes */

// Progress Bar Macros
#define PROGRESS_BAR_SIZE   30
#define SEPARATOR_CHAR      '#'
#define EMPTY_CHAR          '.'
#define NUM_BACKSPACES      PROGRESS_BAR_SIZE + 9

// Error generating Macros
#define PROBABILITY_BCC1    0
#define PROBABILITY_BCC2    0
#define DELAY               0

/** @} */
```

noncanonical.c

```
/*Non-Canonical Input Processing*/
#include "noncanonical.h"

extern int reetransmit;
static struct termios oldtio;
static struct termios newtio;

int open_reader(char *port) {
  int fd;

  /* top layer does the verification of the port name */

  /*
    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(port, O_RDWR | O_NOCTTY);
  if (fd < 0) {
    perror(port);
    return -1;
```

```
  }

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

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

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

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

  /*
    VTIME e VMIN devem ser alterados de forma a proteger com um
temporizador a
    leitura do(s) proximo(s) caracter(es)
  */

  tcflush(fd, TCIOFLUSH);

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

  return fd;
}

int close_reader(int fd) {
  if (tcsetattr(fd, TCSANOW, &oldtio) == -1) {
    perror("tsetattr");
    return -1;
  }
  close(fd);
  return 0;
}
```

noncanonical.h

```
#include "macros.h"

/**
 * \addtogroup DATALINKLAYER
 * @{
 */

/**
 * @brief Opens the reader
```

```
 */
int open_reader(char *port);

/**
 * @brief Reset and close the reader
 */
int close_reader(int fd);

/** @} */
```

utils.c

```
#include "utils.h"

control_packet_t parse_control_packet(unsigned char *raw_bytes,
int size) {
  control_packet_t packet;
  memset(&packet, 0, sizeof(control_packet_t));
  packet.control = raw_bytes[0];

  char *name;
  int namesize = 0;

  unsigned char* filesize;
  int filesize_size = 0;

  for (int i = 1; i < size; i++) {
    if (raw_bytes[i] == FILE_SIZE) {
      int length = raw_bytes[++i];
      int offset = i + length;
      filesize = (unsigned char*) malloc (length);
      for (int k = 0; i < offset; k++) {
        filesize[k] = raw_bytes[++i];
        filesize_size++;
      }
      continue;
    }
    if (raw_bytes[i] == FILE_NAME) {
      int length = raw_bytes[++i];
      name = (unsigned char *) malloc (length);
      int offset = i + length;
      for (int j = 0; i < offset;) {
        name[j++] = raw_bytes[++i];
        namesize++;
      }
      continue;
    }
  }

  packet.file_name = (unsigned char*) malloc (namesize + 1);
  memcpy(packet.file_name, name, namesize);
  packet.file_name[namesize] = '\0';
  free(name);

  packet.filesize_size = filesize_size;
  packet.file_size = (unsigned char*) malloc (filesize_size);
  memcpy(packet.file_size, filesize, filesize_size);
```

```
  free(filesize);

  return packet;
}

data_packet_t parse_data_packet(unsigned char *raw_bytes, int
size) {
  data_packet_t packet;
  memset(&packet, 0, sizeof(data_packet_t));
  packet.raw_bytes_size = size;
  packet.control = raw_bytes[0];
  packet.sequence = raw_bytes[1];

  packet.data_field_size = (raw_bytes[2] << 8) | raw_bytes[3];

  for (int i = 0; i < packet.data_field_size; i++) {
    packet.data[i] = raw_bytes[4 + i];
  }

  return packet;
}

void print_control_packet(control_packet_t* packet) {
  printf("\n---- CONTROL PACKET ----\n");
  switch (packet->control) {
  case START:
    printf("Control: START (0x%x)\n", packet->control);
    break;
  case STOP:
    printf("Control: STOP (0x%x)\n", packet->control);
    break;
  default:
    break;
  }

  printf("Size: %ld %#lx\n", array_to_number(packet->file_size,
packet->filesize_size), array_to_number(packet->file_size,
packet->filesize_size));
  printf("Name: %s\n", packet->file_name);
  printf("------------------------\n");
}

void print_data_packet(data_packet_t* packet, int full_info) {
  printf("---- DATA PACKET ----\n");
  printf("Control: - (0x%x)\n", packet->control);
  printf("Data size: %d (0x%x)\n", packet->data_field_size,
         packet->data_field_size);
  printf("Sequence: %d (0x%x)\n", packet->sequence, packet-
>sequence);

  if (full_info) {
    for (int i = 0; i < packet->data_field_size; i++) {
      printf("DATA[%d]: %c (0x%x)\n", i, packet->data[i],
packet->data[i]);
    }
  }
```

