Protocolo de Ligação de Dados

Redes de Computadores

1º trabalho laboratorial - T6G3

04/11/2019

António Dantas up201703878@fe.up.pt
Eduardo Macedo up201703658@fe.up.pt
Vítor Gonçalves up201703917@fe.up.pt

Sumário

No âmbito da unidade curricular Redes de Computadores, foi-nos proposto a realização de um pequeno projeto de transferência de dados. O trabalho consiste no desenvolvimento de uma aplicação que permite a comunicação entre dois computadores, ligados por uma porta de série, enviando ficheiros entre ambos.

No fim da implementação de tudo o que nos foi pedido no enunciado, é possível concluir que os dados são enviados com sucesso e sem perdas de conteúdo, pelo que o desenvolvimento foi efetuado com sucesso.

Introdução

O objetivo deste 1º trabalho laboratorial de Redes de Computadores passa essencialmente pela implementação de um protocolo de ligação de dados que possibilite a comunicação entre dois computadores, transferindo dados entre ambos via porta de série. Para tal, o projeto foi desenvolvido usando computadores com LINUX, a linguagem de programação C e portas de série RS-232.

Com este relatório pretendemos esclarecer a estrutura e organização do nosso código, mostrando as funcionalidades do mesmo. A informação presente no relatório estará distribuída da seguinte forma:

- arquitetura
    - blocos funcionais e interfaces
- estrutura do código
    - uso de API's, principais estruturas de dados utilizadas, principais funções e sua relação com a arquitetura
- casos de uso principais
    - sua identificação e sequências de chamada de funções
- protocolo de ligação lógica
    - identificação dos principais aspectos funcionais, descrição da estratégia de implementação destes aspectos com apresentação de extratos de código
- protocolo de aplicação
    - identificação dos principais aspectos funcionais; descrição da estratégia de implementação destes aspectos com apresentação de extractos de código

- validação
    - descrição dos testes efectuados com apresentação quantificada dos resultados
- eficiência do protocolo de ligação de dados

    - caracterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido.
- conclusão
    - síntese da informação apresentada nas seções anteriores; reflexão sobre os objectivos de aprendizagem alcançados

Arquitetura

Podemos dividir a arquitetura do nosso projeto em dois blocos funcionais distintos, **ligação lógica** e **aplicação**.

O objetivo do bloco de ligação lógica é fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um cabo de série. Tanto o fim e o início da ligação, como o envio de dados são efetuados com o auxílio de tramas (framing). Todas as tramas têm um cabeçalho com um formato comum, podendo ser de três tipos: Informação (I), Supervisão (S) e Não Numeradas (U). Sendo que apenas as de Informação possuem um campo para transporte de dados gerado pela aplicação. Utiliza o mecanismo *Stop-and-Wait* para controlo de erros.

O bloco de aplicação está implementado numa camada de nível mais alto, sendo que é responsável pelo tratamento de informação dos ficheiros a serem enviados. Serve de interface para o utilizador, separando os procedimentos do emissor e do recetor, e notificando-o de possíveis erros que possam interferir no procedimento correto do programa. O lado do emissor deve ler a informação do ficheiro e chamar as funções do bloco de ligação para o envio deste, dividido em tramas. O lado do receptor utiliza o bloco de ligação para ler corretamente as tramas e criar um novo ficheiro com as propriedades especificadas.

Estrutura do Código

Estes dois blocos da arquitetura do projeto estão especificados em dois ficheiros de código diferentes: **protocolo.c** e **app.c**.

protocolo.c → inclui todas as funções necessárias para estabelecer a ligação dos dois sistemas, para a criação de tramas e leitura das mesmas …

- **llopen(int fd, int type)** : função que tem como objetivo estabelecer ligação entre os dois sistemas. Tem como argumentos **fd**, que especifica o descritor de ficheiro utilizado e **type** que especifica que canal está a chamar a função (emissor ou receptor). Aqui, o emissor envia a trama SET, esta é lida pelo receptor (através de uma máquina de estados) que posteriormente envia a trama UA.
- **llwrite(int fd, char * buffer, int length)** : esta função recebe a informação a ser enviada no **buffer** e com o auxílio de **length** (o seu tamanho), cria a trama a ser enviada. Escreve-a para o **fd** e fica à espera de receber a aceitação do emissor (através de uma máquina de estados). **llwrite()** é utilizado para enviar o pacote de

controlo e posteriormente o número de pacotes de dados necessários. Esta função apenas é utilizada pelo emissor.

