Faculdade de Engenharia da Universidade do Porto

# Data Link Layer Protocol


1º Trabalho Laboratorial

Ana Sofia Teixeira
Rafael Cardoso Magalhães

L/M em Engenharia Electrotécnica e de Computadores
Redes de Computadores, 2021/2022, 2º Semestre

# Índice

# Sumário

Este relatório tem como objectivo aplicar os conhecimentos adquiridos na Unidade Curricular de Redes e Computadores ao longo da primeira metade do semestre. A realização deste trabalho laboratorial consiste na transferência de dados de um computador para outro através de uma ligação por cabo pelas portas séries de cada máquina.
O projeto foi demonstrado com sucesso, tendo sido realizada a transferência de dados eficazmente. Quando perdida da ligação, o programa foi capaz de restabelecer a transmissão e recuperar os dados.

# 1. Introdução

O objetivo deste trabalho é implementar um protocolo de comunicação de dados entre dois computadores utilizando um cabo série, para tal foram testados protocolos de encapsulamento usando uma aplicação para a transferência de dados. Este tipo de ligação serve para entender como é realizada a transferência de informação em baixo nível. O objetivo do relatório é a examinar a componente teórica do trabalho realizado através dos seguintes tópicos:
• **Introdução:** indicação dos objetivos do trabalho e do relatório;
• **Arquitetura:** explicação do funcionamento dos blocos funcionais e interfaces;
• **Estrutura do código:** tópico em que são mencionadas principais estruturas de dados, principais funções e sua relação com a arquitectura;
• **Casos de uso principais:** identificação da sequência de chamada de funções;
• **Protocolo de ligação lógica:** identificação dos principais aspetos funcionais, descrição da estratégia de implementação destes aspetos com apresentação de extratos de código;
• **Protocolo de aplicação:** explicação da lógica de implementação seguida nesta fase do trabalho;
• **Validação:** descrição dos testes realizados e resultados obtidos com recurso a imagens ilustrativas;
• **Elementos de Valorização:** identificação e explicação de alguns elementos de valorização aplicados;
• **Conclusão:** reflexão sobre os pontos anteriormente mencionados e sobre os objetivos de aprendizagem alcançados.

# 2. Arquitectura

A aplicação está organizada em duas camadas bem definidas. A camada de ligação de dados é a responsável pelo estabelecimento de ligação (protocolo da camada de ligação de dados) e a camada da aplicação.

É feita na camada de ligação a interação com a porta de série, sendo feita a abertura, fecho, escrita e leitura, tratando das necessidades de comunicação, como tratamento de erros e stuffing e destuffing de pacotes.

A camada da aplicação serve como ponte entre o utilizador e a camada de ligação de dados, sendo a camada lógica situada acima dessa. Faz uso da interface da camada de

ligação de dados, chamando as suas funções para o envio e receção de ficheiros a receber e enviar.

# 3. Estrutura do código

O código que foi realizado está dividido entre funções que interagem com a porta série e funções auxiliares chamadas na camada de protocolo.

Nas funções que interagem com a porta série, temos a **llopen**, **llclose**, a **llread** e a **llwrite**.

A função **llopen** é utilizada pelo recetor e pelo transmissor, assim como o **llclose,** que têm como objetivo abrir (llopen) e fechar (llclose) a ligação entre o transmissor e o recetor. Através da variável global **role**, estas duas funções distinguem os papéis a serem executados pelo transmissor e recetor. A função **llwrite** é utilizada pelo transmissor para enviar dados e **llread** pelo recetor para receber dados.

