U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

1º Projeto de Redes de Computadores 2022/2023

Ligação de Dados

Licenciatura em Engenharia Informática e Computação

Trabalho realizado por:

Bárbara Rodrigues up202007163

Igor Diniz up202000162

José Rodrigues up202008462

Sumário

Este relatório foi realizado no âmbito da unidade curricular redes de computadores (RCOM) do 3º ano da Licenciatura em Engenharia Informática e de Computação (LEIC). O relatório incide sobre o primeiro trabalho laboratorial realizado, cujo foco é a transferência de dados através de uma aplicação. A transferência de dados é feita através da implementação de um protocolo de comunicação entre duas máquinas. O trabalho prático foi concluído com sucesso sendo todos os objetivos definidos no início alcançados.

O relatório serve para detalhar a implementação do trabalho bem como explicar os conceitos teóricos que foram aplicados.

Introdução

O trabalho apresentado foi desenvolvido em linguagem de progamação C e tem como objetivo a implementação de um protocolo de ligação de dados em Linux, através da utilização de portas série RS-232 (comunicação assíncrona). Este protocolo fornece um serviço de comunicação de dados fiável entre dois sistemas ligados por um meio de transmissão neste caso, um cabo série e é testado com uma aplicação simples de transferência de ficheiros entre o Emissor e o Recetor, sendo todo o seu processo de construção, desenvolvimento e implementação explicado ao longo deste relatório.

O objetivo do relatório é explicar ao leitor a parte teórica deste trabalho, bem como a nossa implementação do que foi proposto pelo guião, tendo a seguinte estrutura:

- **Arquitetura e Estrutura do Código** - Visualização dos blocos funcionais e interfaces. Representação das APIs, principais estruturas de dados, principais funções e sua relação com a arquitetura.
- **Casos de Uso Principais** - Identificação dos mesmos bem com as sequências de chamada de funções.
- **Protocolo de Ligação Lógica** - Identificação dos principais aspetos funcionais bem como a descrição da estratégia de implementação destes aspetos com apresentação de extratos de código.
- **Protocolo de Aplicação** - Identificação dos principais aspetos funcionais bem como a descrição da estratégia de implementação destes aspetos com apresentação de extratos de código.
- **Validação** - Descrição dos testes efetuados com apresentação quantificada dos resultados.
- **Eficiência do protocolo de ligação de dados** - Caracterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido.
- **Conclusões** - Síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

Arquitetura e Estrutura do código

Em termos de fluxo do código, é o ficheiro main.c que inicia todo o processo de transmissão de dados. É neste ficheiro que são definidas as especificações da porta série e do alarme.

```
#define BAUDRATE 9600
#define N_TRIES 3
#define TIMEOUT 4
```

```
"Starting link-layer protocol application\n"
           "   - Serial port: %s\n"
           "   - Role: %s\n"
           "   - Baudrate: %d\n"
           "   - Number of tries: %d\n"
           "   - Timeout: %d\n"
           "   - Filename: %s\n",
```

A função main invoca a função applicationLayer com as especificações pré definidas e as de input, sendo aqui o ponto de partida para o resto do programa.

```
applicationLayer(serialPort, role, BAUDRATE, N_TRIES, TIMEOUT,
filename);
```

No nosso trabalho, é notório a separação do código referente ao emissor e ao recetor. Apesar de existirem funções utilizadas só pelo emissor e outras só pelo recetor, há também funções partilhadas pelos dois, em que o código utilizado por cada um é diferenciado por expressões condicionais.

Funções principais exclusivas do emissor	**Funções principais exclusivas do recetor**	**Funções partilhadas pelo emissor e pelo recetor**
▪ applicationTx ▪ writeCtrl ▪ llwrite ▪ configureDataPackage	▪ applicationRx ▪ receiveCtrl ▪ llread ▪ disconnect	▪ applicationLayer ▪ llopen ▪ stateStep ▪ writeCtrlFrame ▪ printBar ▪ llclose

A distinção entre camadas é percetível pela criação do ficheiro application_layer.c, que contém as funções pertencentes à camada de Aplicação, juntamente com o ficheiro link_layer.c, que agrupa as funções referentes à camada de Ligação de Dados.

Funções principais da camada de Ligação:

- **llopen()** - envia trama de supervisão SET e recebe trama UA
- **llwrite()** - efetua byte stuffing das I-frames e envia-as para o recetor
- **llread()** - recebe as I-frames, lê-as, e efetua byte destuffing
- **llclose()** - envia trama de supervisão DISC, recebe trama DISC e envia trama UA

Estruturas de dados utilizadas na camada de Ligação:

```
typedef enum Step {
    CONTINUE,  // the end of the frame has not yet been reached
    COMPLETE,  // the reading of the frame is complete
    REJECTED,  // the frame is rejected
    DUPLICATE, // the frame is a duplicate
    DISCONNECT // a DISC signal was sent
} Step;
```

```
typedef enum State {
    START,
    FLAG_RCV,
    A_RCV,
    C_RCV,
    BCC_OK,
    DATA,
    STOP
} State;
```

Variáveis globais da camada de Ligação:

```
static int fd; //porta série
static struct termios oldtio;
static struct termios newtio;

static LinkLayer parameters;

static int numTries;
static int alarmTriggered;

static int frameNumber = 0;
```

```
static int state = START;
static int is_disc = 0;
static unsigned char data[BUF_SIZE + 1] = {0};
static int data_idx = 0;
```

Funções principais da camada de Aplicação:

- **applicationTx()** - abre o ficheiro, separa-o em pacotes, envia-os dentro de tramas para o recetor e fecha o ficheiro
- **writeCtrl()** - constrói um pacote de controlo, para ser enviado numa I-frame

- **configureDataPackage()** - constrói um pacote de dados do ficheiro, para ser enviado numa I-frame
- **applicationRx()** - abre o ficheiro, lê os pacotes recebidos, envia os dados e fecha o ficheiro
- **receiveCtrl() -** extrai a informação de um pacote de controlo de uma I-frame

