Universidade do Porto

FEUP Faculdade de Engenharia

MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES, RAMO: TELECOMUNICAÇÕES, ELETRÓNICA E COMPUTADORES, ESPECIALIZAÇÃO: REDES E SERVIÇOS DE COMUNICAÇÃO

# PROTOCOLO DE LIGAÇÃO DE DADOS

**Unidade Curricular de Redes de Computadores**

**Professor Sérgio Reis Cunha**


Anabela Machado Reigoto, 201405662

André Duarte Correia de Oliveira, 201405639

Baltasar de Vasconcelos Dias Aroso, 201404125

4MIEEC_T_RC


**Dezembro 2016**

# Índice

# 1. Sumário

Este projeto consistiu no desenvolvimento e teste de um protocolo de ligação de dados como primeiro trabalho da unidade curricular Redes de Computadores. A transmissão de dados utiliza como meio uma porta série assíncrona e deve estar preparada para identificar, e lidar com casos de erro, sejam esses casos referentes a dados, sejam devido a falhas na comunicação.

O projeto foi completado com sucesso.

# 2. Introdução

Este relatório tem como objetivo complementar o primeiro trabalho laboratorial da Unidade Curricular Redes de Computadores, elucidando a sua natureza mais teórica, visto que esta poderá não ter sido avaliada na demonstração.

Este trabalho tinha como objetivo a implementação do protocolo de dados especificado em guião, bem como o desenvolvimento de uma aplicação que permitisse transferência de ficheiros.

Para tal, foram criadas um conjunto de funções genéricas: sincronismo de trama, dado que os dados são organizados em tramas; estabelecimento e terminação de ligação; numeração de tramas; confirmação de uma trama sem erros e na sequência correta; controlo de erros e de fluxo.

Este relatório encontra-se dividido em 9 partes:

1. **Arquitetura e Estrutura do Código**, onde são explicados os 3 ficheiros de linguagem C que promovem a independência entre camadas no nosso trabalho;
2. **Casos de Uso Principais**, que se destaca pela referência à função ***main()*** e à interação que o programa tem com o seu utilizador;
3. **Protocolo de Ligação Lógica**, onde são referidas as 4 funções imprescindíveis na camada de ligação de dados;
4. **Protocolo de Aplicação**, onde são referidas as duas funções que determinam o modo de utilização do programa (transmissor ou recetor);
5. **Validação**, que indica quais os testes realizados ao programa para a verificação da sua robustez a erros e da sua conformidade com os objetivos do trabalho;
6. **Elementos de Valorização**, onde são apresentadas características do programa não exigidas no guião, que melhoram o seu desempenho e a sua apresentação;
7. **Caracterização Estatística da Eficiência**, que expõe a eficiência da transmissão de dados que o nosso programa executa face a vários fatores;
8. **Conclusão**, onde é referida uma síntese do trabalho, os objetivos alcançados e o progresso académico que este proporcionou aos alunos;
9. **Anexos**, onde se encontram dados de suporte e o código fonte do nosso programa.

# 3. Arquitetura e Estrutura do Código

O projeto pode ser composto em três partes: camada de aplicação, camada de ligação de dados, ferramentas. Estas serão abaixo expostas.

## 3.1. Camada de Ligação de Dados (*Data Link Layer*)

A camada de ligação de dados, representada pelos ficheiros *datalink.c* e *datalink.h*, tem como principal função garantir a transmissão de dados através da porta série entre os dois computadores. É responsável por estabelecer e terminar a ligação, escrever e ler os dados da porta série, fazer o tratamento de erros, bem como o *stuffing*/*destuffing* de pacotes.

Para além das funções ***llopen()***, ***llwrite()***, ***llread()*** e ***llclose()***, e de modo a facilitar a obtenção de uma trama completa da porta série quando necessário, foi implementada a função ***int getFrame(int port, unsigned char *frame)*** que, recebendo o descritor da porta série, escuta a porta até ter uma trama construída para colocar no buffer *frame* e retorna o seu tamanho. Em caso de erro ou timeout, retorna o seu código de erro correspondente.

## 3.2. Camada de Aplicação (*Application Layer*)

A camada de aplicação, representada pelo ficheiro *application.c*, é responsável pela criação, transferência e respetiva descodificação de pacotes de dados e de controlo, e pela consequente leitura e escrita do ficheiro a transferir.

## 3.3. Ferramentas (*Tools*)

É onde podemos definir previamente os parâmetros de operação do programa, como: o BAUDRATE; o tamanho do buffer da camada de aplicação a enviar para a camada de ligação de dados; o tamanho de trama, consequência direta do tamanho desse buffer; a Frame Error Rate; o tempo de espera em segundos para timeout (WAIT_TIME) e o número de tentativas em caso de timeout (TRIES). É, também, onde encontramos as estruturas e funções auxiliares do programa.

Para facilitar a codificação e descodificação dos TLVs foram criadas as seguintes funções e estruturas de dados:

```
typedef struct {
        unsigned char* file_name;
        int file_length;
        int file_flags;                 /* open() flags */
        mode_t file_mode;               /* open() permissions */
        struct timespec file_time_a;    /* date of last access */
        struct timespec file_time_m;    /* date of last modification */
} details;
```

(responsável por conter os parâmetros do ficheiro)

```
typedef struct {
        unsigned char T;        /* Type */
        unsigned char L;        /* Length */
        unsigned char* V;       /* Value */
} tlv;
```

(um array do tipo *tlv* irá conter todos os parâmetros do ficheiro em *unsigned char*)

**int buildTLVPackage(unsigned char** C, **unsigned char*** package, **tlv*** properties);

(preenche *package* com o campo C, os TLVs de *properties* e retorna o seu tamanho como inteiro)

**void tlvPackage(unsigned char*** packet, **tlv*** properties);

(preenche os TLVs de *properties* com os dados em *packet*)

Para facilitar a criação de Data Packages foram criadas as seguintes funções e estruturas de dados:

```
typedef struct {
        unsigned char N;                /* sequence number */
        unsigned char L1;               /* LSB of file_data size */
        unsigned char L2;               /* MSB of file_data size */
        unsigned char *file_data;
} data;
```

(contém a informação dividida relativa a um pacote de dados)

**int buildDataPackage(unsigned char*** buffer, **unsigned char*** package, **int** pack_size, **int*** seq_num);

(coloca os dados de *buffer*, com tamanho *pack_size*, num pacote *package* com um número de sequência *seq_num* passado por referência, e incrementa-o)

**void dataPackage(unsigned char*** packet, **data*** packet_data);

(reestrutura a informação do pacote *packet* para uma estrutura *packet_data*, do tipo *data*)

**void supervisionFrame(unsigned char*** frame, **unsigned char** A, **unsigned char** C);

(constrói uma trama do tipo S - Supervisão ou U – Não Numeradas e coloca-a no buffer *frame*)

**int constructFrame(unsigned char*** frame, **unsigned char*** buffer, **int** length, **int** sendNumber);
(constrói uma trama do tipo I – Informação a partir do array *buffer* de comprimento *length*, e o número de sequência (0 ou 1) *sendNumber* adequado, e coloca-a no array *frame*, fazendo o byte *stuffing* sempre que necessário)

**int setPort(char*** port, **struct termios*** oldtio);
(configura a porta série e guarda as configurações prévias para que no final do programa as possa restaurar)

**int resetPort(int** fd, **struct termios*** oldtio);
(devolve as configurações originais à porta e fecha-a no programa)

**void randomError (unsigned char*** buffer, **int** buffer_size);
(gera um número aleatório de 0 a 100 -> caso seja inferior a FER – Frame Error Rate definido -> gera um índice aleatório da trama entre 1 e buffer_size – 1, de modo a não afetar os campos F -> coloca o buffer nesse índice a 0x00;
Nota: esta função evita caracteres relativos a byte stuffing pois, por exemplo, caso seja um 0x5D, não será possível detetar erro visto que, no cálculo do Bcc2 não haverá diferença apesar do tamanho da trama ser superior e, havendo independência entre camadas, este erro apenas pode ser detetado na camada de aplicação, resultando numa falha de transferência do ficheiro)

**int getFileLength(int** fd);
(retorna o número de bytes num ficheiro dado pelo descritor *fd*)

**void progressBar(int** done, **int** total);
(imprime o progresso no terminal)

# 4. Casos de Uso Principais

Na camada de aplicação é onde podemos encontrar a função ***main()***, responsável por recolher o modo de operação (RX – Recetor; TX – Transmissor), a porta série a ser usada (/dev/ttySx) e o ficheiro a ser transferido, no caso de se tratar do transmissor, como argumentos de execução do programa. Após esta recolha, e dependendo do respetivo modo de operação, chama uma de duas funções: ***int transmitter(int*** *fd_port,* ***char****** *source_path)* ou ***int receiver(int*** *fd_port)*. Ambas retornam 0 em caso de sucesso e valor negativo com uma mensagem apropriada em caso de erro.

A execução pode ainda ter uma configuração mais aprofundada alterando o BAUDRATE e o tamanho de trama nas Ferramentas (*tools.h*).

# 5. Protocolo de Ligação Lógica

## 5.1. llopen() e llclose()

Estas funções são responsáveis por iniciar e terminar a ligação através da porta série.

Após a camada de aplicação configurar a porta série com sucesso, é chamada a função ***llopen()***, quer a aplicação seja Transmissor, quer seja Recetor.

Se se tratar do Transmissor, a função constrói a trama **SET** (*Set Up*), e envia-a para a porta série. Depois disto, a função fica à espera, durante o tempo definido pelo utilizador (3s por defeito), pela resposta **UA** (*Unnumbered Acknowledgment*). Caso este tempo seja ultrapassado, ocorre um *timeout* e a função tenta enviar a trama de novo. Se o número de *timeouts* máximos permitidos pelo utilizador (3 por defeito) for atingido, a função termina, retornando -1 e informando, assim, a camada de aplicação de que ocorreu um erro na função, não tendo sido estabelecida corretamente a ligação. Este mecanismo de *timeouts* é utilizado ao longo da camada de ligação de dados, sempre que são esperados dados por qualquer uma das partes, de modo a que o protocolo não fique preso em modo de espera.

Se a aplicação se tratar do Recetor, a função fica à espera de receber o comando **SET**, e caso isto aconteça é enviada a trama **UA**, estabelecendo assim corretamente a ligação entre os dois computadores.

A função ***llclose()*** no modo Transmissor começa por enviar o comando **DISC** (*Disconnect)*, ficando de seguida à espera de receber a resposta pretendida do Recetor, também uma trama **DISC**. Se esta for recebida corretamente é enviada a trama **UA** e terminada a função, retornando 0, o que significa que a função ocorreu com sucesso.

A função ***llclose()*** só é chamada pela aplicação Recetor caso tenha sido lida na função ***llread()*** a trama **DISC**. Posto isto, após ter sido chamada, a função envia a trama **DISC** e espera a resposta **UA**. Se receber corretamente esta trama, a função termina com sucesso. Ocorrida com sucesso, na camada de aplicação são repostas as configurações anteriores da porta série.

## 5.2. llread() e llwrite()

Estas funções são necessárias para a escrita e leitura de dados.

A função ***llwrite()*** recebe um pacote, podendo este ser de controlo ou de dados, encapsula-o, construindo uma trama de informação (trama **I**) com a função *constructFrame()*, e envia-o para a porta série. Depois disto, fica à espera de uma resposta por parte do Recetor. Caso receba uma resposta, e esta seja a trama **RR** (*Receiver Ready / positive ACK*), modifica o send number (0 passa a 1, 1 passa a 0) e retorna o número de caracteres escritos na porta. Se esta for uma trama **REJ** (*Reject / negative ACK*) é enviada novamente a trama **I**.

A função ***llread()*** chama a função *getFrame()* de modo a obter uma trama da porta série. Caso a função retorne erro é contado um *timeout* e enviado uma trama **REJ**, de modo a tentar obter a trama o mais rapidamente possível. Se ocorrerem o número máximo de *timeouts* permitidos, a função termina com uma mensagem de erro e retornando um número negativo. Caso o campo de proteção, **BCC(1)** (*Block Check Character*), do cabeçalho esteja errado é de imediato enviada uma trama **REJ**, caso contrário é verificado se a trama recebida é a trama **DISC**. Se for, termina a função e retorna de forma a chamar a função ***llclose()***, senão procede ao *destuffing* tanto do pacote de dados como do **BCC(2)** dos dados, e verifica se este está correto. Caso esteja, envia uma trama **RR** e retorna o tamanho do pacote de dados. Caso se trate de uma trama duplicada, por exemplo, devido a uma trama **RR** perdida, responde igualmente uma trama **RR**, mas retornando da função com um valor por forma a que esta trama seja descartada.