- **llread(int fd, char * buffer)** : esta função lê do **fd** a informação a ser processada. Através de uma máquina de estados, extrai da trama o bloco de dados e armazena-o no **buffer**. Esta função apenas é utilizada pelo recetor.
- **llclose(int fd, int type)** : função que tem como objetivo terminar a ligação dos dois sistemas. Recebe o **fd** que especifica o descritor de ficheiro utilizado e **type** que especifica que canal está a chamar a função. Aqui, o emissor envia um **DISC,** este é lido pelo receptor (através de uma máquina de estados) que posteriormente envia também um DISC. O emissor lê a trama do recetor e envia por fim um UA.

app.c → está estruturado de maneira a que separe a chamada de funções do emissor e do receptor. Trata da abertura do ficheiro (no lado do emissor) e de criação do novo (no lado do recetor). É responsável também pela configuração correta da porta de série utilizada.

macros.h → inclui todas as macros que são utilizadas nos dois ficheiros acima.

Casos de uso principais

A aplicação criada será corrida em dois computadores, cada um deles com uma funcionalidade diferente. Um será o emissor do ficheiro, e o outro o receptor.

Após a compilação do programa, que pode ser feita usando: ***gcc -o nome_executavel app.c protocolo.c,*** a aplicação pode ser utilizada chamando os seguinte comandos na consola:

- emissor
  - ***./nome_executavel porta sender nome_ficheiro***
- receptor
  - ***./nome_executavel porta receiver***

em qual ***nome_executavel*** é o mesmo invocado na compilação, ***porta*** será ***/dev/ttySx***, em que x pode tomar o valor 0,1 ou 2 e ***nome_ficheiro*** é como o próprio nome indica o nome do ficheiro que se quer enviar.

Após estas chamadas na consola o processo faz a chamada à função ***llopen*** que estabelece comunicação entre os computadores, são chamadas diversas vezes e alternadamente as funções ***llwrite*** e ***llread*** que permitem ao sender enviar dados e ao receiver recebê-los, interpretá-los e escrever para um ficheiro com o mesmo nome do que está a ser enviado. Se não houver mais nada para ler, a comunicação é encerrada usando ***llclose***.

De realçar que se o receptor, após um x período de tempo, ainda não tiver estabelecido a ligação com o emissor este último termina a ligação e fecha. O mesmo não acontece de modo contrário, ou seja, o receptor espera sempre pelo emissor.

Protocolo de Ligação Lógica

A camada de ligação lógica é a camada com nível mais baixo, que faz comunicação entre os dois sistemas.

Ao iniciar o programa, tanto o emissor como o recetor chamam a função **llopen()** (especificada na **Estrutura do Código**). Dependendo do tipo de comunicação, são enviados a trama SET e UA.

```c
// Sends SET
void send_SET_message(int fd){

    char message[5];

    message[0] = M_FLAG;
    message[1] = M_A_REC;
    message[2] = SET;
    message[3] = message[1] ^ message[2];
    message[4] = M_FLAG;

    write(fd, message, 5);
    fflush(NULL);
}
```

```c
// sends UA
void send_UA_message(int fd, int type){

    char UA_message[5];
    char address;

    UA_message[0] = M_FLAG;
    if (type == 1){
        address = M_A_REC;
        UA_message[1] = address;
    }else{
        address = M_A_R;
        UA_message[1] = address;
    }

    UA_message[2] = UA;
    UA_message[3] = address ^ UA;
    UA_message[4] = M_FLAG;

    write(fd, UA_message, 5);
    fflush(NULL);
}
```

Na parte do emissor é criado o packet de controlo e posteriormente dá se *stuffing* do mesmo (usando as funções **assemble_c_frame()** e **stuffer()**). Para escrever o pacote de controlo utiliza-se a função **llwrite()** que fica a percorrer a máquina de estados à espera da resposta correta do recetor. Dependendo do bit alternado (mecanismo *Stop-and-Wait*), esta resposta tanto pode ser RR0, RR1, REJ0 e REJ1.

