RCOM 2020/2021

Relatório - 1º Trabalho laboratorial

Eduardo Brito, up201806271
Pedro Ferreira, up201806506

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Sumário

Este relatório documenta a jornada de trabalho, na unidade curricular de RCOM, em torno do desenvolvimento de um protocolo de ligação de dados e o fornecimento de um serviço de comunicação de dados fiável entre dois sistemas ligados por um meio (canal) de transmissão – neste caso, um cabo série. O ambiente de desenvolvimento estabeleceu-se com o sistema operativo LINUX, a linguagem de programação C e as portas série RS-232 (de comunicação assíncrona), assim como respectivos Drivers e funções da API disponibilizadas pelo sistema operativo.

Como conclusões, retiradas da concretização deste trabalho, são de destacar a consolidação efetiva da aprendizagem no campo dos protocolos de transferência de informação, a implementação bem sucedida do mecanismo por detrás dos mesmos, incidindo em conceitos como *Stop & Wait*, *framing*, *byte stuffing*, *destuffing*, entre outros, e o amadurecimento no que toca à organização do trabalho, à estruturação do código, à validação teórica das formulações e de tudo o resto que envolveu o implementação deste projeto.

Introdução

Os principais objetivos do trabalho prendem-se com a implementação de um protocolo de ligação de dados capaz de fornecer, às camadas lógicas que de si dependem, um serviço de comunicação resiliente a falhas que conceba a ligação necessária à transmissão de dados entre dois sistemas. O protocolo integrou uma aplicação simples de transferência de ficheiros, a qual permitiu testar a sua fiabilidade e eficiência, recorrendo a funções de uma API por este disponibilizada. Genericamente, de protocolos de ligação de dados como o implementado, esperam-se funções de organização e sincronismo de tramas, de estabelecimento e terminação das ligações, de numeração das tramas, confirmação de receção, rejeição de tramas, controlo de erros e de fluxo. Neste trabalho, todas essas noções foram implementadas com sucesso e este relatório explicará o seu processo.

As secções seguintes documentam as várias componentes do projeto, desde a Arquitetura idealizada, as Estruturas de dados, os Casos de uso principais e os dois Protocolos, terminando, por fim, com a Validação e Eficiência do Protocolo de ligação de dados e principais Conclusões do trabalho. As primeiras seções contém informação sobre os blocos funcionais e interfaces criadas, sobre as estruturas de dados concebidas, as funções disponibilizadas e as relações, e organização sequencial, entre os diversos componentes do código. No final, fazer-se-á uma breve descrição dos testes de validação efetuados sobre o protocolo implementado e uma caracterização estatística da sua eficiência.

Arquitetura e Estrutura do Código

O nosso projeto segue uma estrutura de organização baseada em módulos. São claramente distinguidas as interfaces e as respetivas implementações. As primeiras correspondem aos *header files*, que permitem criar uma camada mais elevada de abstração e fornecer aos seus utilizadores apenas os blocos do código necessários, os quais se encontram nos *source files*, num modelo interior que funciona como uma *black box*.

A camada de ligação de dados engloba a maioria dos módulos disponibilizados, sendo eles:

- O módulo *datalink.h,* que contém as funções principais enunciadas - *llopen*, *llwrite*, *llread* e *llclose*.
- Os módulos *sframe.h* e *iframe.h*, que representam as máquinas de estado associadas à avaliação das tramas com formato de supervisão e informação, respetivamente.
- O módulo *utils.h*, que engloba algumas funções auxiliares, constantes de configuração e as principais estruturas de dados usadas pelos restantes módulos.

A camada da aplicação engloba os restantes módulos, que se servem das funções disponibilizadas pelos módulos anteriores, sendo eles:

- O módulo *sender.h*, que, além de disponibilizar a interface do utilizador para envio dos dados, contém também as funções de criação e envio dos pacotes de controlo e dos pacotes de dados.
- O módulo *receiver.h*, que, além de disponibilizar a interface do utilizador para recepção dos dados, contém também as funções de recepção e confirmação dos pacotes de controlo e dos pacotes de dados.

O seguinte diagrama explicita o que foi acima enunciado, relativamente à organização modular do nosso código e aos vários componentes de cada camada de abstração.