Nota: No caso em que o transmissor era executado antes do recetor, havendo um time-out entre a escrita de 'TX' e 'RX' eram enviadas duas tramas **SET**. Após ler a primeira e o envio da respetiva trama **UA**, o recetor teria então uma trama **SET** na função ***llread()***, que, estando bem construída, passava nas verificações com sucesso. Esta situação foi resolvida, tal como outras situações similares, onde foi necessário uma espécie de "*reset* dos dados lidos" (e.g. no envio de REJs, sempre que há tramas duplicadas e sempre que há time-outs), com recurso à limpeza do buffer usando a função ***memset()*** e a limpeza dos dados pendentes na porta usando a função ***tcflush()***. Tendo-se verificado alguma ineficácia nesta última função, e comprovado após pesquisa, que sem um prévio ***sleep()*** esta função pode não funcionar corretamente em certas *releases* do sistema operativo LINUX.

# 6. Protocolo de Aplicação

## 6.1. transmitter()

Esta é a função responsável pelo procedimento do Transmissor, sendo ela que lê os dados do ficheiro a transmitir e os envia para a camada de ligação de dados. Inicialmente começa por criar um pacote de controlo para indicar o início da transferência do ficheiro, Start Package, que é constituído por um campo de controlo correspondente, seguido de informação sobre o ficheiro codificada no formato TLV (*Type*, *Length*, *Value*). Desta forma, para cada parâmetro do pacote de controlo, é obrigatório indicar primeiramente o tipo de parâmetro (T), seguido do tamanho desse parâmetro em bytes (L) e, finalmente, o valor do parâmetro (V). No protocolo desenvolvido são passados seis parâmetros: tamanho, nome, as flags, as permissões, a última data de acesso e a data de modificação do ficheiro. Este número limita o tamanho mínimo do buffer passado para ***llwrite()***, visto que, dada a natureza dos parâmetros, serão sempre necessários tanto para o Start, como para o End Package 57 bytes mais o número de bytes necessários para o nome do ficheiro, que é o único parâmetro variável em tamanho. Para o "penguin.gif", por exemplo, seriam necessários 68 bytes. Achamos por bem indicar um limite inferior de 128 bytes, por segurança.

Depois de construído, o Start Package é enviado através da função ***llwrite()***.

Concluída esta fase, a função começa por ler do ficheiro a transmitir, onde constrói os pacotes de dados, Data Packages, que depois são enviados utilizando a função ***llwrite()***. Cada Data Package é constituído por um byte C para o campo de controlo, um byte N para o número de sequência (logo, [0, 255]) seguido de dois bytes (L2 - MSB e L1 - LSB) que indicam o número de bytes do campo de dados e, finalmente, o campo de dados com um segmento do ficheiro lido. O número de bytes no campo de dados é limitado pelo valor máximo que podemos representar com 2 bytes, valor esse, k, dado por $256 \times \max\{L2\} + \max\{L1\}$. Ou seja, $k = 256 \times (2^8 - 1) + (2^8 - 1) = 65535$. Este valor limita o tamanho máximo do buffer passado para ***llwrite()***, que é dado por $k + 4 = 65539$, de onde os 4 bytes a mais são devido aos campos C, N, L2 e L1. Uma trama poderá, potencialmente, atingir um tamanho máximo dado por $(65539 + 1) \times 2 + 5 = 131085$, devido a efeitos de byte stuffing e cabeçalho da camada de ligação de dados. À partida, este espaço estará reservado em stack ao executar o programa caso desejemos utilizar o tamanho máximo de trama e não terá de ser alocado, evitando operações de alocação de memória e visando a rapidez do programa.

Quando o ficheiro acaba de ser transferido, a função constrói um pacote de controlo sinalizando o fim da transferência do ficheiro, End Package, da mesma forma que criou o

Start Package, diferenciado estes apenas no campo de controlo C, chamando ***llclose()*** de seguida para terminar a ligação.

## 6.2. receiver()

Seguindo um raciocínio inverso, esta é a função responsável pelo procedimento do Recetor, sendo ela que lê os dados da camada de ligação de dados e os escreve no novo ficheiro.

É responsável, ainda, por verificar a integridade do ficheiro recebido. Isto é, toda a operação de receção do ficheiro tem de corresponder com os parâmetros recebidos no start e end package, simultaneamente. Qualquer falha neste aspeto resulta numa mensagem de erro e o retorno da função com um valor negativo. É analisado, para cada pacote, o campo de controlo C, o número de sequência N, o tamanho dado por L2 e L1 versus o tamanho obtido na leitura do buffer. No final, é analisado o tamanho total recebido contra o tamanho indicado no Start e End Packages e, são, também, comparados os parâmetros iniciais com os finais.

Termina quando a função ***llread()*** lê um **DISC**, indicando através do seu valor de retorno, e sendo chamada de seguida a função ***llclose()*** de forma a terminar a ligação.

# 7. Validação

Para verificar que a aplicação atingiu os objetivos, foram aplicados os seguintes testes:

- Enviar um ficheiro;
- Enviar um ficheiro, interromper a transmissão carregando no botão de interrupção, voltando, depois, a carregar para reabrir a transmissão (com e sem timeouts);
- Enviar um ficheiro e introduzir erros na transmissão com o fio de ligação (com e sem timeouts);
- Enviar um ficheiro, interromper a transmissão e introduzir erros com o fio de ligação (com e sem timeouts).

Para verificar extensivamente a integridade do ficheiro após receção, com auxílio dos comandos ***ls -l*** e ***diff*** no terminal LINUX, comparou-se com uma cópia do original e através de uma comparação das janelas de propriedades do ficheiro lido pelo transmissor contra o ficheiro recebido pelo recetor. Abriu-se, após este processo, de modo a preservar a data de último acesso nas comparações pelo terminal, o ficheiro para uma última verificação.

# 8. Elementos de Valorização

## 8.1. Espera de Dados Bloqueante

Sempre que um ***read()*** na porta série retorne 0, isto é, sempre que não haja leitura de dados apesar de chamar a função ***read()***, recorre-se à função ***select()*** para uma de duas situações: ou ocorre um timeout ao fim de um tempo predefinido, ou a porta indica que já há dados disponíveis para ler. Reduzimos, assim, o custo do programa no CPU. Este processo é intrínseco à função ***getFrame()*** por nós implementada para obtenção de tramas, como é possível ver no seguinte excerto:

```
/*** get byte ***/
if(read(port, &get, 1) == 0) {
	res = select(port + 1, &readfds, NULL, NULL, &tv);
	if(res == -1) {
		perror("select()"); //some error occured in select
		return -2;
	} else if(res == 0) {
		memset(frame, 0, TAM_FRAME);  //cleans buffer if timeout.
		tcflush(port, TCIOFLUSH);
		return ERR_READ_TIMEOUT;
	} else if(FD_ISSET(port, &readfds)) {
		//port has data
		read(port, &get, 1);
	}
}

/*** by this stage function has returned err or get has 1 byte ***/
```

## 8.2. Barra de Progresso

É chamada pelo Transmissor sempre que é enviado um pacote de dados, reaproveitando o número de bytes contados até então. Sendo assim, esta função vai mostrando a percentagem de envio do ficheiro e, consoante essa percentagem, aumenta a barra de progresso que é imprimida, eliminando a barra de progresso anterior sem que se note.

É também chamada pelo Recetor, onde a barra de progresso é aumentada na mesma medida.

## 8.3. Parâmetros do Ficheiro Original Mantidos

A aplicação é capaz de recolher do transmissor, enviar e forçar no recetor os seguintes parâmetros do ficheiro original: nome, tamanho, flags (modo de criação, …), permissões, data de último acesso e data de modificação.

# 9. Caracterização Estatística da Eficiência

## 9.1. Variar FER (Frame Error Rate)

Para tal foi criada a função ***randomError()*** que é chamada sempre que é lida uma trama válida, portanto na função ***llread()***. Esta introduz, tendo em conta uma percentagem FER pré-definida, aleatoriamente, um erro na trama em qualquer campo, exceto nas flags que sinalizam o início e o fim de trama.

**Valores pré-definidos:**

**Baudrate:** 115200; **Tamanho da trama:** 256; **Ficheiro:** penguin.gif

```
void randomError (unsigned char *buffer, int buffer_size) {

  int indice = 0, err = 0;
  struct timeval micros;

  gettimeofday(&micros, NULL);
  srand(micros.tv_usec);
  err = rand() % 101; //n de 0 a 100 que corresponde a percentagem de erro

  if (err < FER) {
    do {
      gettimeofday(&micros, NULL);
      srand(micros.tv_usec);
      indice = rand() % (buffer_size - 3) + 1;
    } while(buffer[indice] == 0x7D || buffer[indice] == 0x7E || buffer[indice] == 0x5D ||
buffer[indice] == 0x5E);

    buffer[indice] = 0x00;
  }

} //randomError()
```

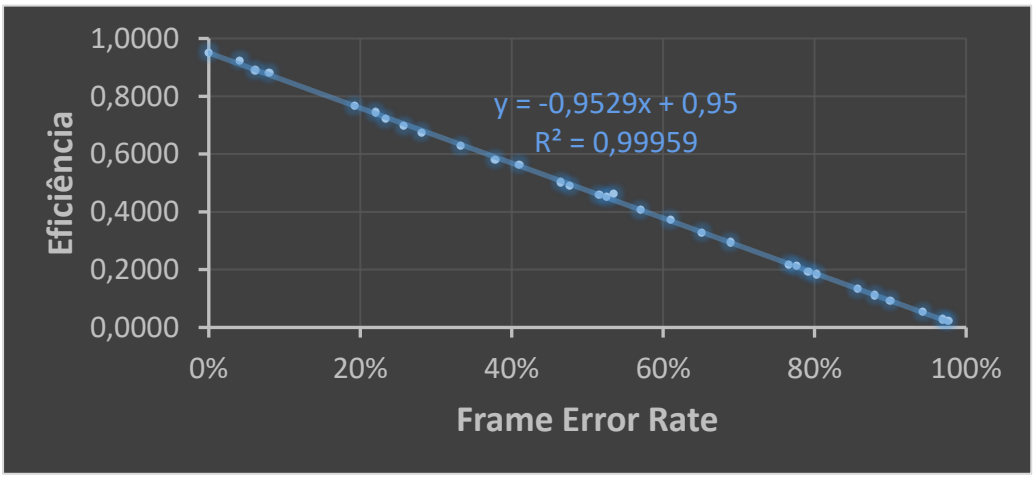


*Figura 1 - Eficiência versus Frame Error Rate*

## 9.2. Variar o tempo de propagação

Para tal foi usada a função ***usleep()*** inserida em ***llread()*** imediatamente após a leitura da trama da porta. Esta função tem como objetivo aumentar o tempo de propagação simulando o aumento do cabo de ligação da porta série aos computadores. Os valores usados foram passados por parâmetro à função em microssegundos.

**Valores pré-definidos:**

**Baudrate:** 115200; **Tamanho da trama:** 1024; **Ficheiro:** penguin.gif

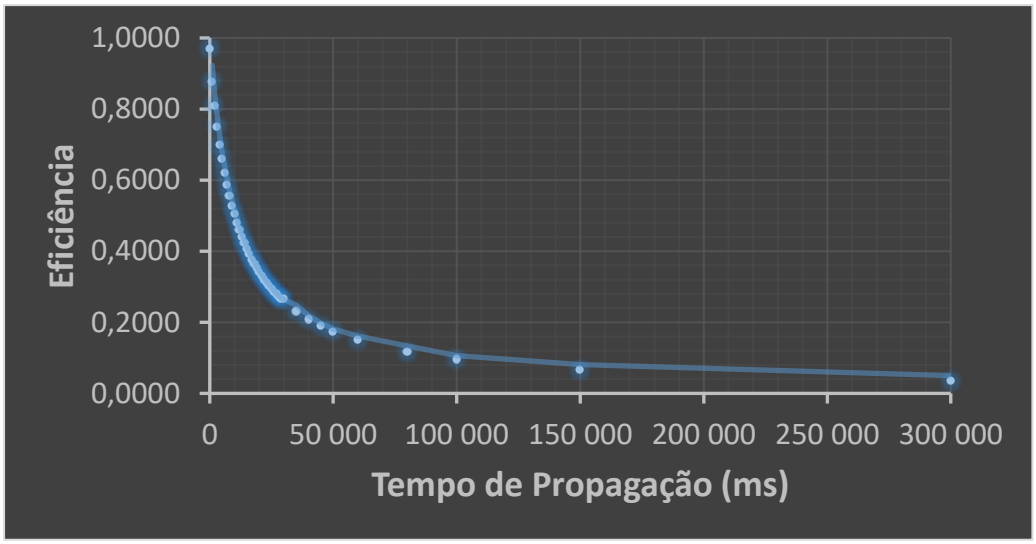


*Figura 2 - Eficiência versus Tempo de Propagação*

## 9.3. Variar C (capacidade de ligação, bit/s)

Variou-se o valor definido para o Baudrate (que equivale à capacidade de ligação da porta série) em *tools.h* e mediu-se o tempo na camada de ligação de dados para o cálculo da eficiência.