Estruturas de dados utilizadas na camada de Aplicação:

```
typedef enum
{
    LlTx,
    LlRx,
} LinkLayerRole;

typedef struct
{
    char serialPort[50];
    LinkLayerRole role;
    int baudRate;
    int nRetransmissions;
    int timeout;
} LinkLayer;
```

Variáveis globais da camada de Aplicação:

```
static LinkLayer parameters;
```

Macros pertinentes:

```

#define MAX_CHUNK_SIZE 128
#define MAX_PAYLOAD_SIZE 1000
#define BUF_SIZE 256

#define VERBOSE 0
#define FALSE 0
#define TRUE 1

#define BAUDRATE 9600
#define N_TRIES 3
#define TIMEOUT 4

#define FLAG 0x7E
#define ADDR_T 0X03
#define ADDR_R 0X01

#define SET 0X03
#define UA 0X07
#define DISC 0x0B
#define RR 0x05
#define REJ 0x01
```

```
#define I_CTRL_SHIFT 6
#define R_CTRL_SHIFT 7
```

Casos de uso principais

Como explicado anteriormente, o programa começa pelo main que chama o applicationLayer e a partir daí segue caminhos distindos dependendo se é o emissor ou recetor. A seguir promenoriza-se a sequência de chamada de funções para cada um dos casos possíveis.

Interface

A interface permite ao transmissor escolher o ficheiro a enviar. Foi criado um ficheiro Makefile que simplifica a chamada do programa, tendo como default as portas série /dev/ttyS10 e /dev/ttyS11, para o emissor e recetor, respetivamente. O ficheiro a enviar está definido como sendo o penguin.gif. Pode ser utilizado o comando 'make check_files' para ver se existem diferenças entre os dois ficheiros finais, assim como o 'make clean' para poder reutilizar os ficheiros. O utilizador, através da consola, correrá o programa com as configurações pré estabelecidas, caso não tenha sido necessário alterar algum parametro. Para compilar o programa começasse por criar um novo diretório bin através do comando 'mkdir bin', de seguida compila-se o programa com 'make'. Do lado do emissor, basta correr os comandos 'make run_tx' para compilar o programa como o transmissor do ficheiro pinguim.gif. Do lado do recetor, devem ser inserido o comando 'make run_rx' para compilar o programa como o recetor dos dados.

Transmissão de Dados

A transmissão de dados ocorre via porta de série, entre dois computadores, dando-se a seguinte sequência de eventos:

1. O utilizador, do lado do transmissor, escolhe o ficheiro a ser enviado (default do Makefile – pinguim.gif);
2. É configurada a ligação entre os dois computadores;
3. A ligação é estabelecida;
4. O transmissor envia os dados, trama a trama;
5. Simultaneamente, o recetor recebe os dados, trama a trama;
6. À medida que recebe os dados, o recetor guarda-os num ficheiro (default do Makefile –penguin-received.gif);
7. A ligação é terminada.

Especificação de casos

Caso I – Emissor

No caso do emissor, a função base chamada será o applicationTx, onde abrimos o ficheiro de onde vamos ler os dados para enviar ao recetor e logo depois começamos por fazer o set up: preenchemos a struct LinkLayer com os parametros definidos para o alarme e chamamos llopen para configurar a porta série através do openPort e iniciar a conexão. Aqui é enviado o SET frame através da função writeCtrlFrame, que configura todos os campos da

trama de Supervisão, e espera-se pela confirmação de recebimento, o UA frame. Após esta etapa, não ocorrendo nenhum erro, a conexão inicial está estabelecida.

O passo seguinte é iniciar a transferência de dados. Como estamos no caso do emissor, a função chamada será a applicationTx. Esta função abre o ficheiro do qual queremos ler os dados para enviar e começa por configurar o pacote de controlo através do writeCtrl para sinalizar o início da transmissão. Ainda dentro desta função é chamado o llwrite para enviar a trama de informação que contém o pacote de controlo configurado no passo anterior. A configuração do cabeçalho desta trama é efetuada pela função prepareWrite que também efetua o mecanismo de byte stuffing, invocando a função stuff do utils.c. Após todas estas etapas estarem concluídas, voltamos ao llwrite onde finalmente podemos enviar a trama que contem o pacote para o recetor e esperar pela receção da trama de confirmação. Ao receber uma trama sem erros e na sequência correta, estamos prontos para começar a enviar dados.

Como queremos iniciar a tranferência de dados, após o envio com sucesso do pacote de controlo inicial, voltamos a estar dentro do applictionTx e, enquanto houver dados para enviar, procedemos à configuração do pacote de dados para posterior envio. Para isso, chamamos a função configureDataPackage e, de seguida, invocamos o llwrite novamente, para enviar a trama de informação que contém o pacote de dados configurado no passo anterior. Após isto, é chamada a função printBar que vai imprimir para o terminal a quantidade de bytes que já foram enviados, em suma, por cada iteração. Este conjunto de intruções serão repetidas até que os dados a enviar acabem, ou caso exista algum erro na trasmissão das tramas.

Para sinalizar o término da transmissão, utiliza-se novamente a função writeCtrl que vai mais uma vez configurar o pacote de controlo. Tal como no inicio da transmissão, o llwrite é invocado para enviar a trama de informação contendo este pacote no seu campo de dados. Após toda a configuração do cabeçalho da trama e da passagem pelo mecanismo de byte stuffing, a trama é finalmente enviada e, desta forma, está assegurado o fim da transmissão de dados, pelo que o ficheiro já poderá ser fechado.

Por fim, voltando à função inicial applicationLayer, só nos resta invocar o llclose que, como estamos no caso do emissor, vai enviar o DISC frame através da função writeCtrlFrame e esperar pela confirmação de recebimento. Caso haja essa confirmação, o writeCtrlFrame é novamente invocado para enviar o UA frame e concluir a terminação da transmissão. Finalizando, as definições iniciais da porta série são restaurados e esta é fechada.