```
  printf("---------------------\n");
}

void print_message(information_frame_t* frame, int stuffed) {
  printf("Address: 0x%x\n", frame->address);
  printf("Control: 0x%x\n", frame->control);
  printf("BCC1: 0x%x\n", frame->bcc1);
  int j = 0;
  for (int i = 0; i < frame->data_size; i++) {
    if (frame->data[i] == ESCAPE && stuffed) {
      printf("DATA[%d]: 0x%x - ESCAPE\n", j++, frame-
>data[i++]);
      if (frame->data[i] == ESCAPE_ESC) {
        printf("DATA[%d]: 0x%x - ESCAPED ESCAPE\n", j++, frame-
>data[i]);
      } else if (frame->data[i] == ESCAPE_FLAG) {
        printf("DATA[%d]: 0x%x - ESCAPED FLAG\n", j++, frame-
>data[i]);
      }
    } else {
      printf("DATA[%d]: 0x%x - %c (char)\n", j++, frame-
>data[i], frame->data[i]);
    }
  }
  printf("BCC2: 0x%x\n", frame->bcc2);
  printf("Message: %s - size: %d - strlen: %ld\n", frame->data,
frame->data_size,
         strlen(frame->data));
}

int verify_message(information_frame_t* frame) {
  if (frame->bcc1 != (frame->control ^ frame->address)) {
    printf("Error in BCC1\n");
    return ERROR;
  }
  unsigned char bcc2 = 0x00;
  //printf("BCC2: 0x%x\tFRAME: 0x%x\tFRAME BCC2:
0x%x\n",bcc2,frame->data[0],frame->bcc2);
  for (int i = 0; i < frame->data_size; i++) {
    //printf("BCC2: 0x%x\tFRAME: 0x%x\n",bcc2,frame->data[i]);
    bcc2 ^= frame->data[i];
  }

  if (bcc2 != frame->bcc2) {
    printf("BCC2: 0x%x\tFRAME BCC2: 0x%x FRAME_DATA_SIZE:
%d\n",bcc2,frame->bcc2,frame->data_size);
    printf("Error in BCC2\n");
    return ERROR;
  }

  return OK;
}

void clearProgressBar() {
    int i;
```

```
    for (i = 0; i < NUM_BACKSPACES; ++i) {
        fprintf(stdout, "\b");
    }
    fflush(stdout);
}

void printProgressBar(int progress, int total) {
    int i, percentage = (int)((((double)progress) / total) *
100);
    int num_separators = (int)((((double)progress) / total) *
PROGRESS_BAR_SIZE);;
    fprintf(stdout, "[");
    for (i = 0; i < num_separators; ++i) {
        fprintf(stdout, "%c", SEPARATOR_CHAR);
    }
    for (; i < PROGRESS_BAR_SIZE; ++i) {
        fprintf(stdout, "%c", EMPTY_CHAR);
    }
    fprintf(stdout, "]  %2d%%  ", percentage);
    fflush(stdout);
}


void print_elapsed_time(struct timespec start) {
  struct timespec end;
  clock_gettime(CLOCK_MONOTONIC_RAW, &end);
  double delta = (end.tv_sec - start.tv_sec) * 1000.0 +
                 (end.tv_nsec - start.tv_nsec) / 1000000.0;
  printf("Elapsed time: %f ms\n\n", delta);
}

unsigned long array_to_number(unsigned char* buffer, unsigned
int size) {
  unsigned long value = 0;
  int offset = 0;
  for (int i = 0; i < size; i++) {
    value |= buffer[i] << (8 * offset++);
  }

  return value;
}

unsigned int number_to_array(unsigned long num, unsigned char*
buffer) {
  unsigned int size = 0;

  for (int i = 0; i < sizeof(unsigned long); i++) {
    buffer[i] = (num >> (8 * i)) & 0xff;
    size += 1;
  }
  for (int i = sizeof(unsigned long) - 1; i != 0; i--) {
    if (buffer[i] != 0) break;
    size--;
  }

  return size;
```

```
}

void generateErrorBCC2(unsigned char *frame, int frameSize){
  int prob = (rand() % 100) + 1;

  if (prob <= PROBABILITY_BCC2){
    int i = (rand() % (frameSize - 5)) + 4; /* only considering
data and BCC2*/
    unsigned char randomAscii = (unsigned char)((rand() % 177));
    frame[i] = randomAscii;
  }

}

void generateErrorBCC1(unsigned char *frame){
  int prob = (rand() % 100) + 1;
  if (prob <= PROBABILITY_BCC1)
  {
    int i = (rand() % 2);
    unsigned char randomAscii = (unsigned char)((rand() % 177));
    frame[i] = randomAscii;
  }

}

void generateDelay() {
  usleep(DELAY*1e6);
}

control_packet_t generate_control_packet(int control, file_t*
file) {
  control_packet_t c_packet;
  c_packet.control = control;
  c_packet.file_name = file->name;

  unsigned char buf[sizeof(unsigned long)];
  int num = number_to_array(file->size, buf);

  c_packet.file_size = (unsigned char *)malloc(num);
  memcpy(c_packet.file_size, buf, num);
  c_packet.filesize_size = num;

  int i = 0;
  // control packet
  c_packet.raw_bytes = (unsigned char *)malloc(i + 1);
  c_packet.raw_bytes[i++] = c_packet.control;
  c_packet.raw_bytes = (unsigned char
*)realloc(c_packet.raw_bytes, (i + 1));
  // file size
  c_packet.raw_bytes[i++] = FILE_SIZE;
  c_packet.raw_bytes = (unsigned char
*)realloc(c_packet.raw_bytes, (i + 1));
  c_packet.raw_bytes[i++] = c_packet.filesize_size;