**Valores pré-definidos:**

**Tamanho da trama:** 256; **Ficheiro:** penguin.gif

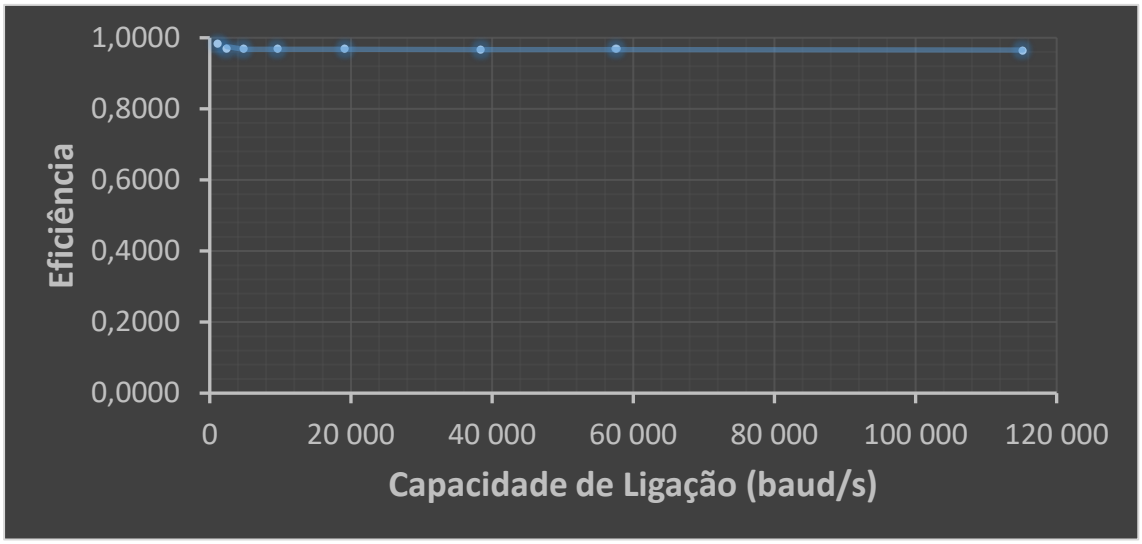


*Figura 3 - Eficiência versus Capacidade de Ligação*

## 9.4. Variar o tamanho da trama

Variou-se o valor definido para o tamanho do buffer e consequente tamanho da trama em *tools.h* e mediu-se o tempo na camada de ligação de dados para o cálculo da eficiência.

**Valores pré-definidos:**

**Baudrate:** 115200; **Ficheiro:** penguin.gif e IFL_Small.jpg

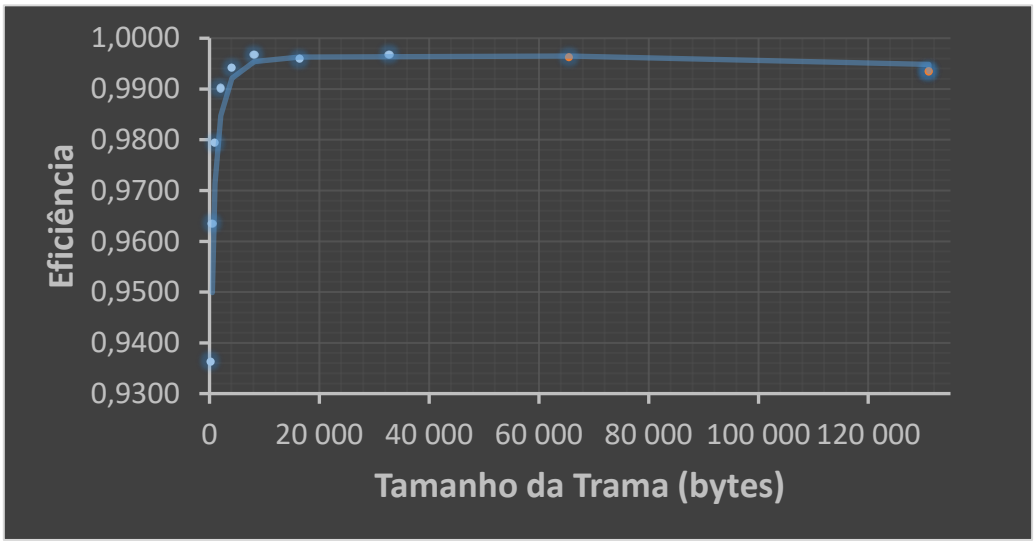


*Figura 4 - Eficiência versus Tamanho da Trama*

# 10. Conclusões

Em suma, os objetivos pretendidos foram atingidos com sucesso. O grau de exigência deste trabalho face aos conhecimentos que tínhamos para a sua execução foi compatível com a nossa vontade de aprender e a proatividade demonstrada nos Elementos de Valorização. A extensa pormenorização do nosso trabalho neste relatório, onde fazemos questão de explicar todos os passos realizados, deve-se às limitações que nos foram aparecendo à medida que íamos testando e limando infindáveis vezes o nosso programa, pelo se julga que se pode afirmar que o protocolo de transmissão de dados implementado ficou bem compreendido. Os 99,7% de eficiência apresentados na Caracterização Estatística da Eficiência corroboram esse progresso académico e contribuem para a nossa satisfação na conclusão deste projeto.

# 10. Anexos

## I. Suporte da Caracterização Estatística de Eficiência

### I.1. Variar FER (Frame Error Rate)

| i | FER Teórico | FER Real | L_TX (bytes) | L_RX (bytes) | L_total (bytes) | Time DataLink (ns) | Time DataLink (s) | S(1) | S(i) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0,00% | 0,0000% | 230 | 11631 | 11861 | 1061413751 | 1,0614 | 0,9512 | 0,9700 |
| 3 | 10,00% | 4,1667% | 240 | 11971 | 12211 | 1094556581 | 1,0946 | 0,9224 | 0,9684 |
| 4 | 10,00% | 6,1224% | 245 | 12420 | 12665 | 1134834048 | 1,1348 | 0,8897 | 0,9688 |
| 2 | 10,00% | 8,0000% | 250 | 12562 | 12812 | 1148626166 | 1,1486 | 0,8790 | 0,9682 |
| 6 | 20,00% | 19,2982% | 285 | 14528 | 14813 | 1318030524 | 1,3180 | 0,7660 | 0,9756 |
| 7 | 20,00% | 22,0339% | 295 | 14865 | 15160 | 1357424921 | 1,3574 | 0,7438 | 0,9695 |
| 5 | 20,00% | 23,3333% | 300 | 15329 | 15629 | 1399166542 | 1,3992 | 0,7216 | 0,9696 |
| 9 | 30,00% | 25,8064% | 310 | 15852 | 16162 | 1448795477 | 1,4488 | 0,6969 | 0,9684 |
| 8 | 30,00% | 28,1250% | 320 | 16370 | 16690 | 1496629006 | 1,4966 | 0,6746 | 0,9680 |
| 10 | 30,00% | 33,3333% | 345 | 17586 | 17931 | 1609575631 | 1,6096 | 0,6273 | 0,9670 |
| 13 | 40,00% | 37,8378% | 370 | 19014 | 19384 | 1738442930 | 1,7384 | 0,5808 | 0,9679 |
| 12 | 40,00% | 41,0256% | 390 | 19628 | 20018 | 1794727251 | 1,7947 | 0,5626 | 0,9682 |
| 11 | 40,00% | 46,5116% | 430 | 21994 | 22424 | 2011354725 | 2,0114 | 0,5020 | 0,9678 |
| 14 | 50,00% | 47,7273% | 440 | 22597 | 23037 | 2064738678 | 2,0647 | 0,4890 | 0,9685 |
| 16 | 50,00% | 51,5789% | 475 | 24165 | 24640 | 2207184316 | 2,2072 | 0,4574 | 0,9691 |
| 15 | 50,00% | 52,5773% | 485 | 24529 | 25014 | 2240724116 | 2,2407 | 0,4506 | 0,9690 |
| 17 | 60,00% | 53,5354% | 495 | 23940 | 24435 | 2192928025 | 2,1929 | 0,4604 | 0,9672 |
| 18 | 60,00% | 57,0093% | 535 | 27164 | 27699 | 2484573284 | 2,4846 | 0,4064 | 0,9677 |
| 19 | 60,00% | 61,0169% | 590 | 29693 | 30283 | 2712208374 | 2,7122 | 0,3723 | 0,9692 |
| 21 | 70,00% | 65,1515% | 660 | 33731 | 34391 | 3079672282 | 3,0797 | 0,3278 | 0,9694 |
| 22 | 70,00% | 68,9189% | 740 | 37515 | 38255 | 3431342562 | 3,4313 | 0,2942 | 0,9678 |
| 20 | 70,00% | 76,6497% | 985 | 51009 | 51994 | 4662649140 | 4,6626 | 0,2165 | 0,9680 |
| 23 | 80,00% | 77,6699% | 1030 | 51906 | 52936 | 4743385523 | 4,7434 | 0,2129 | 0,9687 |
| 24 | 80,00% | 79,1855% | 1105 | 57441 | 58546 | 5243947432 | 5,2439 | 0,1925 | 0,9691 |
| 25 | 80,00% | 80,2575% | 1165 | 60383 | 61548 | 5502511944 | 5,5025 | 0,1835 | 0,9710 |
| 27 | 90,00% | 85,6698% | 1605 | 83139 | 84744 | 7585836369 | 7,5858 | 0,1331 | 0,9697 |
| 26 | 90,00% | 87,9896% | 1915 | 100139 | 102054 | 9135415749 | 9,1354 | 0,1105 | 0,9697 |
| 28 | 90,00% | 90,0000% | 2300 | 119964 | 122264 | 10950911077 | 10,9509 | 0,0922 | 0,9692 |
| 29 | 95,00% | 94,2999% | 4035 | 208761 | 212796 | 19052641187 | 19,0526 | 0,0530 | 0,9695 |
| 30 | 100,00% | 96,9313% | 7495 | 393479 | 400974 | 35923401853 | 35,9234 | 0,0281 | 0,9689 |
| 31 | 100,00% | 97,6410% | 9750 | 507983 | 517733 | 46379998807 | 46,3800 | 0,0218 | 0,9690 |