Caso II – Recetor

Já no caso do recetor, a função invocada será o applicationTx, onde se abre o ficheiro para onde vamos escrever os dados enviados pelo emissor e, seguidamente, a função setUp é novamente invocada para preencher a struct LinkLayer e chamar llopen para configurar a porta série com a função

openPort, iniciando a conexão. Como já explicado no emissor, é enviada a SET frame, pelo que, o recetor, nesta fase da conexão, vai esperar pelo seu envio e enviará uma trama de confirmação (UA frame) de volta para o emissor, caso não ocorra nenhum erro. Após este processo, o estabelecimento da conexão foi efetuado com sucesso.

Depois da conexão inicial, voltamos ao applicationRx para dar início à transferência de dados. Para isso, entramos num ciclo de instruções, que se irá repetir enquanto houver dados para ler da porta série. Pormenorizando, dependendo de algumas especificações, poderemos ler pacotes de controlo ou de dados. No caso de ser lido um pacote de controlo inicial, executam-se técnicas para armazenar o tamanho do ficheiro e o seu nome com a ajuda da função receiveCtrl, que serão usados para futura comparação como o pacote de controlo final e verificar se não ocorreu nenhum erro inesperado. Após a receção do pacote de controlo inicial, passamos à receção de pacotes de dados: depois de verificar se o pacote recebido é o suposto, ou seja, se não existiu perda de pacotes, os dados retirados do ficheiro com dados a enviar pelo emissor são agora escritos pelo recetor para o ficheiro respetivo. Sempre que é recebido um pacote de dados, é impresso para o terminal, através da função printBar, a quantidade de bytes que já foram escritos. No final, espera-se pela receção do pacote de controlo final, e, caso as informações sobre o ficheiro coincidam, o ficheiro é fechado sem nenhum erro ter ocorrido.

Por fim, voltando à função inicial applicationLayer, só nos resta invocar o llclose que, como estamos no caso do recetor, apenas vai restaurar as definições iniciais da porta série assim como a irá fechar. Desta forma, a terminação da trasmissão foi efetuada com sucesso.

Protocolo de ligação lógica

Neste protocolo, a transmissão é organizada em tramas, que podem ser de três tipos: Informação, Supervisão e Não Numeradas. Apenas as tramas de informação possuem um campo para transporte de dados (gerados pela aplicação), contudo todas elas têm um cabeçalho com um formato comum.

As tramas de Supervisão, assim como as Não Numeradas, são construidas pelo writeCtrlFrame.

```c
int writeCtrlFrame(int fd, unsigned char ctrl, unsigned char addr)
{
    unsigned char buf[BUF_SIZE + 1] = {0};

    buf[0] = FLAG;          // flag: 0x7E
    buf[1] = addr;          // campo de endereço: 0x03 ou 0x01
    buf[2] = ctrl;          // campo de controlo: SET, DISC, UA, RR, REJ
    buf[3] = addr ^ ctrl;   // campo de proteção de cabeçalho
    buf[4] = FLAG;          // flag: 0x7E
    buf[5] = '\0';

    return write(fd, buf, 5);
}
```

A delimitação de tramas é feita por meio de uma sequência especial de oito bits que designamos de flag (01111110 ou 0x7E). Uma trama pode ser iniciada com uma ou mais flags o que deve ser tido em conta pelo mecanismo de receção de tramas.

O mecanismo de byte stuffing está recriado no nosso código pelo uso das funções stuff e deStuff.

```c
/**
 * Stuff a sequence of bytes.
 * @param buf byte array to be stuffed.
 * @param bufSize size of the buffer.
 * @return size of the new buffer.
*/
int stuff(const unsigned char *src, unsigned char* dest, int bufSize);

/**
 * De-Stuff a sequence of bytes.
 * @param buf byte array to be de-stuffed.
 * @param bufSize size of the buffer.
 * @return size of the new buffer.
*/
int deStuff(unsigned char *dest, int bufSize);
```

Tal como seria esperado, stuff substitui a ocorrência da flag 0x7E pelo octeto de escape 0x7D seguido do byte 0x5E, assim como substitui o octeto de escape 0x7D pela sequência de dois bytes 0x7D 0x5D. O processo contrário é efetuado pelo deStuff que ao encontrar o octeto de escape 0x7D, verifica qual o byte que lhe sucede e, substitui os dois bytes pela flag 0x7E, caso o segundo byte seja 0x5E, ou pelo octeto de escape 0x7D, caso o segundo byte seja 0x5D. Assim, asseguramos a transparência garantindo que o protocolo garante a transmissão de dados independente de códigos.

A atribuição do endereço ao cabeçalho da trama vai variar da seguinte forma: em Comandos enviados pelo Emissor e Respostas enviadas pelo Recetor o campo de endereço é preenchido com o octeto 00000011 (ou 0x03); em Comandos enviados pelo Receptor e Respostas enviadas pelo Emissor é o octeto 00000001 (ou 0x01) que preenche esse campo.

É importante salientar que o campo de controlo recebe oito bit diferentes dependendo do tipo de trama.

SET (set up)	**0 0 0 0 0 0 1 1**	
DISC (disconnect)	**0 0 0 0 1 0 1 1**	
UA (unnumbered acknowledgment)	**0 0 0 0 0 1 1 1**	
RR (receiver ready / positive ACK)	**R 0 0 0 0 1 0 1**	
REJ (reject / negative ACK)	**R 0 0 0 0 0 0 1**	**R = N(r)**

Por isso, o parametro ctrl é chamado com 0x03, 0x0B e 0x07 para o SET, DISC e UA respetivamente, mas quando se refere a uma trama RR, por exemplo, o número da trama (R – pode ser 0 ou 1) em questão é deslocado sete bits para a esquerda, para ficar na posição correta, e é feito um OR bit a bit com 0x05, para o ctrl ficar com o formato R0000101. O processo idêntico é feito para o caso de ser uma trama REJ.

```
writeCtrlFrame(fd, RR | (!frameNumber << R_CTRL_SHIFT), ADDR_T);
```