Na parte do emissor, através da função **llread()**, é lido o pacote de controlo. Posteriormente verificamos se a trama em que foi enviado está corretamente composta através da função **check_bcc2()**, dentro da qual se faz *destuffing* da mesma (**destuffer()**). Se tudo bater certo, envia-se um RR caso contrário um REJ.

```c
// Sends RR
void send_RR_message(int fd){

    unsigned char c_char;
    if (alternate == 0){
        alternate = 1;
        c_char = RR1;
    }else if (alternate == 1){
        alternate = 0;
        c_char = RR0;
    }

    unsigned char message[5];

    message[0] = M_FLAG;
    message[1] = M_A_REC; //Resposta do Recetor -> 0x03
    message[2] = c_char;
    message[3] = M_A_REC ^ c_char;
    message[4] = M_FLAG;

    write(fd, message, 5);
    fflush(NULL);
}
```

```c
// Sends REJ
void send_REJ_message(int fd){

    unsigned char c_char;

    if (alternate == 0)
        c_char = REJ1;
    else if (alternate == 1)
        c_char = REJ0;

    unsigned char message[5];

    message[0] = M_FLAG;
    message[1] = M_A_REC;
    message[2] = c_char;
    message[3] = M_A_REC ^ c_char;
    message[4] = M_FLAG;

    write(fd, message, 5);
    fflush(NULL);
}
```

Depois disto, o emissor começa a enviar a informação propriamente dita, em ciclo, utilizando a função **build_data_packet()** e **llwrite()** novamente. Por outro lado, o recetor vai recebendo estes pacotes de informação, em ciclo também, e extrai-lhes os dados pretendidos com **rem_data_packet()**. No fim de cada pacote envia uma mensagem RR ou REJ dependendo da estruturação da trama.

Por fim, chama-se a função **llclose()** que para além de utilizar a função **send_UA_message()** (se o tipo de comunicação foi emissor), usa também a função **send_DISC_message()** em ambos recetor e emissor.

```
// sends DISC depending on weather its a receiver or sender
void send_DISC_message(int fd, int type){

    char disc_message[5];
    char address;

    disc_message[0] = M_FLAG;
    if (type == 1){                 // sender
        address = M_A_REC;
        disc_message[1] = address;
    }else{                          // receiver
        address = M_A_R;
        disc_message[1] = address;
    }

    disc_message[2] = DISC;
    disc_message[3] = address ^ DISC;
    disc_message[4] = M_FLAG;

    write(fd, disc_message, 5);

    fflush(NULL);
}
```

Protocolo de Aplicação

O protocolo de aplicação é implementado na camada da aplicação, que é a de mais alto nível, e é onde o programa inicia e onde se encontram as chamadas às funções do protocolo de ligação lógica.

Esta camada é responsável pelo envio e receção de pacotes de controlo, pacotes de dados, e pelo envio e recepção do ficheiro a ser enviado.

A nossa aplicação está dividida essencialmente em 2 grandes blocos, um que controla o sender e outro o receiver. Para isso, é necessário aquando do começo do programa obter a informação passada ao programa pelo utilizador. Através da variável ***comunication_type*** é possível fazer a distinção entre as funcionalidades que cada computador terá.

```
70  //processar argumentos da consola
71      if (((strcmp("/dev/ttyS0", argv[1])!=0) && (strcmp("/dev/ttyS1", argv[1])!=0) && (strcmp("/dev/ttyS2", argv[1])!=0))){
72
73          printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS1\n");
74              exit(1);
75          }
76
77
78      if(strcmp(argv[2], "sender") == 0){
79          comunication_type = SENDER;
80          if(argc != 4){     //sendo SENDER terá obrigatoriamente 4 argumentos
81              printf("Must be specified a file path to transmit\n");
82              exit(1);
83          }
84
85      }
86      else if(strcmp(argv[2], "receiver") == 0){
87          comunication_type = RECEIVER;
88      }
89      else{
90          printf("argv[2] must be sender or receiver");
91          exit(1);
92      }
```