## I.2. Variar o tempo de propagação

| i | usleep() | L_TX (bytes) | L_RX (bytes) | L_total (bytes) | Time DataLink (ns) | Time DataLink (s) | S |
|---|---|---|---|---|---|---|---|
| **1** | 0 | 230 | 11631 | 11861 | 1061413751 | 1,0614 | 0,9700 |
| **2** | 1000 | 230 | 11631 | 11861 | 1173715453 | 1,1737 | 0,8772 |
| **3** | 2000 | 230 | 11631 | 11861 | 1271200336 | 1,2712 | 0,8099 |
| **4** | 3000 | 230 | 11631 | 11861 | 1372222403 | 1,3722 | 0,7503 |
| **5** | 4000 | 230 | 11631 | 11861 | 1472944361 | 1,4729 | 0,6990 |
| **6** | 5000 | 230 | 11631 | 11861 | 1559713847 | 1,5597 | 0,6601 |
| **7** | 6000 | 230 | 11631 | 11861 | 1657247825 | 1,6572 | 0,6213 |
| **8** | 7000 | 230 | 11631 | 11861 | 1755852257 | 1,7559 | 0,5864 |
| **9** | 8000 | 230 | 11631 | 11861 | 1852656129 | 1,8527 | 0,5557 |
| **10** | 9000 | 230 | 11631 | 11861 | 1954922761 | 1,9549 | 0,5267 |
| **11** | 10000 | 230 | 11631 | 11861 | 2041001443 | 2,0410 | 0,5045 |
| **12** | 11000 | 230 | 11631 | 11861 | 2143100322 | 2,1431 | 0,4804 |
| **13** | 12000 | 230 | 11631 | 11861 | 2235943692 | 2,2359 | 0,4605 |
| **14** | 13000 | 230 | 11631 | 11861 | 2341623263 | 2,3416 | 0,4397 |
| **15** | 14000 | 230 | 11631 | 11861 | 2432687706 | 2,4327 | 0,4232 |
| **16** | 15000 | 230 | 11631 | 11861 | 2530184042 | 2,5302 | 0,4069 |
| **17** | 16000 | 230 | 11631 | 11861 | 2629388344 | 2,6294 | 0,3916 |
| **18** | 17000 | 230 | 11631 | 11861 | 2733921613 | 2,7339 | 0,3766 |
| **19** | 18000 | 230 | 11631 | 11861 | 2824756632 | 2,8248 | 0,3645 |
| **20** | 19000 | 230 | 11631 | 11861 | 2918441518 | 2,9184 | 0,3528 |
| **21** | 20000 | 230 | 11631 | 11861 | 3017813363 | 3,0178 | 0,3412 |
| **22** | 21000 | 230 | 11631 | 11861 | 3112190451 | 3,1122 | 0,3308 |
| **23** | 22000 | 230 | 11631 | 11861 | 3209173737 | 3,2092 | 0,3208 |
| **24** | 23000 | 230 | 11631 | 11861 | 3309261042 | 3,3093 | 0,3111 |
| **25** | 24000 | 230 | 11631 | 11861 | 3406217858 | 3,4062 | 0,3023 |
| **26** | 25000 | 230 | 11631 | 11861 | 3500963147 | 3,5010 | 0,2941 |
| **27** | 26000 | 230 | 11631 | 11861 | 3600134137 | 3,6001 | 0,2860 |
| **28** | 27000 | 230 | 11631 | 11861 | 3694886974 | 3,6949 | 0,2787 |
| **29** | 28000 | 230 | 11631 | 11861 | 3794967368 | 3,7950 | 0,2713 |
| **30** | 29000 | 230 | 11631 | 11861 | 3887875893 | 3,8879 | 0,2648 |
| **31** | 30000 | 230 | 11631 | 11861 | 3888738651 | 3,8887 | 0,2648 |
| **32** | 35000 | 230 | 11631 | 11861 | 4469413530 | 4,4694 | 0,2304 |
| **33** | 40000 | 230 | 11631 | 11861 | 4956754973 | 4,9568 | 0,2077 |
| **34** | 45000 | 230 | 11631 | 11861 | 5450051570 | 5,4501 | 0,1889 |
| **35** | 50000 | 230 | 11631 | 11861 | 5929666475 | 5,9297 | 0,1736 |
| **36** | 60000 | 230 | 11631 | 11861 | 6897078259 | 6,8971 | 0,1493 |
| **37** | 80000 | 230 | 11631 | 11861 | 8837858643 | 8,8379 | 0,1165 |
| **38** | 100000 | 230 | 11631 | 11861 | 10775935543 | 10,7759 | 0,0955 |
| **39** | 150000 | 230 | 11631 | 11861 | 15632400254 | 15,6324 | 0,0659 |
| **40** | 300000 | 230 | 11631 | 11861 | 30185365685 | 30,1854 | 0,0341 |

## I.3. Variar C (capacidade de ligação, bit/s)

| i | BAUDRATE | L_TX (bytes) | L_RX (bytes) | L_total (bytes) | Time DataLink (ns) | Time DataLink (s) | S |
|---|---|---|---|---|---|---|---|
| **1** | 1200 | 230 | 11631 | 11861 | 98667441160 | 98,6674 | 0,9823 |
| **2** | 2400 | 230 | 11631 | 11861 | 50068583158 | 50,0686 | 0,9679 |
| **3** | 4800 | 230 | 11631 | 11861 | 25038142979 | 25,0381 | 0,9678 |
| **4** | 9600 | 230 | 11631 | 11861 | 12520813161 | 12,5208 | 0,9676 |
| **5** | 19200 | 230 | 11631 | 11861 | 6263302060 | 6,2633 | 0,9672 |
| **6** | 38400 | 230 | 11631 | 11861 | 3135519398 | 3,1355 | 0,9660 |
| **7** | 57600 | 230 | 11631 | 11861 | 2087462591 | 2,0875 | 0,9673 |
| **8** | 115200 | 230 | 11631 | 11861 | 1047621420 | 1,0476 | 0,9637 |

## I.4. Variar o tamanho da trama

| i | TAM_BUF | TAM_FRAME | L_TX (bytes) | L_RX (bytes) | L_total (bytes) | Time DataLink (ns) | Time DataLink (s) | S |
|---|---|---|---|---|---|---|---|---|
| **1** | 128 | 263 | 455 | 12081 | 12536 | 1120045846 | 1,1200 | 0,9363 |
| **2** | 256 | 519 | 230 | 11631 | 11861 | 1047902671 | 1,0479 | 0,9635 |
| **3** | 512 | 1031 | 120 | 11412 | 11532 | 1011525695 | 1,0115 | 0,9793 |
| **4** | 1024 | 2055 | 65 | 11301 | 11366 | 990774803 | 0,9908 | 0,9901 |
| **5** | 2048 | 4103 | 40 | 11251 | 11291 | 982453287 | 0,9825 | 0,9941 |
| **6** | 4096 | 8199 | 25 | 11221 | 11246 | 977278674 | 0,9773 | 0,9967 |
| **7** | 8192 | 16391 | 20 | 11211 | 11231 | 977178874 | 0,9772 | 0,9959 |
| **8** | 16384 | 32775 | 15 | 11201 | 11216 | 975451627 | 0,9755 | 0,9968 |
| **9** | 32768 | 65543 | 25 | 85543 | 85568 | 7453877481 | 7,4539 | 0,9962 |
| **10** | 65536 | 131079 | 20 | 85533 | 85553 | 7473350417 | 7,4734 | 0,9935 |
| **11** | 65539 | 131085 | 20 | 85533 | 85553 | 7474475212 | 7,4745 | 0,9933 |

**Nota:** visto o ficheiro "penguin.gif" apenas ter 10968 bytes, para as últimas 3 medições (9, 10 e 11) recorreu-se ao uso de um ficheiro de 84569 bytes, "IFL_Small.jpg".

# II. Código Fonte

## II.1. application.c

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

#include "tools.h"
#include "datalink.h"

int transmitter(int fd_port, char *source_path);
int receiver(int fd_port);

int transmitter(int fd_port, char *source_path) {

    unsigned char buffer[TAM_BUF], package[TAM_BUF];
       int source, res = 0, count_bytes = 0, count_bytes2 = 0, ler = 0;
    int clr = 0, state = 0, done = 0, seq_num = 0, i = 0, sum = 0;
    long sum_long = 0, res_long = 0;
       float divi = 0;
    uint64_t total_nanos_elapsed_inDataLink = 0;
    struct timespec starttime, stoptime;
    struct timespec init_api, finit_api;

       fer_count = 0;
       count_frames = 0;

       tlv properties[NUM_TLV];
       struct stat fileStat;
       details detalhes;

    if(clock_gettime(CLOCK_MONOTONIC, &init_api) < 0) {    //CLOCKING
        perror("clock_gettime()");                         //CLOCKING
        return -1;                                         //CLOCKING
    }                                                      //CLOCKING

       while(!done) {
             switch(state) {
                   case 0:
                         if(stat(source_path, &fileStat) < 0) {
                               perror("stat()");
                               return -1;
                         }

                         if((detalhes.file_name = (unsigned char *) malloc(strlen(source_path)+1)) ==
NULL) {
                               perror("malloc");
                               return -1;
```

```
                        }
                        strcpy((char *) detalhes.file_name, source_path);
                        detalhes.file_flags = O_CREAT | O_APPEND | O_TRUNC | O_WRONLY;
                        detalhes.file_mode = fileStat.st_mode;
                        detalhes.file_time_a = fileStat.st_atim;
                        detalhes.file_time_m = fileStat.st_mtim;

                        source = open(source_path, O_RDONLY);

                        fprintf(stderr, "TESTING llopen()...\n");
                if(clock_gettime(CLOCK_MONOTONIC, &starttime) < 0) {  //CLOCKING
                  perror("clock_gettime()");                          //CLOCKING
                  return -1;                                          //CLOCKING
                }                                                     //CLOCKING
                res = llopen(fd_port, TX);
                if(clock_gettime(CLOCK_MONOTONIC, &stoptime) < 0) {   //CLOCKING
                  perror("clock_gettime()");                          //CLOCKING
                  return -1;                                          //CLOCKING
                }                                                     //CLOCKING

                total_nanos_elapsed_inDataLink += nanos(&stoptime) - nanos(&starttime);  //CLOCKING
                        if(res < 0) {
                                perror("llopen()");
                                return -1;
                        }
                        if(res == 0) {
                                fprintf(stderr, "\n\t llopen() success.\n\n TESTING llwrite()...\n");
                        }

                        if((detalhes.file_length = getFileLength(source)) < 0) {
                                perror("getFileLength()");
                                return -1;
                        }

                    case 1: //start package
                        properties[0].T = 0x00; //Tamanho
                        properties[0].L = sizeof(detalhes.file_length);
                        properties[0].V = (unsigned char *) malloc(properties[0].L);
                        sum = 0;
                        for(i = 0; i < properties[0].L; i++) {
                                res = (detalhes.file_length - sum) % (int)pow(256,3-i);
                                properties[0].V[i] = (detalhes.file_length - res) / pow(256,3-i) -
sum;
                                sum += properties[0].V[i] * pow(256,3-i);
                        }

                        properties[1].T = 0x01; //Nome
                        properties[1].L = strlen((char *) detalhes.file_name) + 1;
                        properties[1].V = (unsigned char *) malloc((int) properties[1].L);
                        memcpy(properties[1].V, detalhes.file_name, (int) properties[1].L);

                        properties[2].T = 0x02; //Flags
                        properties[2].L = sizeof(detalhes.file_flags);
                        properties[2].V = (unsigned char *) malloc((int) properties[2].L);
                        sum = 0;
                        for(i = 0; i < ((int) properties[2].L); i++) {
                                res = (detalhes.file_flags - sum) % (int)pow(256,3-i);
                                properties[2].V[i] = (detalhes.file_flags - res) / pow(256,3-i) -
sum;
                                sum += properties[2].V[i] * pow(256,3-i);
                        }

                properties[3].T = 0x03; //Mode - Permissoes
                        properties[3].L = sizeof(detalhes.file_mode);
                        properties[3].V = (unsigned char *) malloc(properties[3].L);
                        sum = 0;
```

```
                        for(i = 0; i < properties[3].L; i++) {
                                res = (detalhes.file_mode - sum) % (int)pow(256,3-i);
                                properties[3].V[i] = (detalhes.file_mode - res) / pow(256,3-i) -
sum;
                                sum += properties[3].V[i] * pow(256,3-i);
                        }

                properties[4].T = 0x04; //Data de ultimo acesso
                properties[4].L = sizeof(detalhes.file_time_a.tv_sec) +
sizeof(detalhes.file_time_a.tv_nsec);
                properties[4].V = (unsigned char *) malloc(properties[4].L);
                sum_long = 0;
                for(i = 0; i < sizeof(detalhes.file_time_a.tv_sec); i++) {
                  res_long = (detalhes.file_time_a.tv_sec - sum_long) %
(long)pow(256,sizeof(detalhes.file_time_a.tv_sec)-1-i);
                  properties[4].V[i] = (detalhes.file_time_a.tv_sec - res_long) /
pow(256,sizeof(detalhes.file_time_a.tv_sec)-1-i) - sum_long;
                  sum_long += properties[4].V[i] * pow(256,sizeof(detalhes.file_time_a.tv_sec)-1-i);
                }
                sum_long = 0;
                for(i = sizeof(detalhes.file_time_a.tv_sec); i < properties[4].L; i++) {
                  res_long = (detalhes.file_time_a.tv_nsec - sum_long) %
(long)pow(256,properties[4].L-1-i);
                  properties[4].V[i] = (detalhes.file_time_a.tv_nsec - res_long) /
pow(256,properties[4].L-1-i) - sum_long;
                  sum_long += properties[4].V[i] * pow(256,properties[4].L-1-i);
                }

                properties[5].T = 0x05; //Data de modificacao
                properties[5].L = sizeof(detalhes.file_time_m.tv_sec) +
sizeof(detalhes.file_time_m.tv_nsec);
                properties[5].V = (unsigned char *) malloc(properties[5].L);
                sum_long = 0;
                for(i = 0; i < sizeof(detalhes.file_time_m.tv_sec); i++) {
                  res_long = (detalhes.file_time_m.tv_sec - sum_long) %
(long)pow(256,sizeof(detalhes.file_time_m.tv_sec)-1-i);
                  properties[5].V[i] = (detalhes.file_time_m.tv_sec - res_long) /
pow(256,sizeof(detalhes.file_time_m.tv_sec)-1-i) - sum_long;
                  sum_long += properties[5].V[i] * pow(256,sizeof(detalhes.file_time_m.tv_sec)-1-i);
                }
                sum_long = 0;
                for(i = sizeof(detalhes.file_time_m.tv_sec); i < properties[5].L; i++) {
                  res_long = (detalhes.file_time_m.tv_nsec - sum_long) %
(long)pow(256,properties[5].L-1-i);
                  properties[5].V[i] = (detalhes.file_time_m.tv_nsec - res_long) /
pow(256,properties[5].L-1-i) - sum_long;
                  sum_long += properties[5].V[i] * pow(256,properties[5].L-1-i);
                }