As tramas são também protegidas por um código detetor de erros. Nas tramas de Supervisão e Não Numeradas, já que não existe o transporte de dados, existe apenas proteção simples da trama. Contudo, nas tramas de Informação existe proteção dupla independente, no cabeçalho e no campo de dados, o que permite usar um cabeçalho válido, mesmo que ocorra erro no campo de dados. No nosso trabalho, as tramas de Informação são estruturadas no prepareWrite e, abaixo, apresentamos um exemplo de como a proteção do campo de dados é efetuada.

```
int prepareWrite(const unsigned char* buf, unsigned char* dest, int bufSize) {
    unsigned char copy[BUF_SIZE + 1] = {0};
    unsigned char to_stuff[BUF_SIZE + 1] = {0};

    dest[0] = FLAG;
    dest[1] = ADDR_T;
    dest[2] = frameNumber << I_CTRL_SHIFT;
    dest[3] = ADDR_T ^ dest[2];

    unsigned char bcc = 0;
    // for loop does the parity over each of the bits of 'the data octets' and
the 'BCC'
    for (int j = 0; j < bufSize; j++) {
        bcc ^= buf[j];
        to_stuff[j] = buf[j];
    }
    to_stuff[bufSize] = bcc;

// continua no código..
```

Apesar de o processo de proteção ocorrer sobre o campo de dados destas tramas, os pacotes que aqui são transportados são considerados informação inacessível ao protocolo de ligação de dados. Nesta camada de ligação de

dados não é feito qualquer processamento que incida sobre o cabeçalho dos pacotes a transportar nestas tramas, pelo que não existe qualquer distinção entre tipos de pacotes.

É também neste protocolo que existe o controlo de erros em transmissões de tramas. Quando o recetor deteta erros numa trama, 'pede' ao emissor para a enviar de novo através do mecanismo Stop and Wait. Aqui, o emissor, após enviar uma trama de informação, espera um time-out pré definido (no nosso código definiu-se TIMEOUT = 4) pela confirmação positiva do recetor. Este, ao receber a trama de informação, verifica se a trama não tem erros, enviando uma trama RR ou REJ caso esta esteja livre de erros, ou não, respetivamente. O emissor, ao receber uma RR dentro do time-out, procede com a transmição de uma nova trama, caso receba uma REJ, envia de novo a mesma trama. Se, durante o time-out, o emissor não receber nenhuma trama de confirmação, vai enviar de novo a mesma trama, e efetua este procedimento um número máximo de vezes, estipulado por nós (definimos N_TRIES = 3), até obter a confirmação esperada. O controlo do time-out é feito pela função alarm e pelo booleano alarmTriggered que nos diz se o time-out está ou não ativado. No nosso trabalho, todo o processamento de tramas, incluindo a verificação de duplicados, ocorre na função stateStep do frame.c.

```
Step stateStep(unsigned char buf, unsigned char expected, unsigned
char addr)
```

Protocolo de aplicação

Como explicado anteriormente, ao nível da ligação de dados não existe qualquer distinção entre os pacotes que são enviados no campo de dados das tramas de Informação. Essa distinção e reconhecimento é feita ao nível da Aplicação, que diferencia os pacotes de controlo dos pacotes de dados.

Os pacotes de controlo são utilizados quer para sinalizar o início, quer o final da transferência do ficheiro. No nosso código estes pacotes são estruturados pelo writeCtrl, que tem como parametros o nome e o tamanho do ficheiro do qual queremos enviar os dados, assim como um valor inteiro que serve para diferenciar se é um pacote de controlo inicial ou final. A única diferença entre estes dois pacotes é o primeiro byte: se for um pacote de controlo inicial, o primeiro byte vai ter o valor 2, caso contrário terá o valor 3.

```
int writeCtrl(int fileSize, const char* fileName, int start) {
    unsigned char buf[BUFFER_SIZE] = {0};
    if (start == 1) buf[0] = 2;
    else buf[0] = 3;
```

O pacote de controlo que sinaliza o início da transmissão deverá ter obrigatoriamente um campo com o tamanho do ficheiro e opcionalmente um campo com o nome do ficheiro. No nosso caso, apesar de não ser obrigatório também adicionamos o campo com o nome do ficheiro. Começando pelo tamanho do ficheiro, a primeira coisa a fazer é colocar o próximo byte a 0, que significa que estamos a inserir informação relativa ao tamanho.

```
buf[1] = 0;     // fileSize -> mandatory
```

O byte seguinte vai indicar o numero de bytes que escrever o tamanho do ficheiro vai ocupar. Calculamos este valor como se fosse uma divisão do tamanho do ficheiro por 8 bits:

```
int numOct = 0, s = fileSize;
    while (s > 0) {
        numOct++;
        s = s >> 8;
    }
    buf[2] =  numOct; //specifies the size of the next field
```

O próximo passo será preencher o buffer com o valor do tamanho do ficheiro. Como são necessários 4 bits para representar um inteiro, cada byte vai ser preenchido com dois algarismos até o valor do tamanho do ficheiro ser representado na totalidade.

```
    int i = 0;
    // the for loop fills V1 with the parameter value
    for (; i < numOct; i++){
        buf[3 + i] = 0xff & (fileSize >> (8 * (numOct - i - 1)));
        // Example: 0x1234 -> buf[3]=12 e buf[4]=34
    }
```

Seguimos com o preenchimento do campo opcional que contém o nome do ficheiro. Para isso, colocomos o próximo byte a 1, para mostrar que vamos preencher um campo referente ao nome do ficheiro.

```
buf[i] = 1;      // fileName -> optional
```

Seguidamente, utilizamos a função strlen, da biblioteca string.h, para descobrir o tamanho que o nome do ficheiro ocupa, e desta forma, poder preencher o byte seguinte.

```
numOct = strlen(fileName);
    buf[i + 1] =  numOct; //specifies the size of the next field
```

Finalmente, preenchemos cada byte seguinte com os caracteres referentes ao nome do ficheiro.

```
i += 2;
    int j = 0;
    //the for loopfills V2 with the parameter value
    for (j = 0; j < numOct; j++){
        buf[i + j] = fileName[j];
    }
```