Conforme explicado em seções anteriores(Arquitetura e Estrutura do código) a aplicação tem funcionalidades diferentes para o caso do computador ser ***sender*** ou ***receiver.***

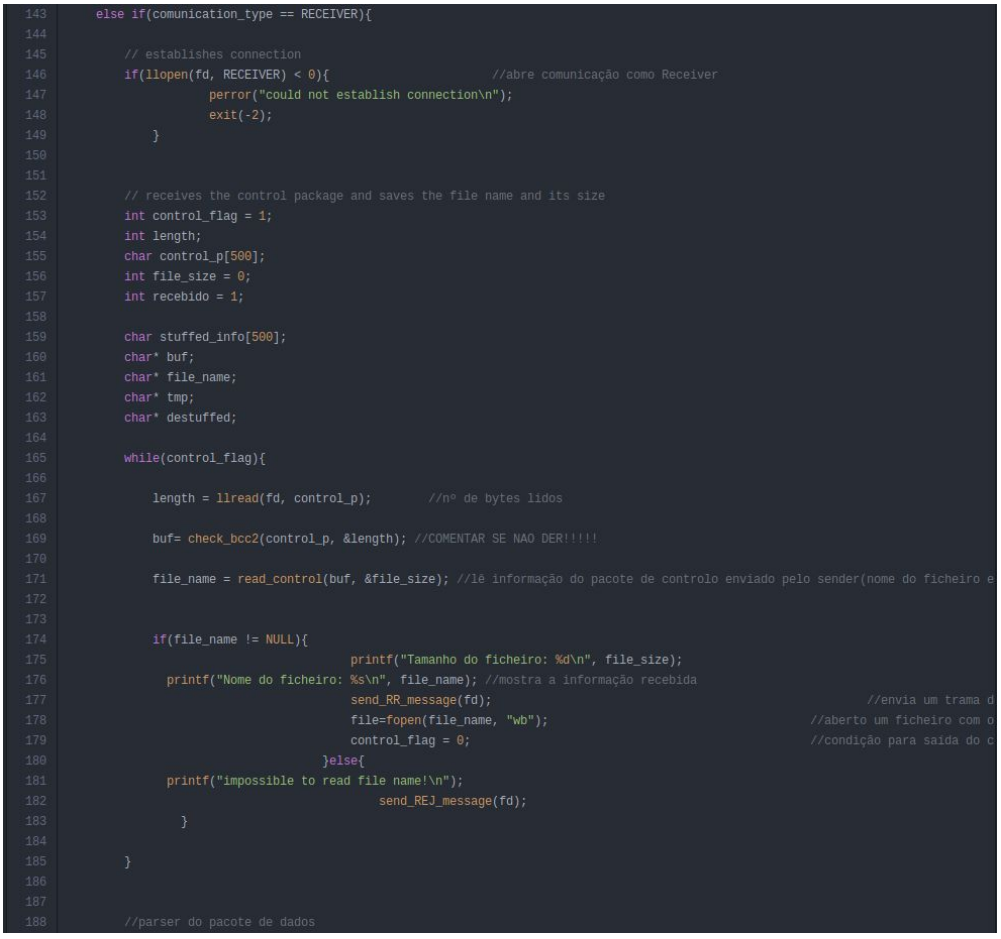
Para a aplicação do 1º caso, **sender**, a aplicação faz o seguinte:

1. Abre o ficheiro com o diretório passado na consola em modo de leitura
2. Estabelece comunicação usando **llopen()**
3. Criação e envio do pacote de controlo inicial
4. Divisão e envio dos dados, através de pacotes de dados, com tamanho definido em **MAX_SIZE**
5. Criação e envio do pacote de controlo final

```c
/* llopen para estabelecer ligações */
/* depende do tipo, SENDER or receiver*/

    if (comunication_type == SENDER){

        if ((file = fopen(name, "rb")) == NULL)   //verificar se o ficheiro a transmitir foi aberto com sucesso
            printf("failed to open file\n");
        if (llopen(fd, comunication_type) == -1)    //estabelece comunicação e verifica-la, entre as duas máquinas
            printf("failed communication with receiver\n");

        fflush(NULL);


        int counter = 0;
        int size;
        char tmp[MAX_SIZE];

        char *control_frame;
        char *data_packet;
        char *data_packet_stuffed;
        char *control_packet_stuffed;


        control_frame = assemble_c_frame(1, file, name, &size); //constrói pacote de controlo START
        control_packet_stuffed = stuffer(control_frame, &size);      //dá stuffer do controlo

        if(llwrite(fd, control_packet_stuffed, size) == -1) //envia controlo --> size = size do control_packet_stuffed
            exit(1);
        else printf("Sending . . .\n");

        while((size = fread(tmp, sizeof(char), MAX_SIZE, file)) > 0){   //Lê fragmento do ficheiro e sai do ciclo quando não houver

            data_packet = build_data_packet(counter++, &size, tmp);             //constrói pacote de dados
            data_packet_stuffed = stuffer(data_packet, &size);          //BCC2 e stuffer

            if (llwrite(fd, data_packet_stuffed, size) == -1)
                exit(1);
                else printf("Pacote de dados nº %d enviado com sucesso:\n",counter);
        }

        control_frame = assemble_c_frame(0, file, name, &size);       //termina a transferência de dados e é enviado um pacote de c
        control_packet_stuffed = stuffer(control_frame, &size);

        if (llwrite(fd, control_packet_stuffed, size) == -1)    //envia pacote de fim
            printf("failed to send END frame\n");

    }
```

No caso do **receiver,** a aplicação faz o descrito a seguir.

1. Estabelece comunicação usando **llopen()**
2. Através de **llread()** lê a informação passado pelo sender e verifica se o bcc2 corresponde através da função **check_bcc2()**
3. Obter os dados do ficheiro enviado usando read_control
4. Se o passo 3 foi obtido com sucesso abrir ficheiro em modo escrita com o nome obtido na função read_control, e de imediato envia RR, caso falhe envia REJ.
5. Neste passo, o receptor vai começar a ler fragmentos do ficheiro enviado e vai escrever para o ficheiro aberto em 4.
6. Para o ponto 5. ser efetuado com sucesso é preciso fazer destuffing e verificar o bcc2 e ainda remover os campos que não fazem parte dos dados presentes nos pacotes de dados enviados com rem_data_packet().
   Para completar os passos anteriores, se estes foram executados com sucesso enviar RR caso contrário REJ.



```
143     else if(comunication_type == RECEIVER){
144
145         // establishes connection
146         if(llopen(fd, RECEIVER) < 0){                    //abre comunicação como Receiver
147                     perror("could not establish connection\n");
148                     exit(-2);
149             }
150
151
152         // receives the control package and saves the file name and its size
153         int control_flag = 1;
154         int length;
155         char control_p[500];
156         int file_size = 0;
157         int recebido = 1;
158
159         char stuffed_info[500];
160         char* buf;
161         char* file_name;
162         char* tmp;
163         char* destuffed;
164
165         while(control_flag){
166
167             length = llread(fd, control_p);        //nº de bytes lidos
168
169             buf= check_bcc2(control_p, &length); //COMENTAR SE NAO DER!!!!!
170
171             file_name = read_control(buf, &file_size); //lê informação do pacote de controlo enviado pelo sender(nome do ficheiro e
172
173
174             if(file_name != NULL){
175                                         printf("Tamanho do ficheiro: %d\n", file_size);
176               printf("Nome do ficheiro: %s\n", file_name); //mostra a informação recebida
177                                         send_RR_message(fd);                                        //envia um trama d
178                                         file=fopen(file_name, "wb");                              //aberto um ficheiro com o
179                                         control_flag = 0;                                         //condição para saida do c
180                                     }else{
181               printf("impossible to read file name!\n");
182                                             send_REJ_message(fd);
183                 }
184
185         }
186
187
188         //parser do pacote de dados
```

No final da aplicação chama-se a função **llclose()** que dá por terminada a ligação entre os dois computadores e fecha a porta de série.