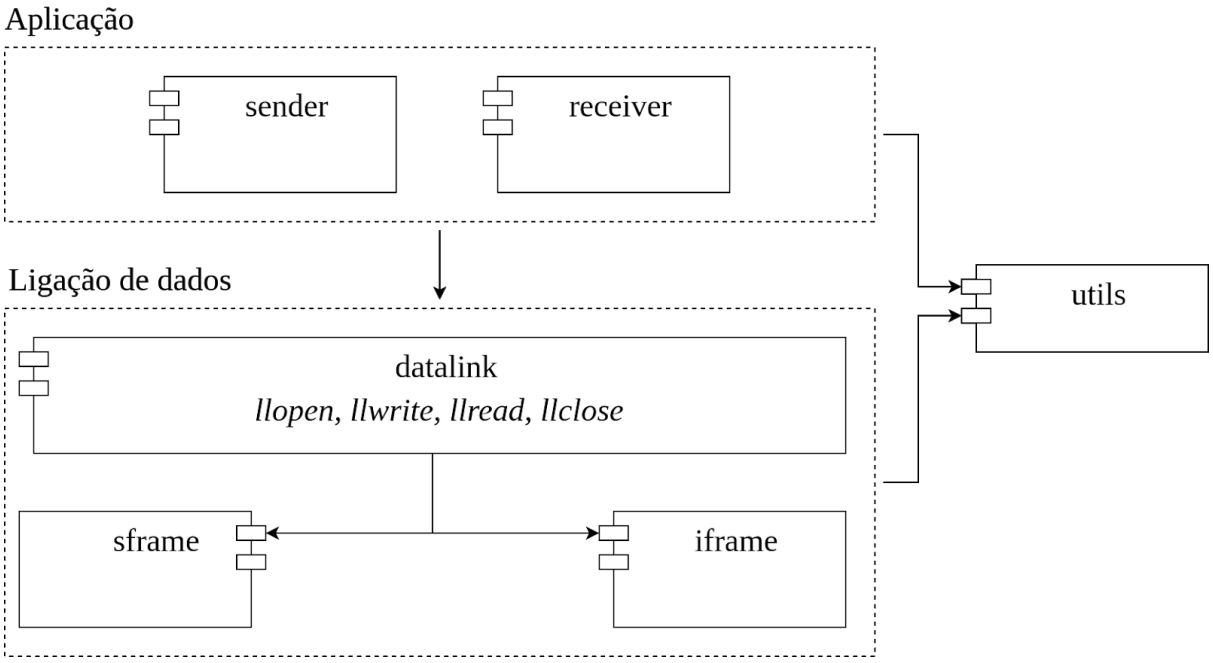


Figura 1. - Visão geral sobre a Arquitetura do projeto.

Estruturas de dados

Neste trabalho, para guardar as informações necessárias de forma simples e compreensível foram utilizadas algumas estruturas de dados que achamos convenientes. Todas elas se encontram no ficheiro utils.h. As estruturas utilizadas foram:

- Uma *enum* **user**, de forma a saber se nos encontramos na presença do emissor, ou do receptor. Esta estrutura é utilizada pelos vários módulos, definindo, durante a execução, o papel do processo na transmissão dos dados, permitindo chamar determinadas funções, ou induzir determinado comportamento, no decorrer do programa.

```
typedef enum // User Type
{
    SENDER,
    RECEIVER,
} user;
```

Figura 2. Estrutura **user**

- A *enum* **fstate**, que engloba os possíveis estados utilizados nas máquinas de estado, definindo o estado atual do **user** aquando da receção e avaliação das tramas, tanto de informação como de supervisão. Estes estados permitem determinar a condição atual dos bytes recebidos e avançar, retroceder, descartar, ou fazer corresponder determinado comportamento, à medida que as tramas vão chegando à entidade que as recebe.

```
typedef enum // Frame States
{
    START,
    FLAG_RCV,
    A_RCV,
    C_RCV,
    RR_DUP,
    BCC1_OK,
    DATA_RCV,
    ESC_RCV,
    BCC2_REJ,
    STOP
} fstate;
```

```
ate iframe_getState(unsigned char input, pframe *t)
{
    switch (t->state)
    {
    case START:
        return iframe_startState(input, t);
    case FLAG_RCV:
        return iframe_flagState(input, t);
    case A_RCV:
        return iframe_aState(input, t);
    case C_RCV:
        return iframe_cState(input, t);
    case BCC1_OK:
        return iframe_dataState(input, t);
    case DATA_RCV:
        return iframe_dataState(input, t);
    case ESC_RCV:
        return iframe_escState(input, t);
    case STOP:
        return STOP;
    default:
        return iframe_startState(input, t);
    }
}
```

```
fstate sframe_getState(unsigned char input, pframe *t)
{
    switch (t->state)
    {
    case START:
        return sframe_startState(input, t);
    case FLAG_RCV:
        return sframe_flagState(input, t);
    case A_RCV:
        return sframe_aState(input, t);
    case C_RCV:
        return sframe_cState(input, t);
    case BCC1_OK:
        return sframe_bccState(input, t);
    case STOP:
        return STOP;
    default:
        return sframe_startState(input, t);
    }
}
```

Figura 3. Estrutura **fstate** e os motores principais das máquinas de estados.