No grupo das funções auxiliares encontram-se as funções **enviaTrama**, **recebeTrama**, **stuffing**/**destuffing**, **remove_extra** e **BCC2_ver**. A função **enviaTrama** tem como objetivo enviar tramas de todo o tipo, e é utilizada pelo transmissor e recetor. A função **recebeTrama** tem como objetivo receber tramas e identificar o byte de controlo para identificar o pacote recebido. A função **stuffing** tem como objetivo substituir bytes na mensagem para que o pacote seja recebido pelo emissor sem gerar conflito, garantindo a transparência no envio de pacotes. A função **destuffing**, por sua vez, reverte a substituição de bytes realizada no **stuffing,** garantindo a restauração dos dados iniciais. O **BCC2_ver** é utilizado para detetar erros no envio de dados. Antes da mensagem ser enviada, calcula-se o BBC2 e guarda-se o valor na penúltima posição da trama. Quando a mensagem é recebida, e após o **destuffing**, calcula-se novamente o BCC2 da mensagem, e se coincidir com o BCC2 guardado na penúltima posição da trama, a mensagem foi enviada com sucesso.

# 4. Casos de uso principais

**Envio de Tramas**

O protocolo de envio de tramas varia de acordo com o tipo de trama que o utilizador pretende enviar.

**Envio de Tramas de Controlo**

Quando se pretende abrir ou fechar a ligação entre o transmissor e o recetor, é necessário enviar tramas de controlo.

O processo inicia-se com a criação de um vetor de 5 bytes. Para isso, foi criada a função **trama_ready**, que tem como argumentos: o campo de controlo que é necessário preencher no vetor(que é o campo que identifica o tipo de trama a enviar e a sua função), e o vetor a preencher.

Após a criação da trama, copiou-se a trama para um vetor global, utilizando a função **copyToPacketToSend**, para facilitar o uso de tramas na função **enviaTrama**.

A última etapa deste processo é chamar a função **enviaTrama,** que varia caso seja recetor ou transmissor. Se for recetor, apenas envia para a porta série o vetor global (trama

guardada anteriormente). Se for transmissor, para além de enviar, monitoriza quantas tentativas de envio fez, de forma a detetar erros no envio do pacote.

**Envio de Tramas de Dados**

Quando a ligação entre o transmissor e o recetor está estabelecida, procede-se ao envio de tramas de dados. O processo de envio de uma trama de dados é parecido com o envio de tramas de controlo, havendo apenas algumas alterações.

Inicia-se por preencher o vetor com 5 bytes, utilizando-se o mesmo procedimento das tramas de controlo. Após estes 5 bytes, faz-se **stuffing** da mensagem que se quer enviar, e preenche-se o vetor com a mensagem "stuffed".

Continua-se o processo, preenchendo o resto do vetor com a variável BCC2, que permite detetar erros na receção da mensagem pelo recetor. seguido de uma FLAG, de forma a detetar o fim do pacote enviado.

Conclui-se, utilizando as mesmas funções que se utiliza nas tramas de controlo, de forma a enviar com sucesso o pacote.

**Receção de Tramas**

A receção de tramas é diferente dependendo do tipo de trama enviado pelo transmissor.

Quando se recebe uma trama de controlo, através de uma máquina de estados, apenas se guarda o campo de controlo da trama recebida para conseguir identificar aquilo que o transmissor pretende com o envio da trama.

Quando se recebe uma trama de dados, o recetor tem de isolar a mensagem, fazer **destuffing**, calcular o BCC2 da mensagem "destuffed" e comparar com o BCC2 enviado no pacote, de forma a encontrar possíveis erros no envio da mensagem.

Em ambos os casos, utilizam-se máquinas de estados, e a leitura é feita byte a byte, alterando o estado de acordo com o byte lido, de forma a reconhecer erros de envio, e ser possível implementar um timer, que apenas é desligado quando se chega ao último estado. Se não chegar ao último estado, dá-se o timeout.

# 5. Protocolo de ligação lógica

A camada de ligação de dados é a responsável pela interação direta com a Porta Série e é nesta camada que se estabelece a ligação entre os dois terminais do cabo.

Após a abertura da ligação, a camada de aplicação utiliza um header e um trailer para delimitar a trama, de forma a facilitar a identificação do conteúdo da mensagem.

Para além disso, esta camada tem mecanismos de deteção de erros. O transmissor, só após a confirmação de receção por parte do recetor, é que procede ao envio de outro pacote. Para além disso, para não existir duplicação de tramas, existe um controlo através dos números de série das tramas.