                        res = buildTLVPackage(FR_START, package, properties);
                        if(res < 1){
                                perror("buildTLVPackage");
                                return -1;
                        }

                if(clock_gettime(CLOCK_MONOTONIC, &starttime) < 0) {  //CLOCKING
                  perror("clock_gettime()");                          //CLOCKING
                  return -1;                                          //CLOCKING
                }                                                     //CLOCKING
                res = llwrite(fd_port, package, res);
                if(clock_gettime(CLOCK_MONOTONIC, &stoptime) < 0) {   //CLOCKING
                  perror("clock_gettime()");                          //CLOCKING
                  return -1;                                          //CLOCKING
                }                                                     //CLOCKING
                total_nanos_elapsed_inDataLink += nanos(&stoptime) - nanos(&starttime);  //CLOCKING
                        if(res < 0) {
```

```
                        perror("llwrite");
                        return -1;
                    }
                    state = 2;
                    break;

                case 2: //data packages
                    while(count_bytes2 < detalhes.file_length) {

                        if(count_bytes2 + TAM_BUF-4 < detalhes.file_length) {
                            ler = TAM_BUF-4;
                            count_bytes2 += ler;
                        } else {
                            ler = detalhes.file_length - count_bytes2;
                            count_bytes2 += ler;
                        }

                        if((res = read(source, buffer, ler)) < 0) {
                            perror("read()");
                            return -1;
                        }

                        if((res = buildDataPackage(buffer, package, res, &seq_num)) < 0) {
                            perror("buildDataPackage");
                            return -1;
                        }

                    if(clock_gettime(CLOCK_MONOTONIC, &starttime) < 0) {  //CLOCKING
                        perror("clock_gettime()");                        //CLOCKING
                        return -1;                                        //CLOCKING
                    }                                                     //CLOCKING
                    res = llwrite(fd_port, package, ler+4);
                    if(clock_gettime(CLOCK_MONOTONIC, &stoptime) < 0) {   //CLOCKING
                        perror("clock_gettime()");                        //CLOCKING
                        return -1;                                        //CLOCKING
                    }                                                     //CLOCKING
                    total_nanos_elapsed_inDataLink += nanos(&stoptime) - nanos(&starttime);
//CLOCKING
                        if(res < 0) {
                            perror("llwrite()");
                            return -1;
                        } else {
                            count_bytes += res;
                        }

                        for(clr = 0; clr < TAM_BUF; clr++) {
                            buffer[clr] = 0;
                        }
                        progressBar(count_bytes2, detalhes.file_length);
                    }
                    state = 3;
                    break;

                case 3: //end package
                    res = buildTLVPackage(FR_END, package, properties);
                    if(res < 1){
                        perror("buildTLVPackage");
                        return -1;
                    }

                if(clock_gettime(CLOCK_MONOTONIC, &starttime) < 0) {  //CLOCKING
                    perror("clock_gettime()");                        //CLOCKING
                    return -1;                                        //CLOCKING
                }                                                     //CLOCKING
                res = llwrite(fd_port, package, res);
                if(clock_gettime(CLOCK_MONOTONIC, &stoptime) < 0) {   //CLOCKING
```

```
                    perror("clock_gettime()");                          //CLOCKING
                    return -1;                                          //CLOCKING
                }                                                       //CLOCKING
                total_nanos_elapsed_inDataLink += nanos(&stoptime) - nanos(&starttime);  //CLOCKING
                        if(res < 0) {
                                perror("llwrite");
                                return -1;
                        }
                        state = 4;
                        break;

                    case 4:
                        fprintf(stderr, "\n\n\t llwrite() success: bytes as frames: %d bytes; bytes
as data: %d bytes",count_bytes, count_bytes2);

                        //FRAME ERROR RATE
                        divi = (float) fer_count/count_frames;
                        fprintf(stderr, "\nFER: %2.4f%% - fer_count %d - count_frames %d\n\n",
(float) divi*100, fer_count, count_frames);

                        fprintf(stderr, "\n\n TESTING llclose()...\n");

                if(clock_gettime(CLOCK_MONOTONIC, &starttime) < 0) {  //CLOCKING
                    perror("clock_gettime()");                          //CLOCKING
                    return -1;                                          //CLOCKING
                }                                                       //CLOCKING
                res = llclose(fd_port, TX);
                if(clock_gettime(CLOCK_MONOTONIC, &stoptime) < 0) {   //CLOCKING
                    perror("clock_gettime()");                          //CLOCKING
                    return -1;                                          //CLOCKING
                }                                                       //CLOCKING
                total_nanos_elapsed_inDataLink += nanos(&stoptime) - nanos(&starttime);  //CLOCKING
                        if(res < 0) {
                                perror("llclose()");
                                return -1;
                        }
                        if(res == 0) {
                                fprintf(stderr, "\n\t llclose() success.\n\n");
                        }

                fprintf(stderr, "\tTime spent in Data-Link Layer:  %" PRId64 "ns\n",
                                total_nanos_elapsed_inDataLink); //CLOCKING

                        if((close(source) < 0)) {
                                perror("close()");
                                return -1;
                        }
                        done = 1;
                        break;

                    default:
                        break;
            }
        }
    if(clock_gettime(CLOCK_MONOTONIC, &finit_api) < 0) {  //CLOCKING
        perror("clock_gettime()");                          //CLOCKING
        return -1;                                          //CLOCKING
    }                                                       //CLOCKING

    fprintf(stderr, "\tTime spent (including API): \t%" PRId64 "ns\n",
        nanos(&finit_api) - nanos(&init_api)); //CLOCKING
    fprintf(stderr, "\tDifference: \t\t\t%" PRId64 "ns\n\n",
        (nanos(&finit_api) - nanos(&init_api)) - total_nanos_elapsed_inDataLink);  //CLOCKING

    fprintf(stderr, "\n\tcount_bytes_S = %d\n", count_bytes_S);
```

```
      return 0;

} //transmitter()

int receiver(int fd_port) {

    unsigned char buffer[TAM_BUF];
      unsigned char packet[TAM_BUF-1];   //nao contem o campo C
      int output = 0, res = 0, done = 0, state = 0, i = 0, j = 0, change = 0, count_bytes = 0, seq_N
= 255; //valor maximo do N no mod 256
      tlv properties_start[NUM_TLV], properties_end[NUM_TLV];
      data packet_data;
      details start, end;
    struct timespec file_time_obtido[2];
    uint64_t total_nanos_elapsed_inDataLink = 0;
    struct timespec starttime, stoptime;
    struct timespec init_api, finit_api;

    if(clock_gettime(CLOCK_MONOTONIC, &init_api) < 0) {    //CLOCKING
        perror("clock_gettime()");                         //CLOCKING
        return -1;                                         //CLOCKING
    }                                                      //CLOCKING

      fprintf(stderr, "TESTING llopen()...\n");

    if(clock_gettime(CLOCK_MONOTONIC, &starttime) < 0) {   //CLOCKING
        perror("clock_gettime()");                         //CLOCKING
        return -1;                                         //CLOCKING
    }                                                      //CLOCKING
    res = llopen(fd_port, RX);
    if(clock_gettime(CLOCK_MONOTONIC, &stoptime) < 0) {    //CLOCKING
        perror("clock_gettime()");                         //CLOCKING
        return -1;                                         //CLOCKING
    }                                                      //CLOCKING
    total_nanos_elapsed_inDataLink += nanos(&stoptime) - nanos(&starttime);  //CLOCKING
      if(res < 0) {
            perror("llopen()");
            return -1;
      }
      if(res == 0) {
            fprintf(stderr, "\n\t llopen() success.\n\n TESTING llread()...\n");
      }

      while (!done) {
            switch (state) {
                  case 0: //start package
                        if (!change) {
                  if(clock_gettime(CLOCK_MONOTONIC, &starttime) < 0) {  //CLOCKING
                      perror("clock_gettime()");                        //CLOCKING
                      return -1;                                        //CLOCKING
                  }                                                     //CLOCKING
                  res = llread(fd_port, buffer);
                  if(clock_gettime(CLOCK_MONOTONIC, &stoptime) < 0) {   //CLOCKING
                      perror("clock_gettime()");                        //CLOCKING
                      return -1;                                        //CLOCKING
                  }                                                     //CLOCKING
                  total_nanos_elapsed_inDataLink += nanos(&stoptime) - nanos(&starttime);
//CLOCKING
                              if (res == CALL_CLOSE) { //Leu o DISC - Fim da leitura
                                    done = 1;
                                    break;
                              } else if (res < 0) {
                                    perror("llread()");
                                    return -1;
                              }
```

```
                        change = 1;
                }

                if (change) {
                        //Leitura dos pacotes do tipo START, DATA e END
                        for (i = 0; i < res-1; i++) {
                                packet[i] = buffer[i+1];
                        }
                        if (buffer[0] == FR_START){
                                tlvPackage(packet, properties_start);
                                for (i = 0; i < NUM_TLV; i++){
                                        switch (properties_start[i].T) {
                                                case 0x00: //Tamanho
                                                        start.file_length = pow(256,3) *
(int)(properties_start[i].V[0]) +

                pow(256,2) * (int)(properties_start[i].V[1]) +

                pow(256,1) * (int)(properties_start[i].V[2]) +

                pow(256,0) * (int)(properties_start[i].V[3]);
                                                        fprintf(stderr, "\nstart.file_length: %d
bytes\n", start.file_length);
                                                        break;

                                                case 0x01: //Nome
                                                        start.file_name = (unsigned char *)
malloc((int)properties_start[i].L);
                                                        strcpy((char *) start.file_name, (char
*)properties_start[i].V);
                                                        fprintf(stderr, "start.file_name: %s\n",
start.file_name);
                                                        break;

                                                case 0x02: //Flags
                                                        start.file_flags = pow(256,3) *
(int)(properties_start[i].V[0]) +

                        pow(256,2) * (int)(properties_start[i].V[1]) +

                        pow(256,1) * (int)(properties_start[i].V[2]) +

                        pow(256,0) * (int)(properties_start[i].V[3]);
                                                        fprintf(stderr, "start.file_flags: %d\n",
start.file_flags);
                                                        break;

                                        case 0x03: //Mode - Permissoes
                                            start.file_mode = pow(256,3) * (int)(properties_start[i].V[0]) +
                                                              pow(256,2) * (int)(properties_start[i].V[1]) +
                                                              pow(256,1) * (int)(properties_start[i].V[2]) +
                                                              pow(256,0) * (int)(properties_start[i].V[3]);
                                            fprintf(stderr, "start.file_mode: ");
                                            fprintf(stderr, (S_ISDIR(start.file_mode)) ? "d" : "-");
                                            fprintf(stderr, (start.file_mode & S_IRUSR) ? "r" : "-");
                                            fprintf(stderr, (start.file_mode & S_IWUSR) ? "w" : "-");
                                            fprintf(stderr, (start.file_mode & S_IXUSR) ? "x" : "-");
                                            fprintf(stderr, (start.file_mode & S_IRGRP) ? "r" : "-");
                                            fprintf(stderr, (start.file_mode & S_IWGRP) ? "w" : "-");
                                            fprintf(stderr, (start.file_mode & S_IXGRP) ? "x" : "-");
                                            fprintf(stderr, (start.file_mode & S_IROTH) ? "r" : "-");
                                            fprintf(stderr, (start.file_mode & S_IWOTH) ? "w" : "-");
                                            fprintf(stderr, (start.file_mode & S_IXOTH) ? "x" : "-");
                                            fprintf(stderr,"\n\n");
                                            break;
```

```
                                case 0x04: //Time - Last access date
                                    start.file_time_a.tv_sec = 0;
                                    start.file_time_a.tv_nsec = 0;
                                    for(j = 0; j < 8; j++) {
                                        start.file_time_a.tv_sec += pow(256,7-j) *
(long)properties_start[i].V[j];
                                        start.file_time_a.tv_nsec += pow(256,7-j) *
(long)properties_start[i].V[j+8];
                                    }
                                    break;