  for (int j = 0; j < c_packet.filesize_size; j++) {
```

```
    c_packet.raw_bytes = (unsigned char
*)realloc(c_packet.raw_bytes, (i + 1));
    c_packet.raw_bytes[i++] = c_packet.file_size[j];
  }
  c_packet.raw_bytes = (unsigned char
*)realloc(c_packet.raw_bytes, (i + 1));
  // file name
  c_packet.raw_bytes[i++] = FILE_NAME;
  c_packet.raw_bytes = (unsigned char
*)realloc(c_packet.raw_bytes, (i + 1));
  c_packet.raw_bytes[i++] = strlen(c_packet.file_name);
  c_packet.raw_bytes = (unsigned char
*)realloc(c_packet.raw_bytes, (i + 1));
  for (int j = 0; j < strlen(c_packet.file_name); j++) {
    c_packet.raw_bytes[i++] = c_packet.file_name[j];
    c_packet.raw_bytes = (unsigned char
*)realloc(c_packet.raw_bytes, (i + 1));
  }

  c_packet.raw_bytes_size = i;
  return c_packet;
}

data_packet_t generate_data_packet(unsigned char *buffer, int
size, int sequence) {
  data_packet_t d_packet;
  d_packet.control = DATA;
  d_packet.data_field_size = size;
  d_packet.sequence = sequence;

  int i = 0;
  // control
  d_packet.raw_bytes = (unsigned char *)malloc(i + 1);
  d_packet.raw_bytes[i++] = d_packet.control;
  d_packet.raw_bytes = (unsigned char
*)realloc(d_packet.raw_bytes, (i + 1));
  // sequence
  d_packet.raw_bytes[i++] = d_packet.sequence;
  d_packet.raw_bytes = (unsigned char
*)realloc(d_packet.raw_bytes, (i + 1));
  // size
  unsigned int x = (unsigned int)size;
  unsigned char high = (unsigned char)(x >> 8);
  unsigned char low = x & 0xff;
  d_packet.raw_bytes[i++] = high;
  d_packet.raw_bytes = (unsigned char
*)realloc(d_packet.raw_bytes, (i + 1));
  d_packet.raw_bytes[i++] = low;
  d_packet.raw_bytes = (unsigned char
*)realloc(d_packet.raw_bytes, (i + 1));
  // data
  for (int j = 0; j < size; j++) {
    d_packet.data[j] = buffer[j];
    d_packet.raw_bytes[i++] = buffer[j];
    d_packet.raw_bytes = (unsigned char
*)realloc(d_packet.raw_bytes, (i + 1));
```

```
  }

  d_packet.raw_bytes_size = i;
  return d_packet;
}
```

utils.h

```
#include "macros.h"
#include "writenoncanonical.h"
#include "noncanonical.h"
#include "app_structs.h"

/**
 * \addtogroup UTILS
 * @{
 */

/**
 * @brief This function parses a buffer of raw bytes to a
control_packet_t
 * structure
 *
 * This method takes the raw_bytes and parses them into a
control packet
 */
control_packet_t parse_control_packet(unsigned char *raw_bytes,
int size);

/**
 * @brief This function parses a buffer of raw bytes to a
data_packet_t
 * structure
 *
 * This method takes the raw_bytes and parses them into a data
packet
 */
data_packet_t parse_data_packet(unsigned char *raw_bytes, int
size);

/**
 * @brief This function pretty-prints a control packet
 */
void print_control_packet(control_packet_t* packet);

/**
 * @brief This function pretty-prints a data packet
 */
void print_data_packet(data_packet_t* packet, int full_info);

/**
 * @brief Method to pretty-print the elapsed time between two
frames
 */
void print_elapsed_time(struct timespec start);

/**
```

```
 * @brief Method to verify an I-Frame
 *
 * Checks if there are errors on the BCC bytes
 * @returns 0 if no error or -1 for error
 */
int verify_message(information_frame_t* frame);

// Créditos ao grupo 4 da turma 3

/**
 * @brief Method to clear progress bar
 */
void clearProgressBar();
/**
 * @brief Method to print the progress bar
 */
void printProgressBar(int progress, int total);

/**
 * @brief Method to pretty-print an information frame details
 */
void print_message(information_frame_t* frame, int coded);

/**
 * @brief takes an array with length size, and converts it to an
8byte number
 *
 * array[0] = MSB ; array[size - 1] = LSB
 */
unsigned long array_to_number(unsigned char* buffer, unsigned
int size);

/**
 * @brief Method to convert a 8byte number into an 8 byte char
array
 *
 * array[0] = MSB ; array[return - 1] = LSB
 * @return array's size
 */
unsigned int number_to_array(unsigned long num, unsigned char*
buffer);

/**
 * @brief Method to generate error in BCC2
 *
 */
void generateErrorBCC2(unsigned char *frame, int frameSize);

/**
 * @brief Method to generate error in BCC1
 *
 */
void generateErrorBCC1(unsigned char *checkBuffer);

/**
```

```
 * @brief Method to generate delay in processing of received
frame
 */
void generateDelay();
/**
 * @brief Method to generate a control packet (START or STOP)
 * @return control_packet_t structure
 */
control_packet_t generate_control_packet(int control, file_t*
file);

/**
 * @brief Method to generate a data packet
 *
 * This method takes a buffer with a given size and a sequence
number and generates
 * a data_packet_t structure
 * @return data_packet_t structure
 */
data_packet_t generate_data_packet(unsigned char *buffer, int
size, int sequence);