As principais funções desta camada são:

**int llopen**

Através desta função, é estabelecida a ligação entre o transmissor e o recetor. Depois de aberta a ligação, o transmissor envia uma trama SET e espera que o recetor envie a trama UA. A diferença entre estas duas tramas de controlo é o seu campo de

controlo C. As tramas são enviadas através do procedimento descrito acima (**Envio de Tramas de Controlo)**. A função retorna -1 se existir algum erro e TRUE se a ligação for estabelecida com sucesso.

**int llwrite**

A função llwrite tem como função enviar as tramas de informação do transmissor para o recetor. Como descrito acima(**Envio de Tramas de Dados**), começa por criar um vetor e preencher as primeiras 5 posições com 5 bytes, utilizando a função **trama_ready**, variando o campo de controlo consoante o número de série da trama. Após estes 5 bytes, chama-se a função stuffing, que copia a mensagem que se pretende enviar para o vetor, procedendo a substituir alguns bytes da mensagem se for necessário. Por fim, adiciona um campo BCC2, que consiste num XOR entre os bytes da mensagem, de forma a ter um campo de controlo da mensagem, para efetuar a avaliação da transparência do envio. Em caso de erro, o transmissor procede ao reenvio da trama até ao número máximo definido pelo utilizador. Retorna o tamanho final da trama I, que corresponde ao tamanho da mensagem inicial, mais os bytes do header e do trailer.

**int llread**

Esta função é usada pelo recetor e tem como objetivo receber as tramas enviadas pelo transmissor. Através de uma máquina de estados, lê os primeiros 5 bytes de controlo da trama, e depois começa a armazenar os dados seguintes até encontrar uma FLAG(fim de trama). Chama a função **remove_extra**, para remover o trailer, ficando apenas com a mensagem enviada, embora “stuffed”. Como tal, chama a função **destuffing** para obter a mensagem original e calcula o BCC2 através de **BCC2_ver.** Se o valor obtido neste cálculo for igual ao byte de controlo presente no trailer, o envio da mensagem teve sucesso. A função retorna o tamanho da mensagem recebida.

**int llclose**

Esta função fecha a ligação entre o transmissor e o recetor. O transmissor inicia por enviar uma trama DISC, ficando à espera da resposta do recetor, que também enviará uma trama DISC. Se o transmissor receber a trama do recetor, envia a trama UA e termina-se a ligação.

# 6. Protocolo de aplicação

A camada da aplicação serve como ponte entre o utilizador e a camada de ligação de dados, sendo a camada lógica situada acima desta, é feita a interação com o utilizador e onde são recolhidos os dados introduzidos.
A camada de aplicação está dividida em duas partes essenciais, a parte destinada às instruções para o transmissor e a parte com as instruções para o receptor.
Em ambos os modos, inicialmente é criada uma variável do tipo linkLayer onde são guardados parâmetros da ligação, a identificação do modo, baudRate e o número de timeouts.

```c
typedef struct linkLayer{
    char serialPort[50];
    int role; //defines the role of the program: 0==Transmitter, 1=Receiver
    int baudRate;
    int numTries;
    int timeOut;
} linkLayer;
```

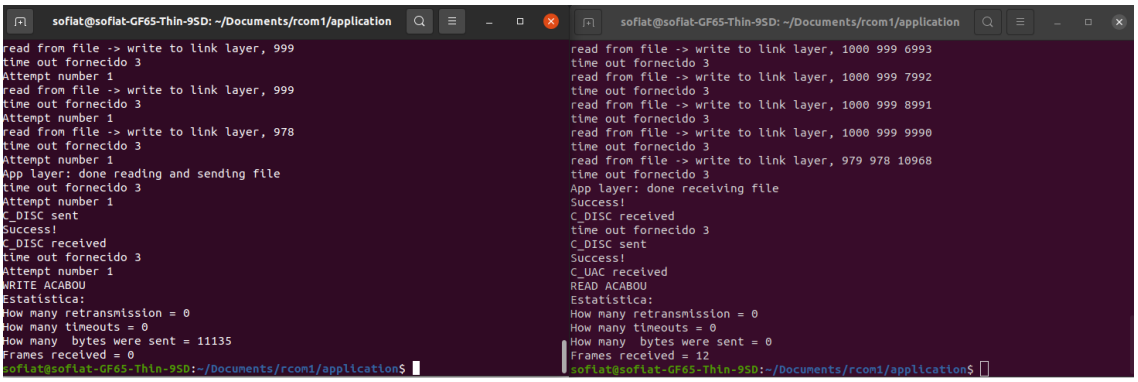
No modo transmissor (0) é aberto o ficheiro e é criado um ciclo que vai inserindo blocos de dados lidos deste ficheiro num buffer com um tamanho definido. Enquanto conseguir ler dados do ficheiros, esses buffers vão sendo enviados para o receptor com a função llread.