O pacote de controlo que sinaliza o fim da transmissão deverá repetir a informação contida no pacote de controlo inicial. No nosso trabalho, depois de todos os dados já terem sido enviados, a função writeCtrl é invocada com o parametro start a 0, para diferenciar do pacote inicial. Do lado do recetor, quando existe a receção de um pacote de controlo final, os seus campos de tamanho e nome do ficheiro são comparados com os mesmos campos do pacote inicial previamente recebido através da função receiveCtrl. Caso, em algum ponto, sejam diferentes verifica-se que ocorreu algum erro inesperado.

```
//compara se os pacotes de controlo end e start tem a mesma informação de file
size e name
if (fileSize != fileSizeEnd || strcmp(rcvFilename, rcvFilenameEnd) != 0) {
      printf("An error occurred.\n");
}
```

Os pacotes de dados contém fragmentos do ficheiro a transmitir. O conteúdo lido do ficheiro é colocado no campo de dados destes pacotes para poder ser enviado para o recetor. Estes pacotes são configurados no nosso código através da função configureDataPackage.

```
void configureDataPackage(unsigned char* buf, int pkgIndex, int size){
    unsigned char auxBuffer[BUF_SIZE] = {0};
    auxBuffer[0] = 1; //always 1 cause its a data package   // C
    auxBuffer[1] = pkgIndex % 255; //seqN mod with 255      // N
    auxBuffer[2] = size / 256;                              // L2
    auxBuffer[3] = size % 256;                              // L1

    memcpy(&auxBuffer[4], buf, MAX_CHUNK_SIZE); //buf tem o conteudo lido do
ficheiro
```

```
    memcpy(buf, auxBuffer, MAX_CHUNK_SIZE + 4);
    //buf tem agr os 4 parametros iniciais do pacote de dados + o conteudo lido do
ficheiro
}
```

O primeiro byte, assim como o primeiro byte do pacote de controlo, corresponde ao campo de controlo. Este valor é colocado a 1 para indicar que se trata de um pacote de dados. O segundo byte indica o número de sequência do pacote em questão. Este valor é único para cada pacote e é utilizado para detetar pacotes que foram perdidos. Os dois bytes seguintes indicam o número de octetos (K) do campo de dados do pacote, calculados através da fórmula K = 256 * L2 + L1.

Validação

De forma a verificar o comportamento do programa, os seguintes testes foram efetuados:

- Envio de vários ficheiros, com diferentes tamanhos
- Interrupção da ligação por cabo entre as portas de série
- Geração de ruído na ligação entre as portas de série
- Envio de ficheiros com diferentes taxas de simulação de erros
- Envio de ficheiros com diferentes valores de baudrate (capacidade de ligação)
- Envio de ficheiros com diferentes tamanhos para as I-frames
- Envio de ficheiros com a simulação de diferentes tempos de propagação de I-frames

Todos os testes que foram efetuados foram concluídos com sucesso, verificando-se o comportamento que era esperado.

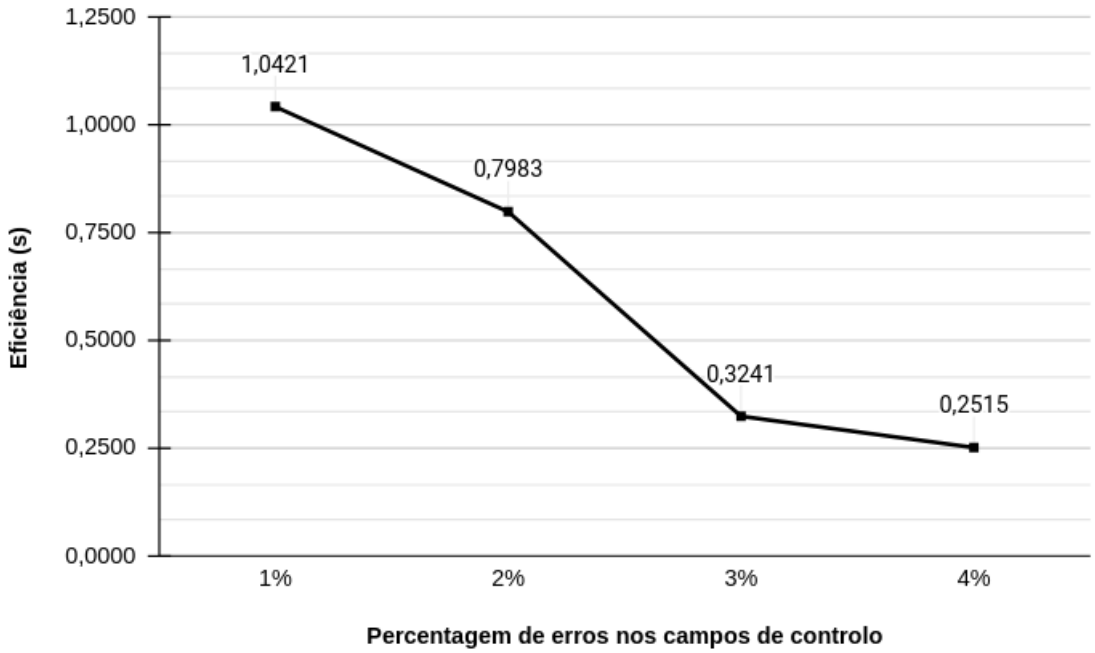
Eficiência do protocolo de ligação de dados

Para proceder à avaliação estatística da eficiência do protocolo, foram efetuados testes variando os seguintes parâmetros: tamanho da *frame* de informação, *baudrate*, FER e tempo de propagação.