/** @} */
```

writenoncanonical.c

```
/*Non-Canonical Input Processing*/
#include "writenoncanonical.h"

int flag = 1, conta = 1;
extern int reetransmit;
static struct termios oldtio;
static struct termios newtio;

void atende() { // atende alarme
  printf("alarme # %d\n", conta);
  flag = 1;
  conta++;
}

int open_writer(char *port) {
  /* top level layer should verifiy ports name */

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

  int fd = open(port, O_RDWR | O_NOCTTY);
  if (fd < 0) {
    perror(port);
    return -1;
  }
```

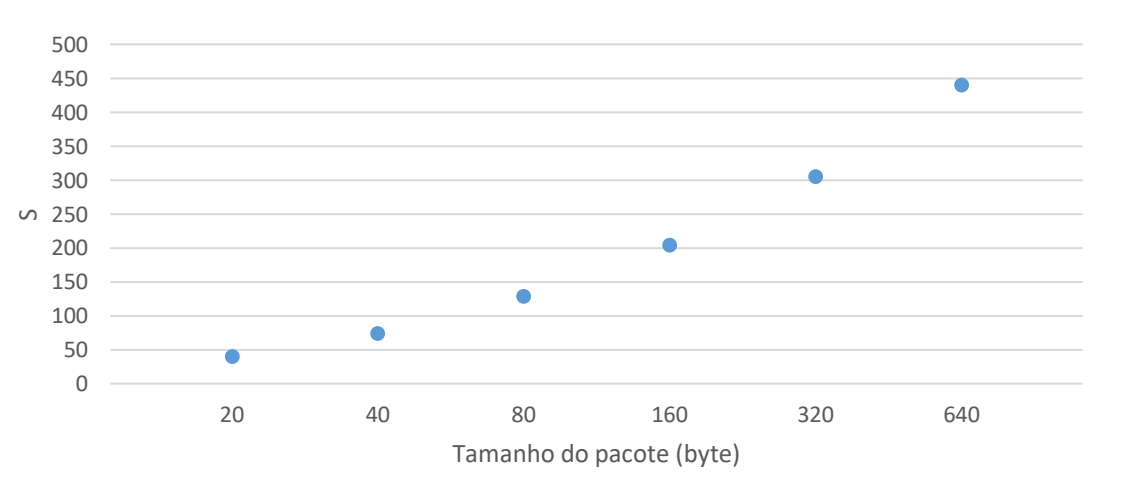
```
  if (tcgetattr(fd, &oldtio) == -1) {
    /* save current port settings */
    perror("tcgetattr");
    return -1;
  }

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

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

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

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

  tcflush(fd, TCIOFLUSH);

  if (tcsetattr(fd, TCSANOW, &newtio) == -1) {
    perror("tcsetattr");
    return -1;
  }
  /*struct sigaction action;
       sigemptyset(&action.sa_mask);
       action.sa_handler = atende;
       action.sa_flags = 0;
       sigaction(SIGALRM,&action,NULL);*/ // instala  rotina que
atende interrupcao
  signal(SIGALRM, atende);
  siginterrupt(SIGALRM, 1);
  printf("New termios structure set\n");

  return fd;
}

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

  close(fd);
  return 0;
}
```

writenoncanonical.h

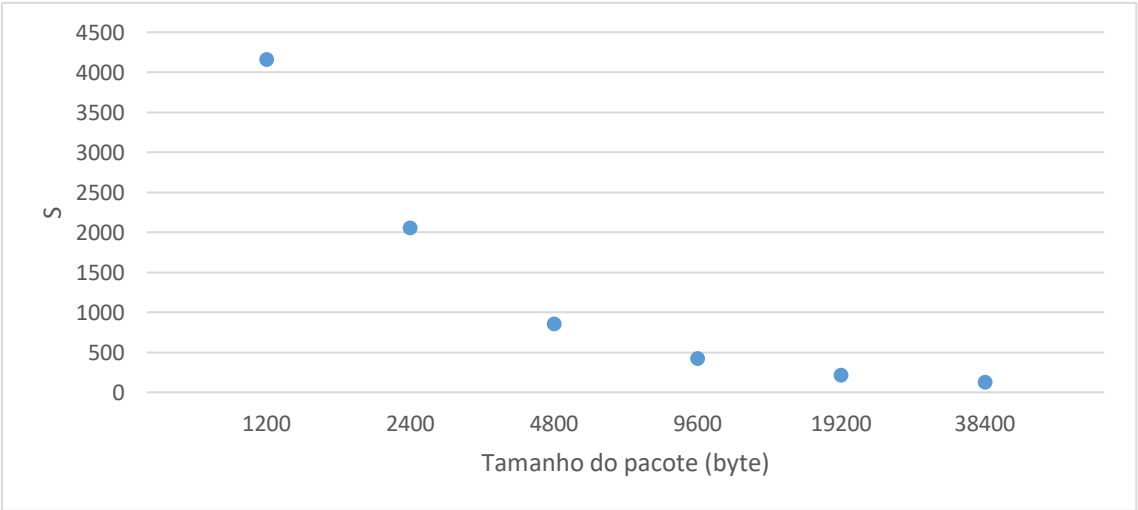
```
#include "macros.h"

/**
 * \addtogroup DATALINKLAYER
 * @{
 */

/**
* function to open and set port
*/
int open_writer(char *port);

/**
* function to reset and close port
*/
int close_writer(int fd);