No modo receptor (1), cria-se o ficheiro onde os dados enviados pelo transmissor vão ser guardados, neste caso corresponde à imagem a receber. Depois a função llwrite é chamada num ciclo onde, enquanto receber blocos de dados, os vai guardando no ficheiro criado para o efeito.
No final de ambos os modos, é chamada a função llclose para encerrar a ligação.

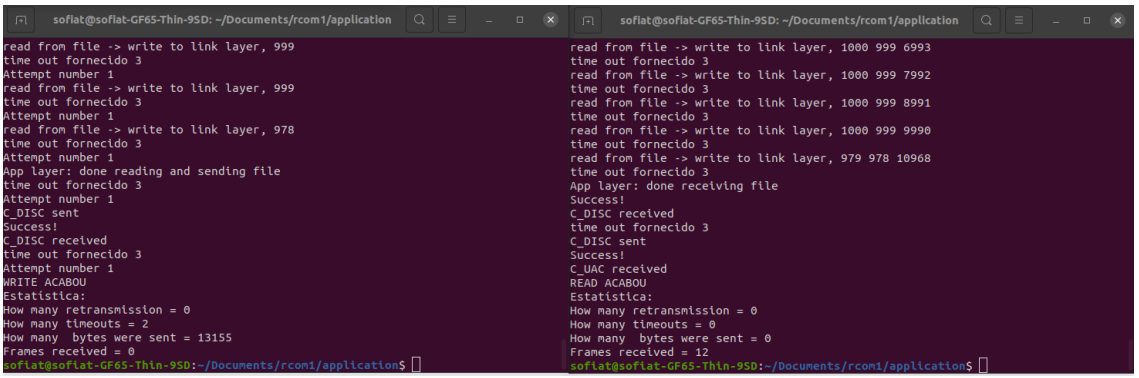
# 7. Validação

Foram efetuados essencialmente testes à nossa aplicação, à esquerda apresenta-se o emissor (Tx) e à direita o receptor (Rx):

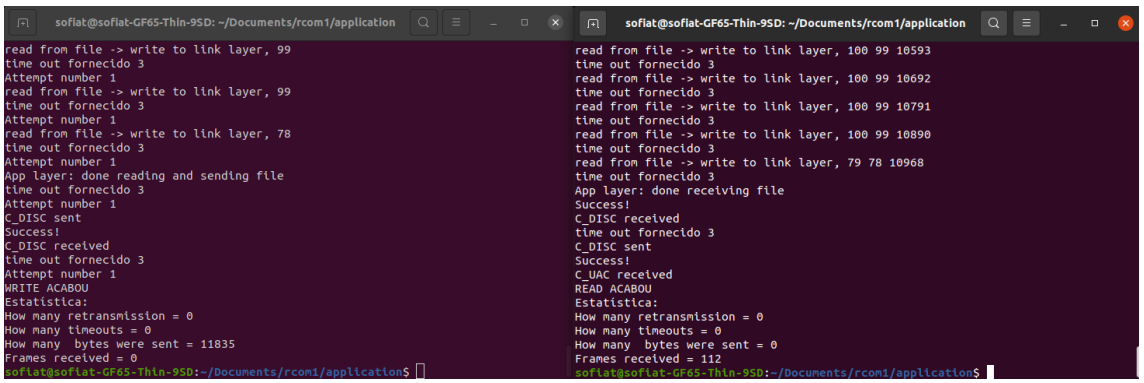
- Envio do ficheiro penguin.gif sem introdução de ruído ou interrupções:



- Envio do ficheiro penguin.gif com introdução de ruído com interrupção:




- Envio do ficheiro penguin.gif com introdução de ruído com redução do "MAX_PAYLOAD_SIZE" para 100:




# 8. Elementos de valorização

**Implementação de REJ:** Quando o BCC2 calculado pelo recetor na função **remove_extra,** é diferente do byte de controlo BCC2 presente no trailer, é enviado o comando REJ, para que o transmissor reenvie a trama pretendida.

**Estatísticas do ficheiro transmitido:** É calculado, ao longo do código, o número de timeouts, de REJs enviados/recebidos e retransmissões de tramas I.

# 9. Conclusões

A implementação deste protocolo de ligação de dados permitiu o uso em prática de conceitos como stop and wait, conceitos teóricos como o funcionamento de uma ligação em série e a comunicação entre as camadas de ligação e a sua aplicação e implementação de forma independente. A nossa implementação teve em conta todos esses aspetos e outros de correção de erros e interrupções na ligação.

Concluindo, podemos afirmar que o projeto foi concluído com sucesso, uma vez que foram cumpridos os objetivos pretendidos quer de enviar e receber os dados corretamente, contemplando os cenários de falha previstos.

# ANEXO I - Ficheiro LinkLayer.c

```
#include "linkv2.h"
#include "linklayer.h"

int fd;
unsigned char C_FIELD_aux = C00;
int COMPLETO;
int LENGTH;
int tentativas = 0;
unsigned char * CopiedPacket;
unsigned char SET[5];
unsigned char UA[5];
unsigned char DISC[5];
unsigned char generic[5];
linkLayer new;
Statistics stats;

/* Send Packet
 * envia todo o tipo de tramas e é utilizada pelo transmissor e pelo recetor.
 */

void enviaTrama()
{
		COMPLETO = FALSE;
		int time = 0;

		printf("time out fornecido %d\n", new.timeOut);

		if(tentativas > 0)
			(stats.timeouts)++;

		if( new.role == TRANSMITTER )
		{
			if(tentativas == new.numTries)
			{
				printf("TIMEOUT: Failed 3 times, exiting...\n");
				exit(-1);
			}
			printf("Attempt number %d\n", tentativas+1);

			int BytesSent = 0;

			while( BytesSent != LENGTH)
			{
				BytesSent = write(fd, CopiedPacket, LENGTH);
				stats.bytes_sent+=LENGTH;
			}
```

```
                        tentativas++;

                        if(!COMPLETO)
                        {
                                alarm(new.timeOut);
                        }
                }

                        else if( new.role == RECEIVER )
                        {
                                int BytesSent = 0;

                                while( BytesSent != 5 )
                                {
                                        BytesSent = write(fd, CopiedPacket, 5);
                                }
                        }

                        else
                        {
                                printf("ERROR\n");
                                exit(-1);
                        }

}



void trama_ready(unsigned char * array, unsigned char C)
{
        array[0] = FLAG;
        array[1] = A;
        array[2] = C;
        array[3] = array[1]^array[2];
        array[4] = FLAG;
}


/* recebeTrama
 * receção de tramas de controlo
 */

unsigned char recebeTrama(int fd)
{
        unsigned char car, res, Cverif;
        int state=0;
```

```
        while( state != 5 )
        {
res = read(fd,&car,1);
if( res < 0)
{
  printf("Read falhou.\n");
  return FALSE;
        }

                switch(state)
                {
                        case 0: //expecting flag
                                if(car == FLAG)
                                        state = 1;
                                break;

                        case 1: //expecting A
                                if(car == A)
                                        state = 2;

                                else if(car != FLAG)
                                {
                                        state = 0;
                                }

                                break;

                        case 2: //expecting Cverif
                                Cverif = car;
                                if(car == C_UA || car == C_DISC || car == C_SET)
                                        state = 3;

                                else if(car != FLAG)
                                {
                                        state = 1;
                                }

                                else
                                        state = 0;

                                break;

                        case 3: //expecting BCC
                                if(car == (A^Cverif))