                                case 0x05: //Time - Last Modification date
                                    start.file_time_m.tv_sec = 0;
                                    start.file_time_m.tv_nsec = 0;
                                    for(j = 0; j < 8; j++) {
                                        start.file_time_m.tv_sec += pow(256,7-j) *
(long)properties_start[i].V[j];
                                        start.file_time_m.tv_nsec += pow(256,7-j) *
(long)properties_start[i].V[j+8];
                                    }
                                    break;

                                                default:
                                                      break;

                                                }
                                        }
                                    } else if (buffer[0] == FR_MID) {
                                        change = 1;
                                        state = 1;
                                        break;
                                    } else {
                                        fprintf(stderr, "\n\nERROR: Wrong C from package\n");
                                        return -1;
                                    }
                                    change = 0;
                                }

                                if((output = open((char *)start.file_name, start.file_flags,
start.file_mode)) < 0) {
                                        perror("open");
                                        return -1;
                                }
                        //0 - Last access date
                        file_time_obtido[0] = start.file_time_a;
                        //1 - Last modification date
                        file_time_obtido[1] = start.file_time_m;
                                break;

                            case 1: //data package
                                  if (!change) {
                                if(clock_gettime(CLOCK_MONOTONIC, &starttime) < 0) {  //CLOCKING
                                    perror("clock_gettime()");                        //CLOCKING
                                    return -1;                                        //CLOCKING
                                }                                                     //CLOCKING
                                res = llread(fd_port, buffer);
                                if(clock_gettime(CLOCK_MONOTONIC, &stoptime) < 0) {   //CLOCKING
                                    perror("clock_gettime()");                        //CLOCKING
                                    return -1;                                        //CLOCKING
                                }                                                     //CLOCKING
                                total_nanos_elapsed_inDataLink += nanos(&stoptime) - nanos(&starttime);
//CLOCKING
                                    if (res == CALL_CLOSE) { //Leu o DISC - Fim da leitura
                                        done = 1;
                                        break;
                                    } else if (res == DUPLICATE) {
```

```
                        break; //se receber frame repetida no llread retorna DUPLICATE; ignora a
leitura do duplicado; volta a ler a proxima
                    } else if (res < 0) {
                                perror("llread()");
                                return -1;
                        }
                        change = 1;
                    }

                    if (change) {
                        //Leitura dos pacotes do tipo START, DATA e END
                        for (i = 0; i < res-1; i++) {
                            packet[i] = buffer[i+1];
                        }
                        if (buffer[0] == FR_MID) {
                    /** Verifica se o tamanho do buffer excede
                    o tamanho suportado pelo pacote
                    (L2 e L1 - 2 bytes - 65535 bytes de dados) **/
                            if ((res-4) != (256*(int)packet[1] + (int)packet[2])) {
                                fprintf(stderr,"ERROR: Described packet size and actual
"
                                    "packet size differ. Writing max data packet size "
                                    "(65535 bytes) from port to output.");
                                return -1;
                            }
                            dataPackage(packet, &packet_data);
                        } else if (buffer[0] == FR_END) {
                            change = 1;
                            state = 2;
                            break;
                        } else {
                            fprintf(stderr, "\n\nERROR: Wrong C from package\n");
                            return -1;
                        }

                        if (((int)packet_data.N - seq_N == 1) || (seq_N - (int)packet_data.N
== 255)) {
                            seq_N = (int)packet_data.N;
                            if (seq_N == 256) {
                                seq_N = 0;
                            }
                        } else {
                            fprintf(stderr, "\n\nERROR: seq_N = %d  |  packet.N = %d\n",
seq_N, (int)packet_data.N);
                    return -1;
                        }

                        if ((res = write(output, packet_data.file_data,
256*(int)packet_data.L2 + (int)packet_data.L1)) < 0) {
                            perror("write()");
                            return -1;
                        }

                        count_bytes += res;

                memset(buffer, 0, TAM_BUF);   //Cleaning the buffer
                        change = 0;

                        progressBar(count_bytes, start.file_length);
                    }
                    break;


                case 2: //end package
                    if (!change) {
                if(clock_gettime(CLOCK_MONOTONIC, &starttime) < 0) {  //CLOCKING
```

```
                        perror("clock_gettime()");                          //CLOCKING
                        return -1;                                          //CLOCKING
                    }                                                       //CLOCKING
                    res = llread(fd_port, buffer);
                    if(clock_gettime(CLOCK_MONOTONIC, &stoptime) < 0) {     //CLOCKING
                        perror("clock_gettime()");                          //CLOCKING
                        return -1;                                          //CLOCKING
                    }                                                       //CLOCKING
                    total_nanos_elapsed_inDataLink += nanos(&stoptime) - nanos(&starttime);
//CLOCKING
                            if (res == CALL_CLOSE) { //Leu o DISC - Fim da leitura
                                done = 1;
                                break;
                            } else if (res < 0) {
                                perror("llread()");
                                return -1;
                            }
                            change = 1;
                        }

                        if (change) {
                            //Leitura dos pacotes do tipo START, DATA e END
                            for (i = 0; i < res-1; i++) {
                                packet[i] = buffer[i+1];
                            }
                            if (buffer[0] == FR_END){
                                tlvPackage(packet, properties_end);
                            }
                            for (i = 0; i < NUM_TLV; i++) {
                                switch (properties_end[i].T) {
                                    case 0x00: //Tamanho
                                        end.file_length = pow(256,3) *
(int)(properties_end[i].V[0]) +

            pow(256,2) * (int)(properties_end[i].V[1]) +

            pow(256,1) * (int)(properties_end[i].V[2]) +

            pow(256,0) * (int)(properties_end[i].V[3]);
                                        fprintf(stderr, "\n\nend.file_length: %d\n",
end.file_length);
                                        break;

                                    case 0x01: //Nome
                                        end.file_name = (unsigned char *)
malloc((int)properties_end[i].L);
                                        strcpy((char *) end.file_name, (char
*)properties_end[i].V);
                                        fprintf(stderr, "end.file_name: %s\n",
end.file_name);
                                        break;

                                    case 0x02: //Flags
                                        end.file_flags = pow(256,3) *
(int)(properties_end[i].V[0]) +

        pow(256,2) * (int)(properties_end[i].V[1]) +

        pow(256,1) * (int)(properties_end[i].V[2]) +

        pow(256,0) * (int)(properties_end[i].V[3]);
                                        fprintf(stderr, "end.file_flags: %d\n",
end.file_flags);
                                        break;

                            case 0x03: //Mode - Permissoes
```

```
                                end.file_mode = pow(256,3) * (int)(properties_end[i].V[0]) +
                                                pow(256,2) * (int)(properties_end[i].V[1]) +
                                                pow(256,1) * (int)(properties_end[i].V[2]) +
                                                pow(256,0) * (int)(properties_end[i].V[3]);
                                fprintf(stderr, "end.file_mode: ");
                                fprintf(stderr, (S_ISDIR(end.file_mode)) ? "d" : "-");
                                fprintf(stderr, (end.file_mode & S_IRUSR) ? "r" : "-");
                                fprintf(stderr, (end.file_mode & S_IWUSR) ? "w" : "-");
                                fprintf(stderr, (end.file_mode & S_IXUSR) ? "x" : "-");
                                fprintf(stderr, (end.file_mode & S_IRGRP) ? "r" : "-");
                                fprintf(stderr, (end.file_mode & S_IWGRP) ? "w" : "-");
                                fprintf(stderr, (end.file_mode & S_IXGRP) ? "x" : "-");
                                fprintf(stderr, (end.file_mode & S_IROTH) ? "r" : "-");
                                fprintf(stderr, (end.file_mode & S_IWOTH) ? "w" : "-");
                                fprintf(stderr, (end.file_mode & S_IXOTH) ? "x" : "-");
                                fprintf(stderr, "\n");
                                break;

                            case 0x04: //Time - Last access date
                                end.file_time_a.tv_sec = 0;
                                end.file_time_a.tv_nsec = 0;
                                for(j = 0; j < 8; j++) {
                                  end.file_time_a.tv_sec += pow(256,7-j) *
(long)properties_end[i].V[j];
                                  end.file_time_a.tv_nsec += pow(256,7-j) *
(long)properties_end[i].V[j+8];
                                }
                                break;

                            case 0x05: //Time - Last Modification date
                                end.file_time_m.tv_sec = 0;
                                end.file_time_m.tv_nsec = 0;
                                for(j = 0; j < 8; j++) {
                                  end.file_time_m.tv_sec += pow(256,7-j) *
(long)properties_end[i].V[j];
                                  end.file_time_m.tv_nsec += pow(256,7-j) *
(long)properties_end[i].V[j+8];
                                }
                                break;

                                        default:
                                break;
                                       }
                                 }
                                 change = 0;
                            }
                            break;

                    default:
                        break;
                     }
            }

            fprintf(stderr, "\n\n\t llread() success: bytes written to output: %d bytes;", count_bytes);

            if (count_bytes != start.file_length && count_bytes != end.file_length) {
                    fprintf(stderr, "\n\nERROR: The size of the file in START_PACKAGE (%d) and in END_PACKAGE
(%d) is different\n\n", start.file_length, end.file_length);
            }

        if((end.file_time_a.tv_sec != file_time_obtido[0].tv_sec)
            || (end.file_time_a.tv_nsec != file_time_obtido[0].tv_nsec)
            || (end.file_time_m.tv_sec != file_time_obtido[1].tv_sec)
            || (end.file_time_m.tv_nsec != file_time_obtido[1].tv_nsec)) {
                fprintf(stderr, "\n\nERROR: Dates in initial package don't match the ones in end
package\n\n");
```

```
    }
    if(futimens(output, file_time_obtido) < 0) {
        perror("futimens");
        return -1;
    }

       fprintf(stderr, "\n\n TESTING llclose()...\n");

    if(clock_gettime(CLOCK_MONOTONIC, &starttime) < 0) {  //CLOCKING
        perror("clock_gettime()");                        //CLOCKING
        return -1;                                        //CLOCKING
    }                                                     //CLOCKING
    res = llclose(fd_port, RX);
    if(clock_gettime(CLOCK_MONOTONIC, &stoptime) < 0) {   //CLOCKING
        perror("clock_gettime()");                        //CLOCKING
        return -1;                                        //CLOCKING
    }                                                     //CLOCKING
    total_nanos_elapsed_inDataLink += nanos(&stoptime) - nanos(&starttime);  //CLOCKING
       if(res < 0) {
              perror("llclose()");
              return -1;
       }
       if(res == 0) {
              fprintf(stderr, "\n\t llclose() success.\n\n");
       }

    fprintf(stderr, "\tTime spent in Data-Link Layer:  %" PRId64 "ns\n",
total_nanos_elapsed_inDataLink); //CLOCKING

       if(close(output) < 0) {
              perror("close()");
              return -1;
       }

    if(clock_gettime(CLOCK_MONOTONIC, &finit_api) < 0) {  //CLOCKING
        perror("clock_gettime()");                        //CLOCKING
        return -1;                                        //CLOCKING
    }                                                     //CLOCKING

    fprintf(stderr, "\tTime spent (including API): \t%" PRId64 "ns\n", nanos(&finit_api) -
nanos(&init_api)); //CLOCKING
    fprintf(stderr, "\tDifference: \t\t\t%" PRId64 "ns\n\n", (nanos(&finit_api) - nanos(&init_api)) -
total_nanos_elapsed_inDataLink);  //CLOCKING

    fprintf(stderr, "\n\tcount_bytes_S = %d\n", count_bytes_S);

       return 0;

} //receiver()

int main(int argc, char** argv) {

       int fd_port;
    struct termios oldtio;
       char buf[100];

    if (argc < 2) {
        printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS1\n");
        exit(-1);
    }

       if((fd_port = setPort(argv[1], &oldtio)) < 0) {
              perror("setPort()");
              exit(-1);
       }
```

```
        fprintf(stderr, "SHALL WE BEGIN?... RX / TX?\n\n");
        if(fgets(buf, sizeof(buf), stdin) == 0){
                perror("fgets");
                exit(-1);
        }
        if((strncmp(buf, "TX", 2) == 0) || (strncmp(buf, "tx", 2) == 0)){
                if(transmitter(fd_port, argv[2]) < 0) {
                        perror("transmitter()");
                        exit(-1);
                }
        } else if((strncmp(buf, "RX", 2) == 0) || (strncmp(buf, "rx", 2) == 0)) {
                if(receiver(fd_port) < 0) {
                        perror("receiver()");
                        exit(-1);
                }
        } else {
                perror("Bad input: Use 'RX' or 'TX' as an argument");
                exit(-1);
        }

        // sleep(1); //para o set de default nao alterar durante a escrita

        //set original port configurations
        if(resetPort(fd_port, &oldtio) < 0) {
                perror("resetPort()");
                exit(-1);
        }

    return 0;