/** @} */
```

Anexo II – Resultados em Ambiente Emulado (socat)

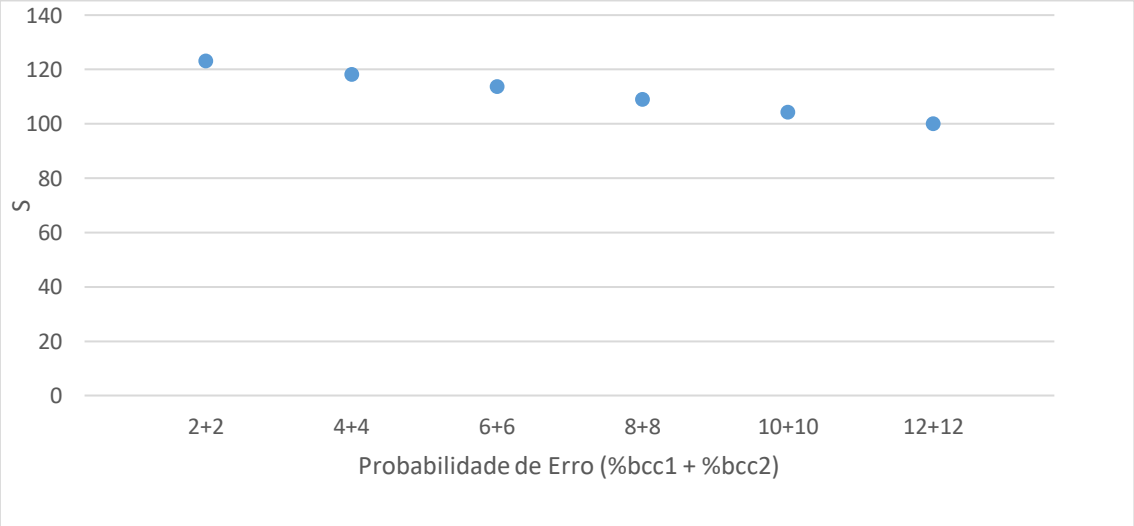
Nª de bytes	3309702			
Baudrate	38400			
Tamanho do pacote (byte)	Tempo (s)	R (bits/s)	S (R/C)	S (média)
10	32.8775408250	805340.5253	20.97240951	20.90135027
	33.09377789	800078.3739	20.83537432	
	32.99734114	802416.6519	20.89626698	
20	17.32165967	1528584.241	39.80688127	40.1104848
	16.86313914	1570147.514	40.88925819	
	17.39663861	1521996.093	39.63531492	
40	9.369149916	2826042.516	73.59485718	73.28307749
	9.510464306	2784050.825	72.50132358	
	9.349053823	2832117.185	73.7530517	
80	5.285104169	5009856.978	130.4650255	128.4879281
	5.440654181	4866623.593	126.7349894	
	5.375806843	4925328.75	128.2637695	
160	3.409024421	7766918.84	202.2635114	203.6454566
	3.34785441	7908831.376	205.9591504	
	3.401453491	7784206.39	202.7137081	
320	2.264595292	11691985.8	304.4787969	304.5733374
	2.263796473	11696111.52	304.5862374	
	2.263285685	11698751.15	304.6549777	
640	1.564099827	16928341.49	440.8422264	439.7445045
	1.585893333	16695710.52	434.7841281	
	1.554351042	17034514.91	443.6071591	



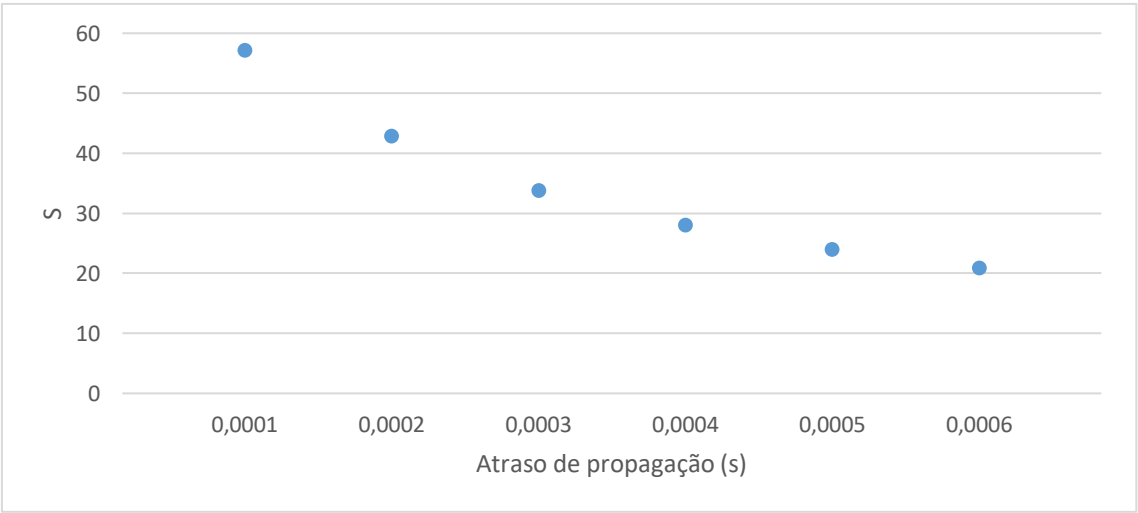
Nª de bytes	3309702			
Tamanho do pacote (byte)	80			
Baudrate	Tempo (s)	R (bits/s)	S (R/C)	S (média)
600	5.446173493	4861691.614	8102.819357	8107.761492
	5.458661425	4850569.387	8084.282311	
	5.423840766	4881709.686	8136.182809	
1200	5.330546552	4967148.442	4139.290368	4154.306764
	5.303102648	4992853.761	4160.711467	
	5.300291184	4995502.149	4162.918458	
2400	5.399015633	4904156.202	2043.398417	2051.796341
	5.406818799	4897078.483	2040.449368	
	5.325667571	4971698.974	2071.541239	
4800	5.41531813	4889392.528	1018.623443	849.2492831
	5.315436942	4981268.01	1037.764169	
	5.423690153	4881845.248	1017.051093	
9600	5.386113534	4915903.802	512.0733127	423.3129352
	5.486924366	4825584.286	502.6650298	
	5.391246562	4911223.35	511.5857656	
19200	5.393457829	4909209.794	255.6880101	211.1797406
	5.45962937	4849709.423	252.5890324	
	5.444188029	4863464.645	253.3054503	
38400	5.401877541	4901557.986	127.6447392	128.2127245
	5.35097642	4948184.018	128.8589588	
	5.405191045	4898553.22	127.5664901	



Nª de bytes	3309702			
Tamanho do pacote (byte)	80			
Baudrate	38400			
Probabilidade de Erro (%bcc1 + %bcc2)	Tempo (s)	R (bits/s)	S (R/C)	S (méd)
0+0	5.414071622	4890518.236	127.3572457	127.4972028
	5.393290746	4909361.881	127.8479656	