- Uma *struct* **pframe** que armazena, em runtime, informações sobre o **user**, a trama e seus valores expectáveis (flag, a, c, bcc e bcc2), o estado atual da máquina de estados, a informação sobre a porta, o número de tentativas restantes e o número de sequência atual. Armazena, ainda, uma *array de bytes*, com os dados da trama atual, e o tamanho desses dados, e, por fim, uma *struct*, que contém as configurações da porta série, usada nas funções *llopen e llclose*. Esta *struct* **pframe** pode ser usada tanto pelo emissor, como pelo receptor, fazendo, cada processo, a sua própria gestão dos recursos e campos desta estrutura comum.

```
typedef struct // Protocol Frame Struct
{
    user u;
    unsigned char flag1;
    unsigned char a;
    unsigned char expected_a;
    unsigned char c;
    unsigned char expected_c;
    unsigned char bcc;
    unsigned char bcc2;
    unsigned char flag2;
    fstate state;
    int port;
    unsigned int num_retr;
    unsigned int seqnumber;
    unsigned char *buffer;
    unsigned int length;
    struct termios *oldtio;
} pframe;
```

Figura 4. Estrutura **pframe**

Casos de uso

O fornecimento de um serviço de comunicação de dados fiável entre dois sistemas ligados por um meio (canal) de transmissão – neste caso, o cabo série - é garantido através da ponte feita entre os dois módulos criados. Usando-o, o utilizador consegue transferir dados entre as duas máquinas, isto é, o processo emissor, iniciado com o comando `./sender.o -p <port> <filename>`, e o recetor, iniciado com o comando `./receiver.o -p <port>`. O que acontece, de seguida, é o estabelecimento da ligação, a efetiva transmissão e receção dos dados e, por fim, a terminação e conclusão da ligação, tudo isto contemplado num conjunto de funções que são enumeradas nos próximos tópicos.

Protocolo de ligação de dados

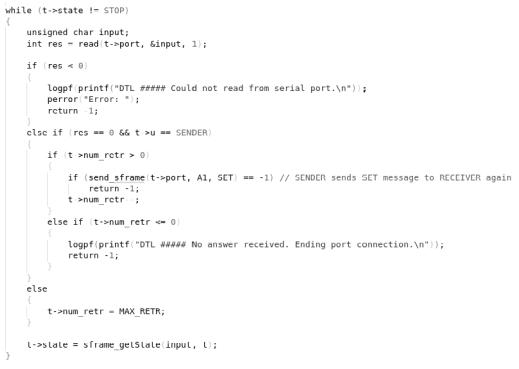
Toda a implementação deste módulo *datalink.h* se baseia num mesmo mecanismo de escrita e leitura da porta série, com a criação das tramas, o seu envio, e a sua posterior recepção e confirmação através de máquinas de estados implementadas nos módulos correspondentes *sframe.h* e *iframe.h*. As tramas são criadas e enviadas, e feito o *byte stuffing* nas tramas de informação, através das funções:

```
int send_sframe(int fd, unsigned char A, unsigned char C)
int send_iframe(int fd, int ns, unsigned char *buffer, int length)
```

- As funções principais *llopen, llread, llwrite e llclose* são responsáveis, respetivamente:
  - pela abertura e configuração da porta de série: `int llopen(int port, user u)`
  - pela receção, leitura e validação das tramas de informação (exclusivamente por parte do recetor): `int llread(int port, unsigned char *buffer)`
  - pelo envio das tramas de informação e validação da receção comunicada pelo recetor (exclusivamente por parte do emissor): `int llwrite(int port, unsigned char *buffer, int length)`
  - pela comunicação final e envio das tramas DISC, correspondendo ao final da transmissão de dados e encerramento do canal: `int llclose(int port)`

As tramas são recebidas byte a byte para serem confirmadas nas máquinas de estado, sendo o processo, em todas as funções *llopen*, *llwrite*, *llread*, *llclose*, bastante semelhante ao loop da Figura 5. A máquina de estados avalia cada input e retorna o novo estado, fazendo também o *destuffing no caso das tramas de informação*, através das funções:

```
fstate sframe_getState(unsigned char input, pframe *t)
fstate iframe_getState(unsigned char input, pframe *t)
```



```
while (t->state != STOP)
{
    unsigned char input;
    int res = read(t->port, &input, 1);

    if (res < 0)
    {
        logpf(printf("DTL ##### Could not read from serial port.\n"));
        perror("Error: ");
        return -1;
    }
    else if (res == 0 && t->u == SENDER)
    {
        if (t->num_retr > 0)
        {
            if (send_sframe(t->port, A1, SET) == -1) // SENDER sends SET message to RECEIVER again
                return -1;
            t->num_retr--;
        }
        else if (t->num_retr <= 0)
        {
            logpf(printf("DTL ##### No answer received. Ending port connection.\n"));
            return -1;
        }
    }
    else
    {
        t->num_retr = MAX_RETR;
    }

    t->state = sframe_getState(input, t);
}
```

Figura 5. loop de recepção das tramas.