} //main()
```

## II.2. datalink.c

```
/** DATA LINK LAYER **/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <errno.h>
#include <math.h>
#include <sys/time.h>

#include "tools.h"
#include "datalink.h"

/* GLOBAL VAR */
int readNumber = 0;  //Numero da trama a ler
int sendNumber = 0;     //Numero da trama a escrever

//cool function to read stuff from port
int getFrame(int port, unsigned char *frame) {

	int done = 0, res = 0, b = 0;
	unsigned char get;

	fd_set readfds;
	struct timeval tv;

	//make sure fd set is cleared
	FD_ZERO(&readfds);
	//add port to set
	FD_SET(port, &readfds);
	//set timeout value
	tv.tv_sec = WAIT_TIME; //s
	tv.tv_usec = 0;  //us

	memset(frame, 0, TAM_FRAME);  //cleans frame before a read


	while(!done) {
		/*** get byte ***/
		if(read(port, &get, 1) == 0) {
			res = select(port + 1, &readfds, NULL, NULL, &tv);
			if(res == -1) {
				perror("select()"); //some error occured in select
				return -2;
			} else if(res == 0) {
				memset(frame, 0, TAM_FRAME);  //cleans buffer if timeout
				tcflush(port, TCIOFLUSH);              //cleans port if timeout
				return ERR_READ_TIMEOUT;
			} else if(FD_ISSET(port, &readfds)) {
				//port has data
				read(port, &get, 1);
			}
		}

		/*** by this stage function has returned err or get has 1 byte ***/

		if(get == FR_F) {
			if(b == 0) {
				frame[b++] = get;
```

```
                } else {
                        if(frame[b-1] == FR_F) {
                                memset(frame, 0, TAM_FRAME);
                                b = 0;
                                frame[b++] = FR_F;
                        } else {
                                frame[b++] = get;
                                done = 1;
                        }
                }
        } else {
                if(b > 0) {
                        frame[b++] = get;
                }
        }
    }

    return b;  //returns frame length

} //getFrame()

int llopen(int port, int MODE) {

    int state = 0, res = 0, done = 0, bad = 0, got = 0;
    unsigned char SET[5], UA[5];
    unsigned char frame_got[TAM_FRAME];

    //SET
    supervisionFrame(SET, FR_A_TX, FR_C_SET);

    //UA
    supervisionFrame(UA, FR_A_TX, FR_C_UA);

    if(MODE == TX) {
        while (!done) {
            switch (state) {
                case 0: //Envia o SET
                    tcflush(port, TCIOFLUSH);  //clear port
                    if((res = write(port, SET, 5)) < 0) {
                        perror("write()");
                        return -1;
                    }
                    state = 1;
                    break;

                case 1:  //get UA
                    got = getFrame(port, frame_got);

                    /* T_prop */
                    usleep(T_PROP);

                    if(got == ERR_READ_TIMEOUT) {
                        if(bad < TRIES) {
                            fprintf(stderr, "\n\nTime-out: nothing from port after
%d seconds...\n\n\n", WAIT_TIME);
                            bad++;
                            state = 0;
                        } else {
                            fprintf(stderr, "\n\nNothing from RX after %d tries.
Giving up...\n", TRIES);
                            return -1;
                        }
                    } else {
                        if(memcmp(frame_got, UA, 5) == 0) {
```

```
                                        fprintf(stderr, "\nConnection successfully
established.\n");
                                        done = 1;
                                } else {
                                        state = 0;
                                }
                        }
                        break;

                default:
                        break;
                }
        }
        return 0;

    } else if(MODE == RX) {
        while(!done) {
                switch(state) {
                case 0: //get SET
                        got = getFrame(port, frame_got);

                        /* T_prop */
                        usleep(T_PROP);

                        if(got == ERR_READ_TIMEOUT) {
                                if(bad < TRIES) {
                                        fprintf(stderr, "\n\nTime-out: nothing from port after
%d seconds...\n\n\n", WAIT_TIME);
                                        bad++;
                                        state = 0;
                                } else {
                                        fprintf(stderr, "\n\nNothing from TX after %d tries.
Giving up...\n", TRIES);
                                        return -1;
                                }
                        } else {
                                if(memcmp(frame_got, SET, 5) == 0) {
                                        state = 1;
                                }
                        }
                        break;

                case 1:  //send UA
                        // fprintf(stderr, "\nSending UA...\n");
                        tcflush(port, TCIOFLUSH);  //clear port
                        if((res = write(port, UA, 5)) < 0) {    //0 ou 5?
                                perror("write()");
                                return -1;
                        }
                        fprintf(stderr, "\nConnection successfully established.\n");
                        done = 1;
                        break;

                default:
                        break;
                }
        }
        return 0;
    }

    return -1;

} //llopen()

int llread(int port, unsigned char *buffer) {
```

```
	int state = 0, bad = 0, i = 0, j = 0, done = 0, got = 0;
	unsigned char RR[5], REJ[5], RR_LOST[5], DISC[5];
	unsigned char frame_got[TAM_FRAME];
	unsigned char BCC2 = 0x00;

	//DISC
	supervisionFrame(DISC, FR_A_TX, FR_C_DISC);

	while(!done) {
		switch(state) {
			case 0:  //reads frame
				got = getFrame(port, frame_got);

				/* T_prop */
				usleep(T_PROP);

				if(got == ERR_READ_TIMEOUT) {
					if(bad < TRIES) { //daft punk - one more time
						fprintf(stderr, "\n\nTime-out: nothing from port after %d
seconds...\n\n\n", WAIT_TIME);
						bad++;
						state = 6;
					} else {
						fprintf(stderr, "\n\nNothing from TX after %d tries. Giving
up...\n", TRIES);
						return -3;
					}
				} else {
					if (frame_got[2] != FR_C_SET && frame_got[2] != FR_C_UA &&
frame_got[2] != FR_C_DISC) {
						randomError(frame_got, got);
					}
					count_bytes_S+=got; //calc eficiencia
					count_frames++; //calc eficiencia
					state = 1;
				}
				break;

			case 1: //check Bcc1 first
				if(frame_got[3] != (frame_got[1]^frame_got[2])) {  //Bcc1 != A^C
					state = 6; //REJ
					break;
				}
				state = 2;
				break;

			case 2: //check if llread() got a DISC
				if(memcmp(frame_got, DISC, 5) == 0) {
					return CALL_CLOSE;
				} else {
					state = 3;
				}
				break;

			case 3: //destuffing
				for (i = 4; i < got - 1; i++) {
					if((frame_got[i] == ESC) && ((frame_got[i+1] == FR_F_AUX) ||
(frame_got[i+1] == ESC_AUX))) {
						if(frame_got[i+1] == FR_F_AUX) {
							frame_got[i] = FR_F;
						} else if(frame_got[i+1] == ESC_AUX) {
							frame_got[i] = ESC;
						}
						for (j = i + 1; j < got - 1; j++){
```

```
                        frame_got[j] = frame_got[j+1];
                    }
                    got--;
                }
            }
            state = 4;
            break;

        case 4: //check Bcc2
            //Comparacao de BCC2 com o XOR do pacote (D1^D2^...^Dn)
            BCC2 = frame_got[4];
            for (i = 5; i < got - 2; i++) {
                BCC2 = BCC2 ^ frame_got[i];
            }
            if (BCC2 != frame_got[got-2]) {
                state = 6; //REJ
                break;
            }
            state = 5;
            break;

        case 5:  //RR
            if(frame_got[2] == FR_C_SEND0 && readNumber == 0) {
                readNumber = 1;
                supervisionFrame(RR, FR_A_TX, FR_C_RR1);
            } else if (frame_got[2] == FR_C_SEND1 && readNumber == 1) {
                readNumber = 0;
                supervisionFrame(RR, FR_A_TX, FR_C_RR0);
            } else {
                state = 7;  //Entra no RR LOST
                break;
            }

            //Transferencia do pacote de dados da frame para o buffer da API
            for (i = 4, j = 0; i < got - 2; i++, j++) {
                buffer[j] = frame_got[i];
            }
            //SEND RR
            tcflush(port, TCIOFLUSH); //clear port
            if (write(port, RR, 5) < 0) {
                perror("RR write");
                return -3;
            }
            done = 1;
            break;

        case 6: //REJ
            if (got == ERR_READ_TIMEOUT) {
                if(readNumber == 0) {
                    supervisionFrame(REJ, FR_A_TX, FR_C_REJ0);
                } else if (readNumber == 1) {
                    supervisionFrame(REJ, FR_A_TX, FR_C_REJ1);
                }
            } else {
                if(frame_got[2] == FR_C_SEND0 && readNumber == 0) {
                    supervisionFrame(REJ, FR_A_TX, FR_C_REJ0);
                } else if (frame_got[2] == FR_C_SEND1 && readNumber == 1) {
                    supervisionFrame(REJ, FR_A_TX, FR_C_REJ1);
                }
            }

            //send REJ
            tcflush(port, TCIOFLUSH); //clear port
            if (write(port, REJ, 5) < 0) {
                perror("REJ write");
```

```
                        return -3;
                    }
                    state = 0;  //go back and listen the port again
                    break;

                case 7: //DUPLICATE
                    if (frame_got[2] == FR_C_SEND0 && readNumber == 1) {
                        supervisionFrame(RR_LOST, FR_A_TX, FR_C_RR1);
                    } else if (frame_got[2] == FR_C_SEND1 && readNumber == 0) {
                        supervisionFrame(RR_LOST, FR_A_TX, FR_C_RR0);
                    }

                    tcflush(port, TCIOFLUSH); //clear port
                    if (write(port, RR_LOST, 5) < 0) {
                        perror("llwrite(): RR_LOST");
                        return -3;
                    }
                    memset(frame_got, 0, TAM_FRAME);  //cleans buffer if RR is lost.
                    return DUPLICATE;

                default:
                    break;
            }
        }

        return (got - 6);  //frame length without headers

} //llread()

int llwrite(int port, unsigned char* buffer, int length) {

        int state = 0, res = 0, bad = 0, done = 0, sent = 0, frame_len = 0;
        unsigned char RR[5], REJ[5];
        unsigned char frame_sent[TAM_FRAME];
        unsigned char frame_got[TAM_FRAME];

        //construct RR & REJ
        if(sendNumber == 0) {
                supervisionFrame(RR, FR_A_TX, FR_C_RR1);
                supervisionFrame(REJ, FR_A_TX, FR_C_REJ0);
        } else if(sendNumber == 1) {
                supervisionFrame(RR, FR_A_TX, FR_C_RR0);
                supervisionFrame(REJ, FR_A_TX, FR_C_REJ1);
        } else {
                return -1;
        }

        frame_len = constructFrame(frame_sent, buffer, length, sendNumber);

        while(!done) {
                switch(state) {
                        case 0: //writes frame on port
                                sent = write(port, frame_sent, frame_len);
                                state = 1;
                                break;

                        case 1: //listens port for ACK or NACK
                                res = getFrame(port, frame_got);

                                /* T_prop */
                                usleep(T_PROP);

                                if(res == ERR_READ_TIMEOUT) {
                                        if(bad < TRIES) {  //let's give another try
```

```
                                        fprintf(stderr, "\n\nTime-out: nothing from port after %d
seconds...\n\n\n", WAIT_TIME);
                                        bad++;
                                        state = 0;
                                } else {   //you're fresh out of luck, pal
                                        fprintf(stderr, "\n\nNo answer from RX after %d tries. Giving
up...\n", TRIES);
                                        return -1;
                                }
                        } else {
                                count_bytes_S+=res;  //calc eficiencia
                                count_frames++;
                                state = 2;
                        }
                        break;

                case 2: //checks if RR or REJ
                        if(memcmp(RR, frame_got, 5) == 0) { //congrats! it's an RR
                                sendNumber = 1 - sendNumber;
                                done = 1;
                        } else if(memcmp(REJ, frame_got, 5) == 0){   //you still have a shot at
this, bud
                                fer_count++;
                                bad = 0;
                                state = 0;
                        } else {   //what is this, a frame for ants?
                                bad++;
                                state = 0;
                        }
                        break;

                default:
                        break;
            }
        }

        return sent;

} //llwrite()

int llclose(int port, int MODE) {

        int state = 0, res = 0, bad = 0, done = 0, got = 0;
        unsigned char DISC[5], UA[5];
        unsigned char frame_got[TAM_FRAME];

        //DISC
        supervisionFrame(DISC, FR_A_TX, FR_C_DISC);

        //UA
        supervisionFrame(UA, FR_A_TX, FR_C_UA);

        if (MODE == RX) {
                while (!done) {
                        switch (state) {
                        case 0: //Envia o DISC
                                tcflush(port, TCIOFLUSH);  //clear port
                                if((res = write(port, DISC, 5)) < 0) {   //0 ou 5?
                                        perror("write()");
                                        return -1;
                                }
                                state = 1;
                                break;