	5.417085144	4887797.643	127.286397	
2+2	5.6371184	4697012.573	122.3180358	123.1284581
	5.631430171	4701756.96	122.4415875	
	5.532734958	4785628.844	124.6257511	
4+4	5.789007976	4573774.317	119.1087062	118.2651449
	5.833020275	4539263.495	118.2099868	
	5.869427772	4511106.879	117.4767416	
6+6	6.022460681	4396478.018	114.4916151	113.8404209
	6.134936291	4315874.647	112.3925689	
	6.014818748	4402063.821	114.6370787	
8+8	6.39647259	4139408.968	107.7971085	108.9957732
	6.310871105	4195556.455	109.2592827	
	6.272313527	4221347.655	109.9309285	
10+10	6.631386477	3992772.264	103.9784444	104.3675619
	6.620821271	3999143.749	104.1443685	
	6.568128069	4031227.12	104.9798729	
12+12	6.995931339	3784716.39	98.56032265	100.1615929
	6.859194174	3860164.23	100.5251102	
	6.800056194	3893734.882	101.3993459	

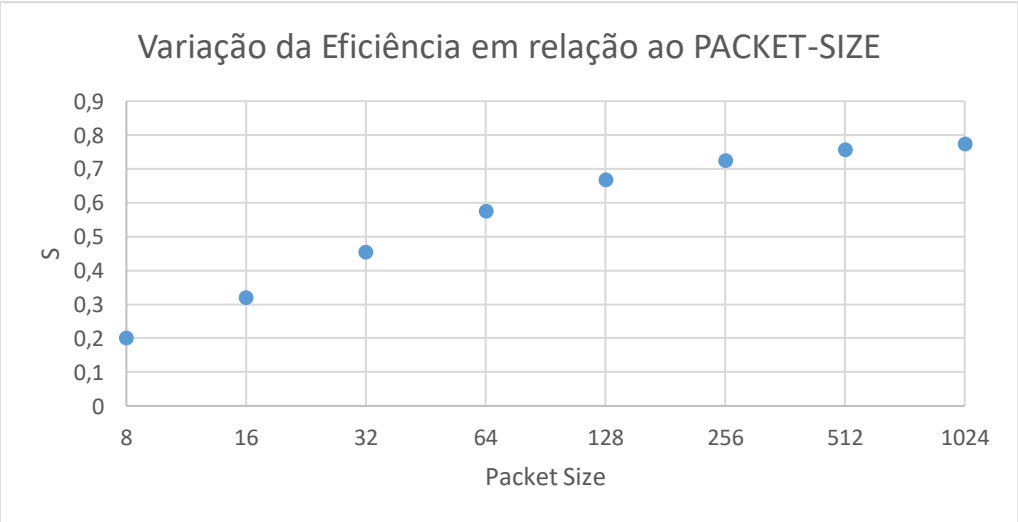


Nª de bytes	3309702			
Tamanho do pacote (byte)	80			
Baudrate	38400			
Atraso de propagação (s)	Tempo (s)	R (bits/s)	S (R/C)	S (média)
0	5.413779014	4890782.563	127.3641292	127.8881822
	5.340016754	4958339.5	129.1234245	
	5.421745195	4883596.526	127.1769929	
0.0001	11.99577542	2207245.056	57.48034001	57.07580946
	12.08446702	2191045.41	57.05847421	
	12.1633111	2176842.783	56.68861415	
0.0002	16.02523687	1652244.907	43.02721111	42.81513049
	16.12719695	1641799.011	42.75518258	
	16.16204407	1638259.114	42.66299777	
0.0003	20.40006409	1297918.275	33.79995509	33.77263124
	20.40083658	1297869.129	33.79867523	
	20.44888234	1294819.715	33.7192634	
0.0004	24.63699705	1074709.549	27.98722785	27.98460174
	24.66738437	1073385.634	27.95275088	
	24.61360465	1075730.937	28.0138265	
0.0005	28.84104608	918053.2471	23.90763664	23.92054387
	28.85447256	917626.0611	23.89651201	
	28.78103895	919967.3455	23.95748296	
0.0006	33.12978908	799208.7101	20.81272683	20.84779124
	33.0878775	800221.0476	20.83908978	
	33.00478017	802235.7933	20.89155712	

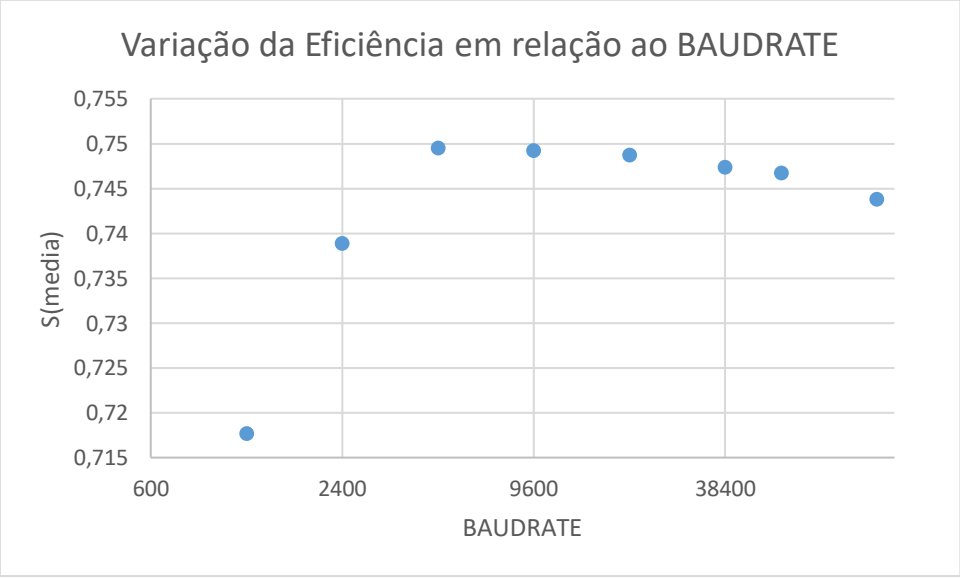


Anexo III – Resultados em ambiente de Laboratório

Nº Bytes	10968				
Baudrate	38400				
Tamanho do Pacote (byte)	Tempo (ms)	Tempo (s)	R (bits/s)	S (R/C)	S(media)
8	11441.99315	11.44199315	7668.594	0.199702969	0.199653671
	11443.30931	11.44330931	7667.711993	0.19968	
	11449.15523	11.44915523	7663.796868	0.199578043	
16	7182.167579	7.182167579	12216.92463	0.318149079	0.318124175
	7183.123636	7.183123636	12215.29859	0.318106734	
	7182.898328	7.182898328	12215.68175	0.318116712	