Cada teste foi feito com diferentes valores e repetido seis vezes para cada valor. Foram utilizados ficheiros com 11kb e 1,4mb, dependendo do teste. Os valores obtidos podem ser consultados no Anexo III. Passamos a listar os resultados obtidos em cada teste

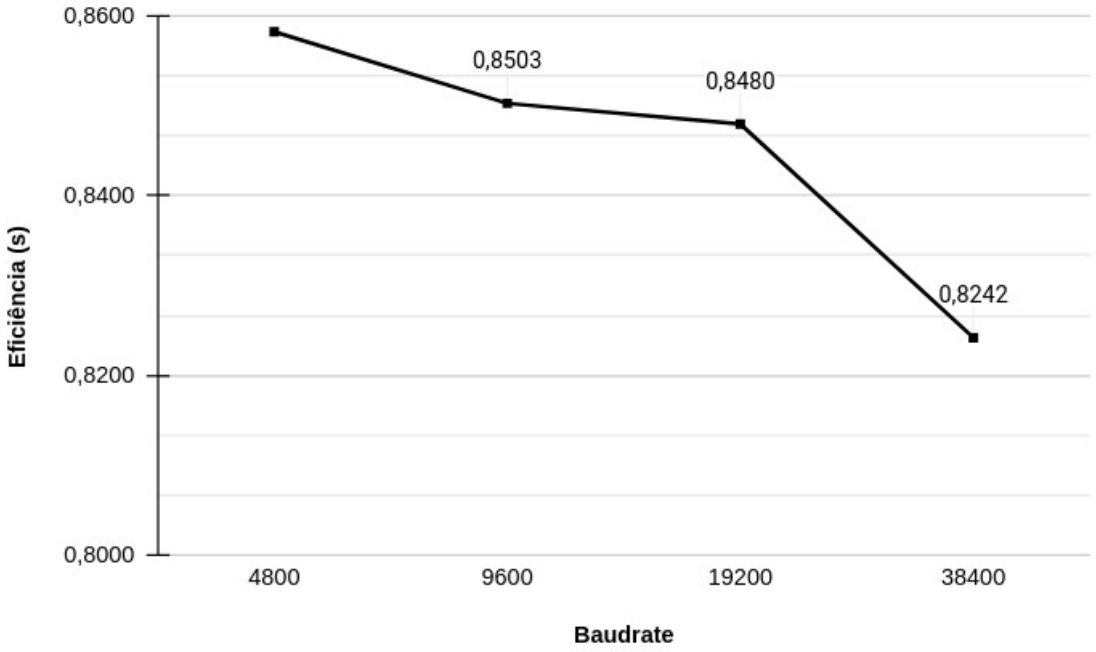
Variação do FER

Para avaliar a variação do FER, foram gerados erros aleatórios na validação dos campos BCC1 e BCC2. A probabilidade de erro variou entre 1% e 4%. À medida que a taxa de erro aumenta-se observa-se uma perda significativa na eficiência. De notar que os erros occorridos no BCC1 são de maior impacto, uma vez que o programa irá ficar estacionário por 3 segundos, devido ao atraso de retransmissão.



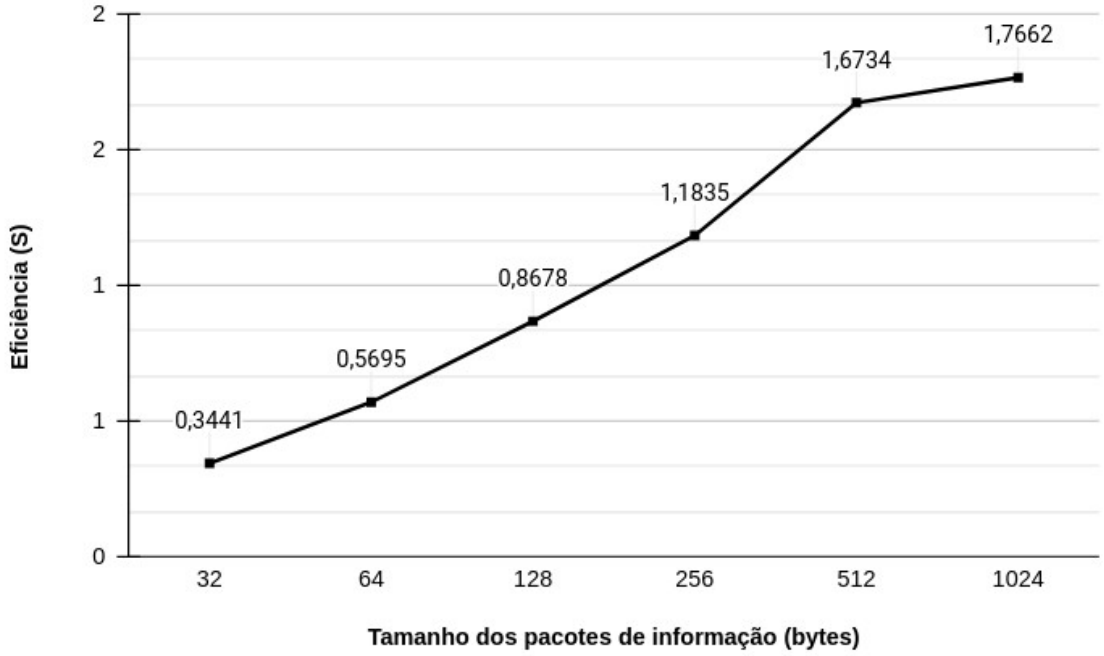
Variação do baudrate

Com a variação da capacidade da ligação, fez-se notar uma ligeria queda de eficiência à medida que a primeira aumentava.