Destacamos, também, o mecanismo de timeout que é implementado com recurso às configurações da porta série, nomeadamente, os campos VTIME e VMIN, fazendo todas chamadas read retornar com 0 (VMIN) apenas no caso de terem passado VTIME * 0.1 segundos. Estas configurações encontram-se no módulo *utils.h* e são estabelecidas na função *llopen*.

Protocolo de aplicação

A camada de aplicação é representada por dois ficheiros distintos, correspondendo um ao emissor e outro ao recetor. Falando, primeiramente, do ficheiro correspondente ao emissor (*sender.c*), encontraremos três funções:

- A função *main*, correspondendo à função principal do ficheiro, responsável pela análise dos argumentos passados, aquando da inicialização do processo, bem como da transmissão dos dados para a camada inferior (ligação de dados). Esta transferência é feita com a ajuda das duas seguintes funções.
- A função

  ```
  int send_ctrl_packet(int ctrl_type, int fd, long int filesize, char *filename)
  ```

  que está encarregue de criar e escrever na porta de série um pacote de controlo. Esta função é chamada uma vez, no princípio da emissão, e outra no final, correspondendo, respetivamente, ao pacote *start* e ao pacote *end*.
- A função

  ```
  int send_data_packet(int fd, int nr, unsigned char *data, int length)
  ```

  que, tal como o nome indica, se encarrega de criar e escrever na porta de série os pacote de dados, sendo chamada até não haver mais dados do ficheiro a ser transferido.

O ficheiro correspondente ao recetor (*receiver.c*), apresenta quatro funções que, ao receber os dados provenientes da camada de ligação de dados, os analisam e guardam num novo ficheiro, cópia do original. Deste modo, as funções presentes nele são:

- Tal como no emissor, uma função *main* que, além de validar os argumentos passados, é responsável pela leitura dos pacotes provenientes da porta série. Para obter as informações enviadas pelo emissor, de forma simplista, são utilizadas as três funções auxiliares de obtenção de dados.
- As funções

  ```
  int get_ctrl_packet_filesize(unsigned char *buffer)
  int get_ctrl_packet_filename(unsigned char *buffer)
  int get_data_packet_size(unsigned char * buffer, int nr, int lread)
  ```

  Estas três funções são utilizadas para obtenção da informação correspondente ao tamanho total e ao nome do ficheiro, nos pacotes de controlo, e ao tamanho de cada pacote de dados a receber.

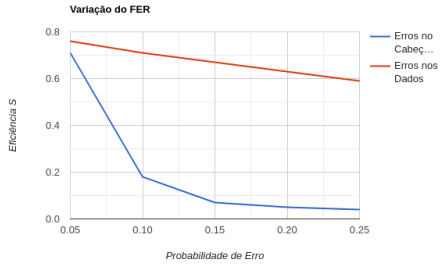
Validação

Foram realizados vários testes de validação dos resultados obtidos. Entre eles, destacam-se os testes ordinários de transferência dos mais variados ficheiros, como imagens, pdfs, vídeos e até ficheiros compostos apenas por bytes do tipo ESC e FLAG, com a finalidade de testar o mecanismo de byte stuffing e destuffing do protocolo. Outros testes mais elaborados foram também conduzidos, nomeadamente, testes para analisar a eficiência, variando diferentes campos do protocolo, e testes para validar o envio de mensagens REJ, ou deteção de duplicados. Para confirmar, detetar e analisar os resultados, um mecanismo de logging foi criado, baseando-se na duplicação de descritores de ficheiros, com o armazenamento das mensagens numa pasta logs, a criar pelo utilizador da nossa aplicação, caso pretenda o relatório das mensagens guardado em ficheiros. Ao longo da transferência, é possível também visualizar a percentagem total de dados enviados e recebidos, até completar o total do ficheiro.