                                        state = 4;

                                else
                                {
```

```
                                        state = 0;
                                }

                                break;

                        case 4: //expecting FLAG
                                if(car == FLAG)
                                        state = 5;

                                else
                                {
                                        state = 0;
                                }

                                break;
                        }
                }

                COMPLETO = TRUE;
                alarm(0);
                printf("Success!\n");
                tentativas = 0;

                return Cverif;
}

/* llopen
 *
 */


int llopen(linkLayer connectionParameters)
{
        signal(SIGALRM, enviaTrama);

        printf("%s\n", connectionParameters.serialPort);
        fd = open(connectionParameters.serialPort, O_RDWR | O_NOCTTY);

        if( fd < 0 )
        {
                printf("Error at opening connection!\n");
                return NOT_DEFINED;
        }

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

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

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

newtio.c_cc[VTIME]    = 0;   /* inter-character timer unused */
        newtio.c_cc[VMIN]     = 1;   /* blocking read until 5 chars received */

tcflush(fd, TCIOFLUSH);

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

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

                CopiedPacket = (unsigned char *) malloc(sizeof(unsigned char));

        new = connectionParameters;

        stats.retransmissions = 0;
        stats.timeouts = 0;
        stats.I_Frame_Received = 0;

        switch(connectionParameters.role)
        {
                case TRANSMITTER:
                        trama_ready(SET, C_SET);

                        copyToPacketToSend(SET, 5);

                        enviaTrama();
                        printf("C_SET enviado\n");

                        if(recebeTrama(fd) == C_UA)
                                printf("C_UA recebido\n");
                        break;

                case RECEIVER:
                        if(recebeTrama(fd) == C_SET){
                                printf("C_SET recebido\n");
                                trama_ready(UA, C_UA);
```

```
                        copyToPacketToSend(UA, 5);

                        enviaTrama();
                        printf("C_UA enviado\n");
                    break;
                    }

        }

        printf("ligação estabelecida\n");

        return TRUE;
}

/* llclose
 *
 */


int llclose(int showStatistics)
{
                switch(new.role)
                {
                        case TRANSMITTER:
                                trama_ready(generic, C_DISC);
                                copyToPacketToSend(generic, 5);
                                enviaTrama();
                                printf("C_DISC sent\n");

                                if(recebeTrama(fd) == C_DISC)
                                        printf("C_DISC received\n");

                                        trama_ready(UA, C_UA);
                                        copyToPacketToSend(UA, 5);
                                        enviaTrama();

                                        printf("WRITE ACABOU\n");

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

                        case RECEIVER:
                                if(recebeTrama(fd) == C_DISC){
                                        printf("C_DISC received\n");
```

```
                                        trama_ready(generic, C_DISC);
                                        copyToPacketToSend(generic, 5);
                                        enviaTrama();
                                        printf("C_DISC sent\n");

                                        if(recebeTrama(fd) == C_UA)
                                                printf("C_UAC received\n");

                                                printf("READ ACABOU\n");

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

                }

                printf("Estatística:\n");
                printf("How many retransmission = %d\n", stats.retransmissions);
                printf("How many timeouts = %d\n", stats.timeouts);
                printf("How many  bytes were sent = %d\n", stats.bytes_sent);