32	5046.919794	5.046919794	17385.65374	0.4527514	0.45270126
	5047.566454	5.047566454	17383.42641	0.452693396	
	5047.950162	5.047950162	17382.10505	0.452658986	
64	3982.716447	3.982716447	22031.19433	0.573729019	0.573665759
	3984.113762	3.984113762	22023.46751	0.5735278	
	3982.637038	3.982637038	22031.6336	0.573740458	
128	3424.833772	3.424833772	25619.92956	0.667185666	0.667127015
	3425.187519	3.425187519	25617.28358	0.66711676	
	3425.383353	3.425383353	25615.81901	0.66707862	
256	3156.71057	3.15671057	27796.02312	0.723854769	0.723826016
	3157.141282	3.157141282	27792.23106	0.723756017	
	3156.656095	3.156656095	27796.50281	0.723867261	
512	3026.13398	3.02613398	28995.4115	0.755088841	0.755091102
	3025.991643	3.025991643	28996.77539	0.755124359	
	3026.249148	3.026249148	28994.30804	0.755060105	
1024	2955.072429	2.955072429	29692.67323	0.773246699	0.773247265
	2955.069635	2.955069635	29692.7013	0.77324743	
	2955.068728	2.955068728	29692.71042	0.773247667	

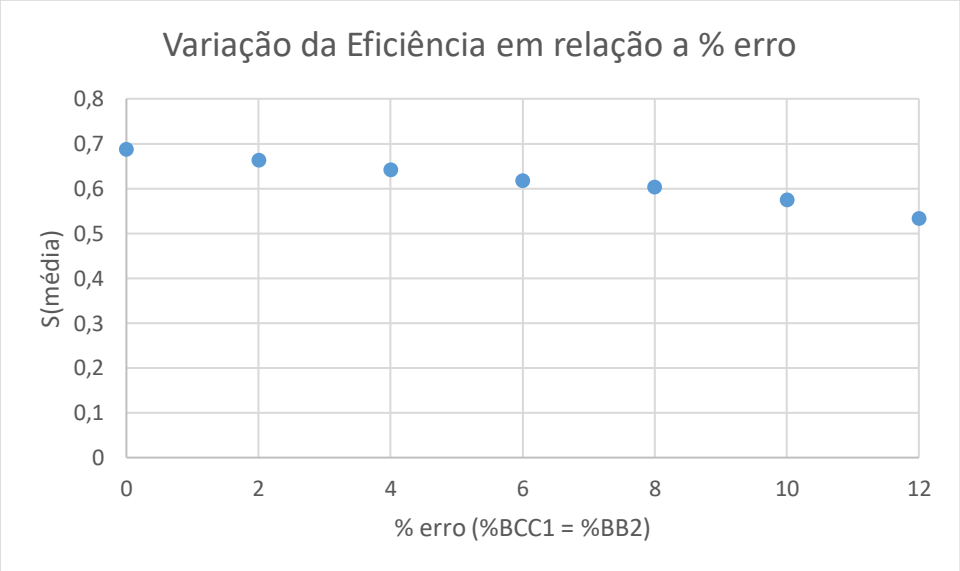


Nº Bytes	10968				
Tamanho do Pacote (byte)	256				
Baudrate	Tempo (ms)	Tempo (s)	R (bits/s)	S (R/C)	S(media)
1200	101887.6863	101.8876863	861.1835559	0.717652963	0.717655833
	101887.0938	101.8870938	861.1885642	0.717657137	
	101887.0566	101.8870566	861.1888782	0.717657399	
2400	50964.36116	50.96436116	1721.673695	0.71736404	0.738871218
	48771.23196	48.77123196	1799.093369	0.749622237	
	48770.89756	48.77089756	1799.105704	0.749627377	
4800	24389.79043	24.38979043	3597.570888	0.749493935	0.749495768
	24389.72617	24.38972617	3597.580366	0.749495909	
	24389.67575	24.38967575	3597.587803	0.749497459	
9600	12199.00244	12.19900244	7192.719277	0.749241591	0.749243017
	12198.94964	12.19894964	7192.750409	0.749244834	
	12198.98561	12.19898561	7192.729201	0.749242625	
19200	6103.522296	6.103522296	14375.96125	0.748747982	0.748734591
	6103.298821	6.103298821	14376.48763	0.748775397	
	6104.07329	6.10407329	14374.66358	0.748680395	
38400	3059.717301	3.059717301	28677.15915	0.74680102	0.747363668
	3056.734519	3.056734519	28705.14252	0.747529753	
	3055.79236	3.05579236	28713.99286	0.747760231	
57600	2039.873855	2.039873855	43014.42454	0.746778204	0.746721054
	2040.32531	2.04032531	43004.9069	0.746612967	
	2039.890827	2.039890827	43014.06665	0.746771991	
115200	1024.019605	1.024019605	85685.8595	0.743800864	0.743809234
	1024.004519	1.024004519	85687.12186	0.743811822	