                        case 1:  //get UA
```

```
                                got = getFrame(port, frame_got);

                                /* T_prop */
                                usleep(T_PROP);

                                if(got == ERR_READ_TIMEOUT) {
                                        if(bad < TRIES) {
                                                fprintf(stderr, "\n\nTime-out: nothing from port after
%d seconds...\n\n\n", WAIT_TIME);
                                                bad++;
                                                state = 0;
                                        } else {
                                                fprintf(stderr, "\n\nNothing from RX after %d tries.
Giving up...\n", TRIES);
                                                return -1;
                                        }
                                } else {
                                        if(memcmp(frame_got, UA, 5) == 0) {
                                                fprintf(stderr, "\nConnection successfully
terminated.\n");
                                                done = 1;
                                        } else {
                                                state = 0;
                                        }
                                }
                                break;

                        default:
                                break;
                }
        }
        return 0;

    } else if (MODE == TX) {
        while (!done) {
                switch (state) {
                        case 0: //Envia o DISC
                                tcflush(port, TCIOFLUSH);  //clear port
                                if((res = write(port, DISC, 5)) < 0) {   //0 ou 5?
                                        perror("write()");
                                        return -1;
                                }
                                state = 1;
                                break;

                        case 1:  //get DISC
                                got = getFrame(port, frame_got);

                                /* T_prop */
                                usleep(T_PROP);

                                if(got == ERR_READ_TIMEOUT) {
                                        if(bad < TRIES) {
                                                fprintf(stderr, "\n\nTime-out: nothing from port after
%d seconds...\n\n\n", WAIT_TIME);
                                                bad++;
                                                state = 0;
                                        } else {
                                                fprintf(stderr, "\n\nNothing from RX after %d tries.
Giving up...\n", TRIES);
                                                return -1;
                                        }
                                } else {
                                        if(memcmp(frame_got, DISC, 5) == 0) {
                                                state = 2;
```

```
                                        } else {
                                                state = 0;
                                        }
                                }
                                break;

                        case 2: //DISC received, send UA
                                tcflush(port, TCIOFLUSH);  //clear port
                                if((res = write(port, UA, 5)) < 0) {  //0 ou 5?
                                        perror("write()");
                                        return -1;
                                }
                                fprintf(stderr, "\nConnection successfully terminated.\n");
                                done = 1;
                                break;

                        default:
                                break;
                        }
                }
                return 0;
        }

        return -1;

} //llclose()
```

## II.3. datalink.h

```
#ifndef _DATA_LINK_
#define _DATA_LINK_

int getFrame(int port, unsigned char *frame);
int llopen(int port, int MODE);
int llwrite(int port, unsigned char* buffer, int length);
int llread(int port, unsigned char* buffer);
int llclose(int port, int MODE);

#endif
```

## II.4. tools.c

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

#include "tools.h"

uint64_t nanos(struct timespec* ts) {

    return ts->tv_sec * (uint64_t)1000000000L + ts->tv_nsec;

}

void randomError (unsigned char *buffer, int buffer_size) {

    int indice = 0, err = 0;
    struct timeval micros;

    gettimeofday(&micros, NULL);
    srand(micros.tv_usec);
    err = rand() % 101; //n de 0 a 100 que corresponde a percentagem de erro

    if (err < FER) {
        do {
            gettimeofday(&micros, NULL);
            srand(micros.tv_usec);
            indice = rand() % (buffer_size - 3) + 1;
        } while(buffer[indice] == 0x7D
            || buffer[indice] == 0x7E
            || buffer[indice] == 0x5D
            || buffer[indice] == 0x5E); //because of byte stuffing and Bcc2

        buffer[indice] = 0x00;
    }

} //randomError()

void supervisionFrame(unsigned char *sframe, unsigned char A, unsigned char C) {

      sframe[0] = FR_F;
      sframe[1] = A;
      sframe[2] = C;
      sframe[3] = sframe[1]^sframe[2];
      sframe[4] = FR_F;

} //supervisionFrame()

int constructFrame(unsigned char* frame, unsigned char* buffer, int length, int sendNumber) {

      int b = 0, i = 0;
      unsigned char Bcc2 = 0x00;
```

```
	frame[b++] = FR_F;                      //Flag inicial
	frame[b++] = FR_A_TX;                  //A

	if(sendNumber == 0) {          //C
		frame[b++] = FR_C_SEND0;
	} else if(sendNumber == 1) {
		frame[b++] = FR_C_SEND1;
	}

	frame[b++] = frame[1]^frame[2]; //Bcc1
	for(i = 0; i < length; i++){    //Byte stuffing dos dados
		Bcc2 = Bcc2 ^ buffer[i];         //Bcc2 feito com XOR dos dados originais
		if(buffer[i] == FR_F) {
			frame[b++] = ESC;
			frame[b++] = FR_F_AUX;
		} else if(buffer[i] == ESC) {
			frame[b++] = ESC;
			frame[b++] = ESC_AUX;
		} else {
			frame[b++] = buffer[i];
		}
	}

	if(Bcc2 == FR_F) {                  //Byte stuffing do Bcc2
		frame[b++] = ESC;
		frame[b++] = FR_F_AUX;
	} else if(Bcc2 == ESC) {
		frame[b++] = ESC;
		frame[b++] = ESC_AUX;
	} else {
		frame[b++] = Bcc2;               //Bcc2
	}

	frame[b] = FR_F;                        //Flag final

	return b+1;
} //constructFrame()S

int buildTLVPackage(unsigned char C, unsigned char* package, tlv* properties) {

	int i = 0, j = 0, b = 0;

	package[b++] = C;
	for(i = 0; i < NUM_TLV; i++) {
		package[b++] = properties[i].T;
		package[b++] = properties[i].L;
		for(j = 0; j < (int) properties[i].L; j++) {
			package[b++] = properties[i].V[j];
		}
	}

	return b;

} //buildTLVPackage()

int buildDataPackage(unsigned char* buffer, unsigned char* package, int pack_size, int* seq_num) {

	int i = 0, tmp = 0;
	package[0] = FR_MID;                        //C
	package[1] = (char) (*seq_num)++;           //N
	if((*seq_num) == 256) {
		*seq_num = 0;
	}

	tmp = pack_size % 256;
	package[2] = (pack_size - tmp) / 256;       //L2 - MSB
```

```
	package[3] = tmp;                              //L1 - LSB

	for(i = 0; i < pack_size; i++) {
		package[i+4] = buffer[i];          //Dados do ficheiro
	}

	return i+4;

} //buildDataPackage()

void tlvPackage(unsigned char *packet, tlv *properties) {

	int i = 0, j = 0, tam_V = 0, k = 0;
	while (k < NUM_TLV) {
		properties[k].T = packet[i];
		properties[k].L = packet[++i];
		tam_V = (int)(properties[k].L);
		properties[k].V = (unsigned char *) malloc(tam_V);
		for (j = 0; j < tam_V; j++){
			properties[k].V[j] = packet[++i];
		}
		i++;
		k++;
	}

} //tlvPackage()

void dataPackage(unsigned char *packet, data *packet_data) {

	int i = 0, j = 0;

	(*packet_data).N = packet[0]; //N
	(*packet_data).L2 = packet[1]; //L2
	(*packet_data).L1 = packet[2]; //L1
	(*packet_data).file_data = (unsigned char*) malloc(256*(int)packet[1] + (int)packet[2]);
	//P1, P2, ..., Pk
	for (j = 0, i = 3; j < 256*(int)(*packet_data).L2 + (int)(*packet_data).L1; j++, i++) {
		(*packet_data).file_data[j] = packet[i];
	}

} //dataPackage()

int setPort(char *port, struct termios *oldtio) {

	if((strcmp("/dev/ttyS0", port) != 0) && (strcmp("/dev/ttyS1", port) != 0)) {
		perror("changePort(): wrong argument for port");
		return -1;
	}

	/*
		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.
	*/
	struct termios newtio;

	int fd;
	if ((fd = open(port, O_RDWR | O_NOCTTY )) < 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 (estava a 0)*/
        newtio.c_cc[VMIN]     = 0;   /* blocking read until 5 chars received (estava a 5)*/

        /*
                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("\nNew termios structure set\n");

        return fd;

} //setPort()

int resetPort(int fd, struct termios *oldtio) {

        if ( tcsetattr(fd, TCSANOW, oldtio) == -1) {  //volta a por a configuracao original
                perror("tcsetattr()");
                return -1;
        }

        close(fd);

        return 0;

} //resetPort()

int getFileLength(int fd) {

        int length = 0;

        if((length = lseek(fd, 0, SEEK_END)) < 0) {
                perror("lseek()");
                return -1;
        }
        if(lseek(fd, 0, SEEK_SET) < 0) {
                perror("lseek()");
                return -1;
        }

        return length;

} //getFileLength()

void progressBar(int done, int total) {

        float fraction = 0.0;
        int j = 0;

        fraction = (float) done/total;
        fprintf(stderr, "\r\33[2KProgress: %2.2f%% - |", fraction*100);
```

```
        for(j = 0; j < fraction * 28; j++) {
                fprintf(stderr, "|");
        }
        for(j = 0; j < 28 - (fraction*28); j++) {
                fprintf(stderr, " ");
        }
        fprintf(stderr, "| - sent/total: %d/%d", done, total);

} //progressBar()
```

## II.5. tools.h

```
#ifndef _TOOLS_
#define _TOOLS_

#include <inttypes.h>

#define BAUDRATE          B115200
#define MODEMDEVICE       "/dev/ttyS0"
#define _POSIX_SOURCE     1 /* POSIX compliant source */
#define FALSE                   0
#define TRUE              1
#define TX                      1
#define RX                      0

#define TAM_BUF           256           //size of buffer (min is 128 for safety because of TLVs; max
is 65539 because of 256*L2 + L1)
#define TAM_FRAME         TAM_BUF*2+7    //potential size of frame
#define DUPLICATE         -2                        //quando recebe uma trama duplicada, descarta
#define CALL_CLOSE        -1                                  //quando recebe DISC em llread
#define FER                     0                             //Frame Error Rate em percentagem
#define NUM_TLV                 6
#define DIRECTORY         "/output/"
#define WAIT_TIME         3
#define T_PROP                  0

#define FR_F              0x7E          //0111 1110
#define FR_F_AUX          0x5E   //0101 1110

#define ESC               0x7D   //0111 1101
#define ESC_AUX           0x5D   //0101 1101

#define FR_A_TX           0x03     //0000 0011
#define FR_A_RX           0x01     //0000 0001

#define FR_C_SET          0x03     //0000 0011
#define FR_C_DISC         0x0B     //0000 1011
#define FR_C_UA           0x07     //0000 0111
#define FR_C_RR0          0x05     //0000 0101
#define FR_C_RR1          0x85     //1000 0101
#define FR_C_REJ0         0x01     //0000 0001
#define FR_C_REJ1         0x81     //1000 0001

#define FR_C_SEND0        0x00     //0000 0000
#define FR_C_SEND1        0x40     //0100 0000
```

```
#define FR_START           0x02         //0000 0010
#define FR_MID             0x01         //0000 0001
#define FR_END             0x03         //0000 0011

#define ERR_READ_TIMEOUT   -1
#define TRIES              3

typedef struct {
       unsigned char* file_name;
       int file_length;
       int file_flags;
       mode_t file_mode;
       struct timespec file_time_a;
       struct timespec file_time_m;
} details;

typedef struct {
       unsigned char T;
       unsigned char L;
       unsigned char *V;
} tlv;

typedef struct {
       unsigned char N;
       unsigned char L1;
       unsigned char L2;
       unsigned char *file_data;
} data;

int fer_count;      //counts REJs
int count_frames;   //counts frames sent
int count_bytes_S;  //count all bytes passed by the port for the efficiency calculation

uint64_t nanos(struct timespec* ts);

void randomError (unsigned char *buffer, int buffer_size);

void supervisionFrame(unsigned char* frame, unsigned char A, unsigned char C);
int constructFrame(unsigned char* frame, unsigned char* buffer, int length, int sendNumber);

int buildTLVPackage(unsigned char C, unsigned char* package, tlv* properties);
int buildDataPackage(unsigned char* buffer, unsigned char* package, int pack_size, int* seq_num);
void tlvPackage(unsigned char *packet, tlv *properties);
void dataPackage(unsigned char *packet, data *packet_data);

int setPort(char *port, struct termios *oldtio);
int changePort(int fd, int readMode);
int resetPort(int fd, struct termios *oldtio);

int getFileLength(int fd);
void progressBar(int done, int total);

#endif
```