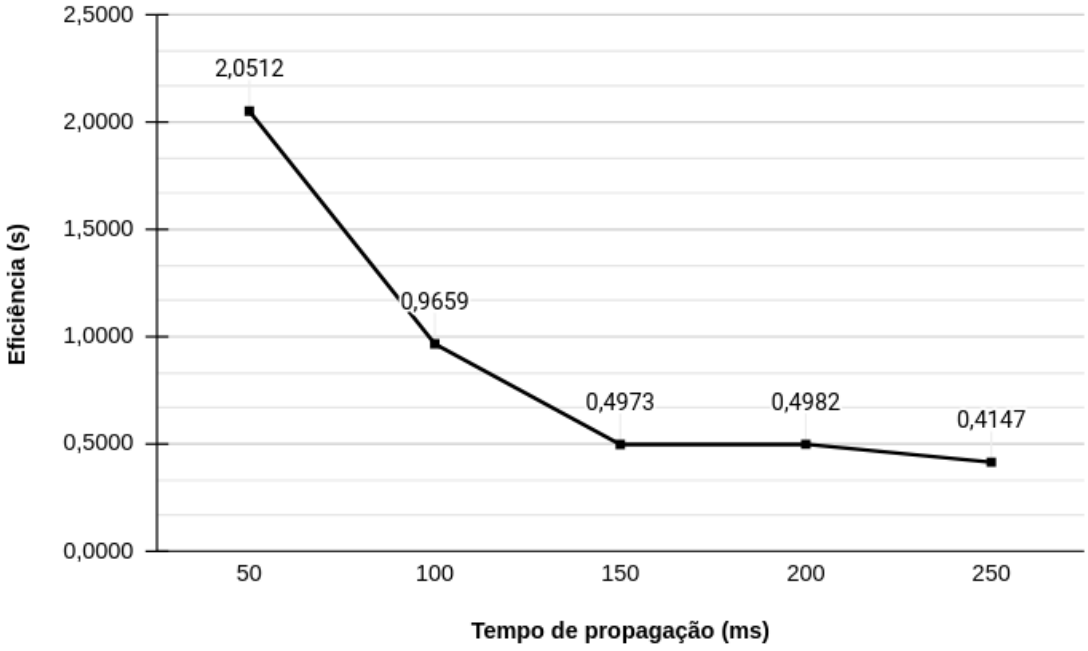
Variação do tamanho da trama de informação

Procedeu-se à variação do tamanho da trama de informação. Estes valores estão compreendidos entre 32 e 1024 bits e registou-se um aumento significativo da eficiência à medida que o tamanho também aumentava. Isto acontece porque o número de frames a enviar diminui.



Variação do tempo de propagação

Para testar a variação do tempo de propagação recorremos ao uso da função *usleep*, de modo a simular delays de 50ms a 250ms. Como seria de esperar, o aumento do tempo de propagação provoca o consequente aumento do tempo total. Por sua vez, este causa uma queda significativa da eficiência.



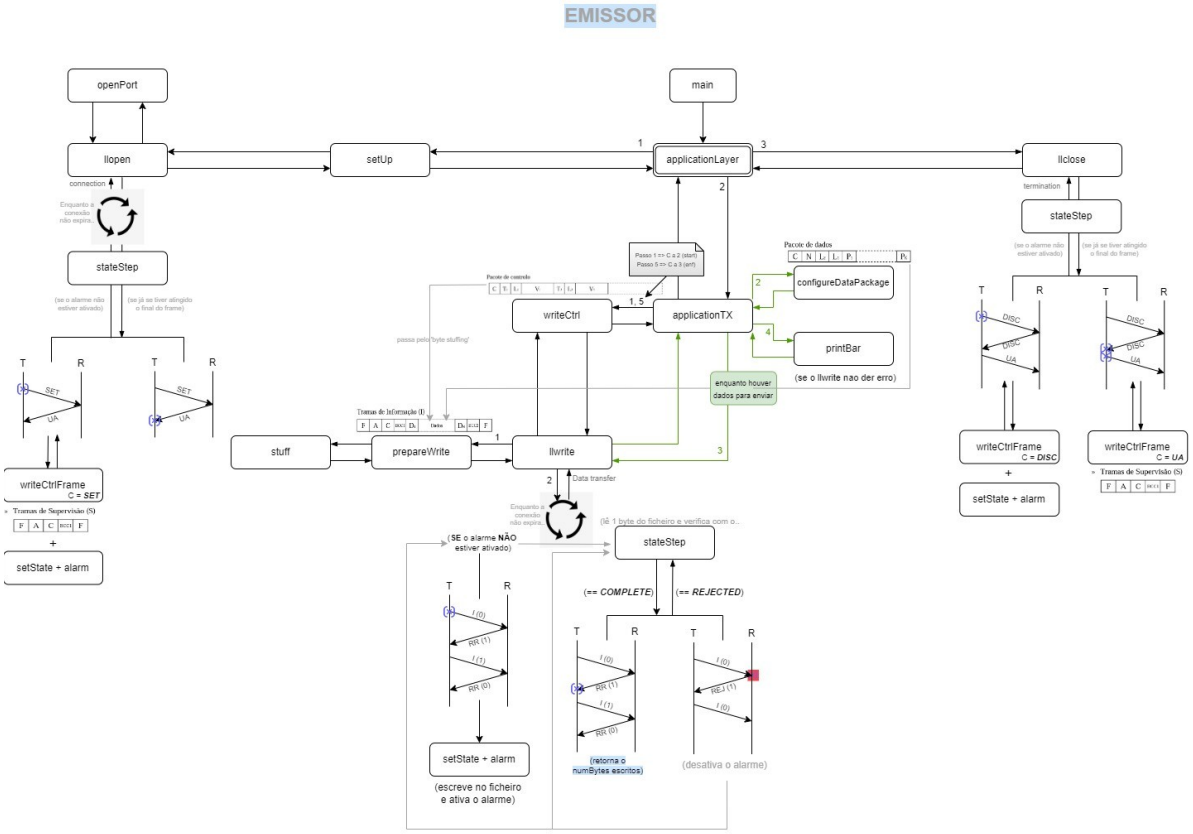
Conclusões

O projeto desenvolvido consiste no desenvolvimento de um serviço fiável de comunicação entre dois computadores ligados por porta de série, implementando o protocolo de ligação de dados.