	1024.000119	1.024000119	85687.49004	0.743815018	



Nº Bytes	10968
Tamanho do pacote (byte)	128
Baudrate	38400

Probabilidade de Erro (%bcc1 = %bcc2)	Tempo (ms)	Tempo (s)	R (bits/s)	S (R/C)	S(media)
0	3327.764236	3.327764236	26367.25254	0.686647202	0.686668161
	3327.651721	3.327651721	26368.14407	0.686670419	
	3327.572033	3.327572033	26368.77553	0.686686863	
2	3443.786043	3.443786043	25478.93478	0.663513927	0.663534493
	3443.596284	3.443596284	25480.3388	0.66355049	
	3443.655579	3.443655579	25479.90006	0.663539064	
4	3559.324758	3.559324758	24651.86685	0.641975699	0.642007889
	3559.326993	3.559326993	24651.85137	0.641975296	
	3558.78719	3.55878719	24655.5906	0.642072672	
6	3702.799986	3.702799986	23696.66208	0.617100575	0.617133545
	3702.534449	3.702534449	23698.36154	0.617144832	
	3702.472081	3.702472081	23698.76074	0.617155228	
8	3790.115	3.790115	23150.74872	0.602884081	0.602896097
	3790.254473	3.790254473	23149.89683	0.602861897	
	3789.748962	3.789748962	23152.98477	0.602942312	
10	3982.707229	3.982707229	22031.24532	0.573730347	0.573749521
	3982.542263	3.982542263	22032.1579	0.573754112	
	3982.472911	3.982472911	22032.54158	0.573764104	
12	4283.961599	4.283961599	20481.97631	0.5333848	0.533381262
	4283.738247	4.283738247	20483.04423	0.53341261	
	4284.270228	4.284270228	20480.50084	0.533346376	



Nº Bytes	10968
Tamanho do pacote (byte)	128
Baudrate	38400

Delay (s)	Tempo (ms)	Tempo (s)	R (bits/s)	S (R/C)	S(media)
0.0001	3424.559	3.424559	25621.9852	0.667239198	0.667239453
	3424.715934	3.424715934	25620.8111	0.667208622	
	3424.398157	3.424398157	25623.18865	0.667270538	
0.001	3503.407837	3.503407837	25045.32846	0.652222095	0.652205605
	3503.394567	3.503394567	25045.42332	0.652224566	
	3503.686855	3.503686855	25043.33396	0.652170155	
0.01	4295.849505	4.295849505	20425.29653	0.531908764	0.531935179
	4295.535707	4.295535707	20426.78864	0.531947621	
	4295.523345	4.295523345	20426.84743	0.531949152	
0.1	12216.20637	12.21620637	7182.589861	0.187046611	0.187047983
	12216.01389	12.21601389	7182.703033	0.187049558	
	12216.12996	12.21612996	7182.634785	0.187047781	
1	91414.70188	91.41470188	959.8456068	0.024995979	0.024996028
	91414.4339	91.4144339	959.8484206	0.024996053	
	91414.43865	91.41443865	959.8483707	0.024996051	
4	352037.5063	352.0375063	249.2461696	0.006490786	0.006490786
	352037.0053	352.0370053	249.2465243	0.006490795	
	352037.9534	352.0379534	249.245853	0.006490777