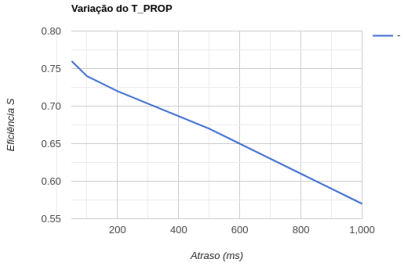
Eficiência do protocolo de ligação de dados

A caracterização estatística da eficiência S (FER, a) foi feita através de medições e testes de variação de determinadas componentes que permitiram validar as fórmulas teóricas estudadas. Seguindo as sugestões do guião deste projeto, apresentam-se, de seguida, os testes de eficiência efetuados. Em anexo, é possível rever as tabelas que originam estes gráficos.

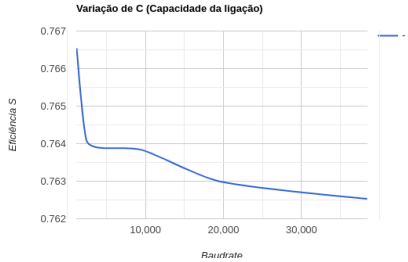
- Variação do FER - através da geração aleatória de erros em tramas de informação, tanto no cabeçalho, como no campo de dados. O seu impacto na eficiência do protocolo é notório, sobretudo pelos erros no cabeçalho, que são ignorados pelo recetor e levam ao *timeout* no emissor. No entanto, a validade é verificada com os erros no Campo de Dados, porque o tempo de execução depende apenas da probabilidade e não do *timeout*. Neste teste, a *baudrate* (38400 bit/s), o tamanho das tramas (256 bytes) e o tamanho do ficheiro (10968 bytes) mantiveram-se constantes, fazendo variar, apenas, a probabilidade de erro.



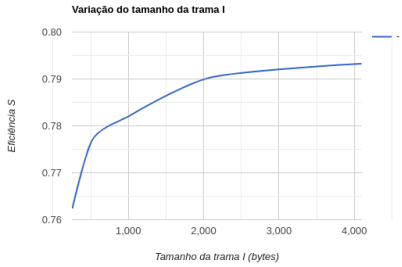
- Variação do T_PROP - através da geração de atraso de propagação simulado. Neste teste, a *baudrate* (38400 bit/s), o tamanho das tramas (256 bytes) e o tamanho do ficheiro (10968 bytes) mantiveram-se constantes, fazendo variar, apenas, o tempo de atraso.



- Variação de C (Capacidade da ligação) - Neste teste, o tamanho das tramas (256 bytes) e o tamanho do ficheiro (10968 bytes) mantiveram-se constantes, fazendo variar, apenas, a *baudrate*.



- Variação do tamanho da trama I - Neste teste, a *baudrate* (38400 bit/s) e o tamanho do ficheiro (10968 bytes) mantiveram-se constantes, fazendo variar, apenas, o tamanho das tramas.



Conclusões

Com a concretização deste trabalho, consolidou-se, efetivamente, a aprendizagem no campo dos protocolos de transferência de informação, a implementação bem sucedida do mecanismo por detrás dos mesmos, incidindo em conceitos como *Stop & Wait*, *framing*, *byte stuffing*, *destuffing*, entre outros, e o amadurecimento no que toca à organização do trabalho, à estruturação do código, à validação teórica das formulações e de tudo o resto que envolveu o implementação deste projeto.

- Variação do FER

Probabilidade de Erro	Eficiência S - Erros no Cabeçalho com *timeout de* 3 s	Eficiência S - Erros no Campo de Dados
0.05	3.2 s => S = 0.71	3.01 s => S = 0.76
0.1	13.0 s => S = 0.18	3.20 s => S = 0.71
0.15	35.1 s => S = 0.07	3.43 s => S = 0.67
0.2	42.2 s => S = 0.05	3.61 s => S = 0.63
0.25	46.1 s => S = 0.04	3.82 s => S = 0.59

- Variação do T_PROP

Atraso	T_PROP	Eficiência S
50 ms	2.962 s	0.76
100 ms	3.049 s	0.74
200 ms	3.133 s	0.72
500 ms	3.411 s	0.67
1000 ms	3.997 s	0.57

- Variação de C (Capacidade da ligação)

Baudrate	Tempo decorrido	Eficiência S
1200	95.4 s	0.76654
2400	47.8 s	0.76412
9600	11.9 s	0.76383
19200	5.98 s	0.76301
38400	2.99 s	0.76253

- Variação do tamanho da trama I

Tamanho da trama I	Tempo decorrido	Eficiência S
256 bytes	3.0 s	0.76242
512 bytes	2.94 s	0.77673
1024 bytes	2.92 s	0.78224
2048 bytes	2.89 s	0.79015
4096 bytes	2.88 s	0.79327