                printf("Frames received = %d\n", stats.I_Frame_Received);

                return 1;
}

/* llwrite
 *
 */


int llwrite(char * buf, int size){

        int sizeNewMsg = size + 6;
        int BytesSent = 0;
        int retransmission = -1;

        unsigned char *newMsg = (unsigned char *)malloc(sizeof(unsigned char) * (size+6));

        unsigned char BCC2;
        int sizeBCC2 = 1;
        BCC2 = BCC2_ver(buf, size);
```

```
	newMsg[0] = FLAG;
	newMsg[1] = A;
	newMsg[2] = C_FIELD_aux;
	newMsg[3] = newMsg[1]^newMsg[2];

	sizeNewMsg = stuffing(buf, newMsg, size, sizeNewMsg);

	newMsg[sizeNewMsg-2] = BCC2;
	newMsg[sizeNewMsg-1] = FLAG;

	unsigned char readChar;

	copyToPacketToSend(newMsg, sizeNewMsg);

	do
	{
			retransmission++;

			if( retransmission > MAX_RETRANSMISSIONS_DEFAULT )
			{
				stats.retransmissions =
MAX_RETRANSMISSIONS_DEFAULT;
					printf("Atingiu limite de retransmissões\n");
					return -1;
			}

			enviaTrama();
			readChar = getCmdExpectingTwo(correctREJ(newMsg[2]),
correctRR(newMsg[2]));

	} while( readChar == correctREJ(newMsg[2]) );

	(stats.retransmissions)+=retransmission;

	complementC();

	free(newMsg);

	return sizeNewMsg;

}

/* llread
 *
 */


int llread(char *packet)
```

```
{

        int size=0, n, res;
        unsigned char *auxbuff = (unsigned char *) malloc(size*sizeof(char));
        unsigned char ctrl_aux;
        unsigned char c;
        int state = 0;
        int i;

  while (state != 5)
  {
                res = read(fd, &c, 1);

    if(res < 0)
                {
                        perror("llread");
                        return FALSE;
                }

    switch (state)
    {
    //flag
    case 0:
      if (c == FLAG)
        state = 1;
      break;
    // A
    case 1:
      if (c == A)
        state = 2;
      else
      {
        if (c == FLAG)
          state = 1;
        else
          state = 0;
      }
      break;
    // C
    case 2:

      if (c == C00 || c == C10)
      {
                                ctrl_aux= c;
        state = 3;
      }

      else
```

```
    {
      if (c == FLAG)
        state = 1;
      else
        state = 0;
    }
    break;
  //BCC
  case 3:
    if (c == (A ^ ctrl_aux))
                        {
                                state = 4;
                                i = 0;
                        }
    else
      state = 0;
    break;
  //segunda FLAG
  case 4:
                ++size;
                auxbuff = (unsigned char *) realloc(auxbuff, size*sizeof(unsigned char));
                auxbuff[i] = c;

                if(c == FLAG){

                        n = remove_extra(auxbuff, ctrl_aux, size);

                        if(n != -1){
                                for (i = 0; i < n; i++) {
                                        packet[i] = auxbuff[i];
                                }
                                state = 5;
                        }

                        else{
                                printf("pacote corrompido, REJ enviado\n");
                                return 0;
                        }

                }
                i++;
                break;
        }
 }
        return n;
}

/* destuffing
```

```
 *
 */


int destuffing(unsigned char *vet, int size){
	int n = 0;
	for(int i = 0; i < size; i++){
		if(vet[i] == ESCAPE){
			if(vet[i+1] == ESCAPE_FLAG){

				vet[i] = FLAG;
				for(int j = i+1; j<=size ; j++){
					vet[j]=vet[j+1];
				}
				n++;
			}
			if(vet[i+1] == ESCAPE_ESCAPE){
				for(int j = i+1; j<=size ; j++){
					vet[j]=vet[j+1];
				}
				n++;
			}
		}
	}
	return n;}


/* remove extra
 *
 */

int remove_extra(unsigned char *buffer, char ctrl_aux, int size)
{
	int reducedSize = 0;
	int flags;

	unsigned char bcc2vet = buffer[size-2];

	flags = destuffing(buffer, size-2);

	reducedSize = size - flags - 2;

	buffer = (unsigned char *) realloc(buffer, reducedSize);

		unsigned char bcc2cal = BCC2_ver(buffer, reducedSize);

	if( bcc2cal == bcc2vet){
```

```
                if(ctrl_aux == C00)
                {
                        C_FIELD_aux = RR1;
                        trama_ready(generic, C_FIELD_aux);
                        copyToPacketToSend(generic, 5);
                        enviaTrama();
                }
                else
                {                       //C == C10
                        C_FIELD_aux = RR0;
                        trama_ready(generic, C_FIELD_aux);

                        copyToPacketToSend(generic, 5);
                        enviaTrama();
                }
                (stats.I_Frame_Received)++;