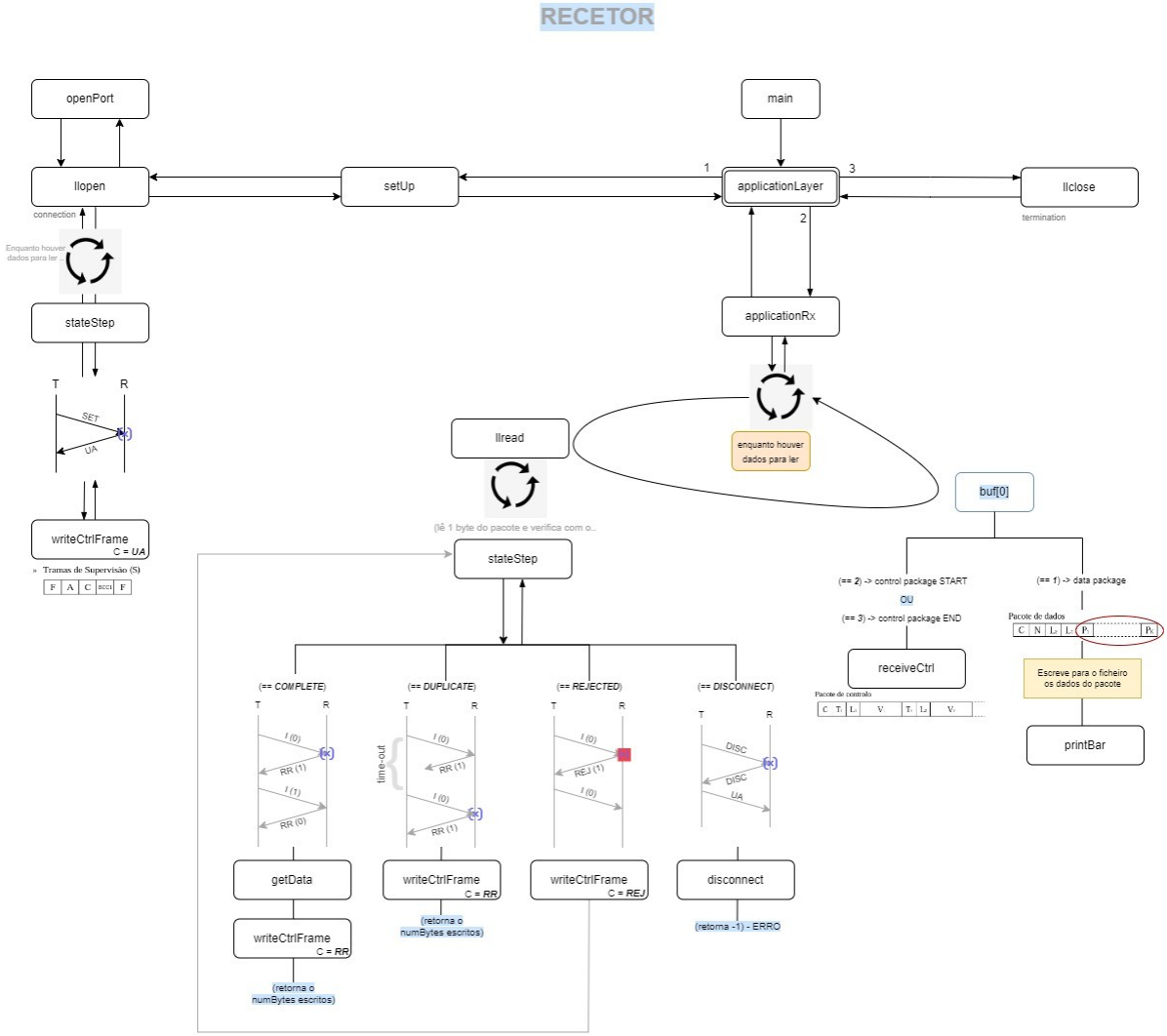
Este trabalho permitiu-nos compreender o protocolo de ligação de dados, incidindo sobre a estrutura das tramas e o processo de encapsulamento, envio e receção da informação. Adicionalmente, é destacada a importância da independência entre camadas, sendo que cada camada do programa respeita esse conceito. A camada da ligação de dados não recorre, de todo, a qualquer processamento desenvolvido na camada da aplicação. Já a camada da aplicação não conhece quaisquer detalhes da implementação da camada da ligação de dados, conhecendo apenas como utilizar as suas funcionalidades.

Anexos

Anexo I – Diagrama do Emissor



Anexo II – Diagrama do Recetor



Anexo III – Tabelas de teste à eficiência

FER	1%	2%	3%	4%
	8,072594	8,127758	12,06892	48,418951
	8,058056	32,264496	44,387972	16,165163
	12,216453	4,012123	40,144733	24,160097
	4,011144	4,12435	20,193748	40,401125
	8,022378	16,161828	32,340078	48,431417
	12,24514	4,007624	20,047998	40,432621
Average	8,770960833	11,4496965	28,1972415	36,33489567
Efficiency	1,0421	0,7983	0,3241	0,2515

Baudrate	9600
File Size	10968

Baudrate	4800	9600	19200	38400
	1,411352	1,400716	1,477531	1,475694
	1,310346	1,433811	1,41834	1,337631
	1,409528	1,372042	1,311109	1,452731
	1,30627	1,313455	1,28983	1,340894
	1,444519	1,404949	1,320933	1,496645
	1,320255	1,354181	1,483933	1,437757
Average	1,367	1,380	1,384	1,424
Efficiency	0,8583	0,8503	0,8480	0,8242

Baudrate	9600
File Size	1407923

Frame Size	32	64	128	256	512	1024
	3,222251	2,057929	1,261869	0,988753	0,706224	0,628094
	3,394638	2,136192	1,353617	0,985799	0,698867	0,572158
	3,371139	2,023705	1,317333	1,078165	0,693328	0,683471
	3,533823	2,089486	1,458251	0,959803	0,689784	0,694546
	3,518483	2,058139	1,330041	0,972185	0,719854	0,692152
	3,417315	1,995779	1,391035	0,963359	0,698742	0,715382
Average	3,41	2,06	1,35	0,99	0,70	0,66
Efficiency	0,3441	0,5695	0,8678	1,1835	1,6734	1,7662

Baudrate	9600
File Size	1407923

Propagation Time	50	100	150	200	250
	4,464784	9,681815	18,370770	18,297781	22,045150
	4,459632	9,699408	18,411092	18,362556	22,052030
	4,461643	9,380113	18,380381	18,439476	22,062442
	4,443123	9,469367	18,331376	18,343252	22,037051
	4,456426	9,164107	18,440507	18,292676	22,027782
	4,450523	9,379134	18,337967	18,332304	22,024180
Average	4,4560	9,4623	18,3787	18,3447	22,0414
Efficiency	2,0512	0,9659	0,4973	0,4982	0,4147

Baudrate	9600
File Size	10968