                return reducedSize;
        }

        else
        {
                if(ctrl_aux == C00){
                        C_FIELD_aux = REJ1;

                        trama_ready(generic, C_FIELD_aux);
                        copyToPacketToSend(generic, 5);
                        enviaTrama();
                }
                else{
                        C_FIELD_aux = REJ0;
                        trama_ready(generic, C_FIELD_aux);
                        copyToPacketToSend(generic, 5);
                        enviaTrama();
                }

                return -1;
        }
}


/* calculo de BCC2
 *
 */
unsigned char BCC2_ver(unsigned char *arr, int size)
{
    unsigned char result = arr[0];

    for(int i = 0; i < size; i++)
```

```
        result ^= arr[i];

    return result;
}


/* stuffing
 *
 */
int stuffing(char * buf, unsigned char * Newmsg, int size, int sizeBuf)
{
        int aux=4;

        for(int i = 0; i < size; i++)
        {
                if( buf[i] == FLAG )
                {
                        sizeBuf++;
                        Newmsg = (unsigned char *)realloc(Newmsg, sizeBuf);
                        Newmsg[aux] = ESCAPE;
                        Newmsg[aux+1] = ESCAPE_FLAG;

                        aux = aux + 2;
                }

                else if( buf[i] == ESCAPE )
                {
                        sizeBuf++;
                        Newmsg = (unsigned char *)realloc(Newmsg, sizeBuf);
                        Newmsg[aux] = ESCAPE;
                        Newmsg[aux+1] = ESCAPE_ESCAPE;

                        aux = aux + 2;
                }

                else
                {
                        Newmsg[aux] = buf[i];
                        aux++;
                }

        }

        return sizeBuf;
}

/* O REJ que é esperado
 *
```

```
 */
unsigned char correctREJ(unsigned char var)
{
  if( var == C00 )
    return REJ1;

          else
    return REJ0;
}

/* O RR que é esperado
 *
 */
unsigned char correctRR(unsigned char var)
{
  if(var == C00)
    return RR1;

          else
    return RR0;
}


/* Complemento (0/1)
 *
 */
void complementC()
{
  if(C_FIELD_aux == C00)
    C_FIELD_aux = C10;

  else
    C_FIELD_aux = C00;
}


unsigned char getCmdExpectingTwo(unsigned char expecting1, unsigned char expecting2)
{
          unsigned char readChar, res;
          unsigned char matchExpected = 0;
          unsigned char packet_A, packet_C;

          int state = 0;

          while (state != 5)
          {       /* loop for input */

                              res = read(fd, &readChar, 1);
```

```
    if(res < 0)
            {
printf("Read falhou.\n");
exit(-1);
    }

    switch (state)
            {
                    case 0:
                    if (readChar == FLAG)
                            state = 1;
                    break;

            case 1:
                    packet_A = readChar;

                    if(readChar == A)
                            state = 2;

                            else if(readChar != FLAG)
                            state = 0;
                    break;

            case 2:
                    packet_C = readChar;

                            if(readChar == expecting1 || readChar == expecting2)
                            {
                                    matchExpected = readChar;
                            state=3;
                            }

                            else if(readChar != FLAG)
                                    state = 1;

                                    else
                                            state=0;
                            break;

            case 3:
                    if(readChar == (packet_A ^ packet_C)) //expecting bcc1
                            state=4;

                    else
                            state=0;
                    break;
```

```
                        case 4:
                                if(readChar == FLAG)
                                        state = 5;

                                else
                                        state=0;
                                break;
                }
    }

    COMPLETO = TRUE;
                alarm(0);
                tentativas = 0;

        return matchExpected;
}

/* Pacote copiado
 *
 */
void copyToPacketToSend(unsigned char * sourcePacket, int length)
{
        LENGTH = length;
        CopiedPacket = (unsigned char *) realloc(CopiedPacket,sizeof(unsigned char) *
LENGTH);

  for(int i = 0; i < length; i++){
    CopiedPacket[i] = sourcePacket[i];
  }
}
```