Validação

Para testar o projeto desenvolvida, o grupo realizou os seguintes teste com a aplicação:

- Envio de vários ficheiros com tamanhos variados.
- Interrupção do sinal múltiplas vezes durante o envio da mensagem.
- Envio do ficheiro com ruído induzido.
- Envio do ficheiro com uma probabilidade de erro associada ao Bcc2.
- Envio do ficheiro com a introdução de um *delay*.
- Envio do ficheiro com níveis variados de *baudrate* e tamanho de pacotes.

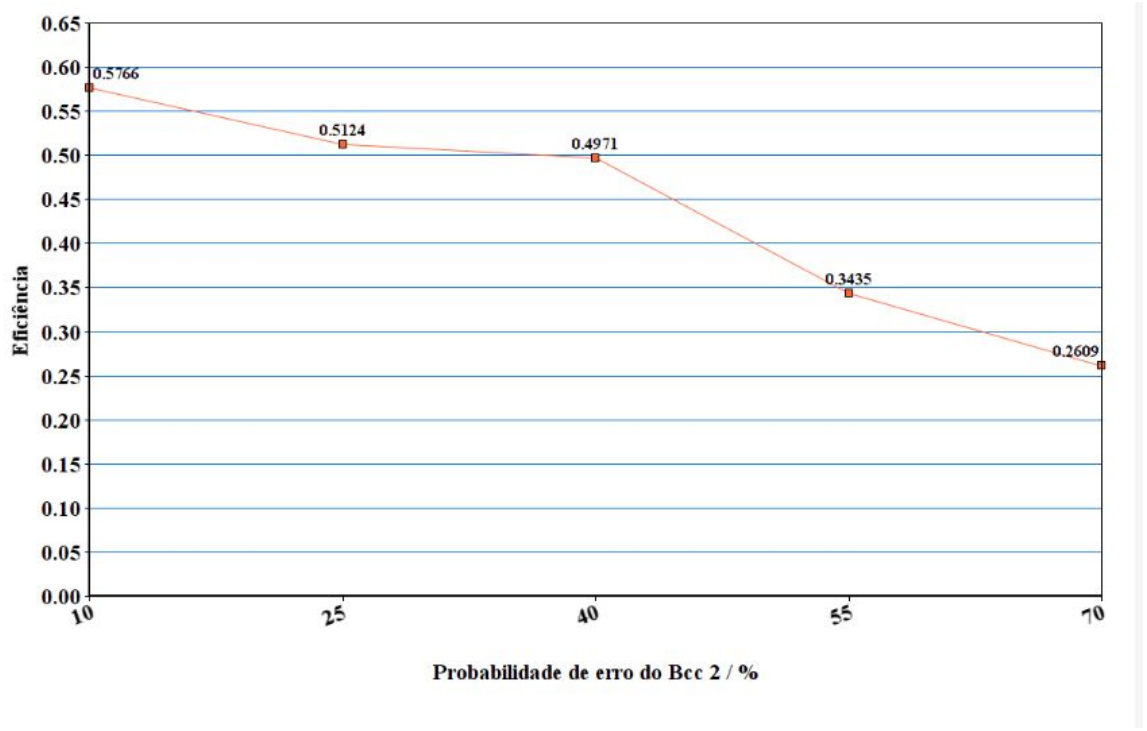
Todos os testes foram concluídos com sucesso há exceção do teste com ruído induzido onde obtivemos uma taxa de sucesso de ~75%.

Eficiência do protocolo de ligação de dados

Com o objetivo de testar a eficiência da aplicação desenvolvida foram realizados 4 testes em que foi calculada a eficiência variando FER, C, Tempo de propagação e tamanho dos tramas.

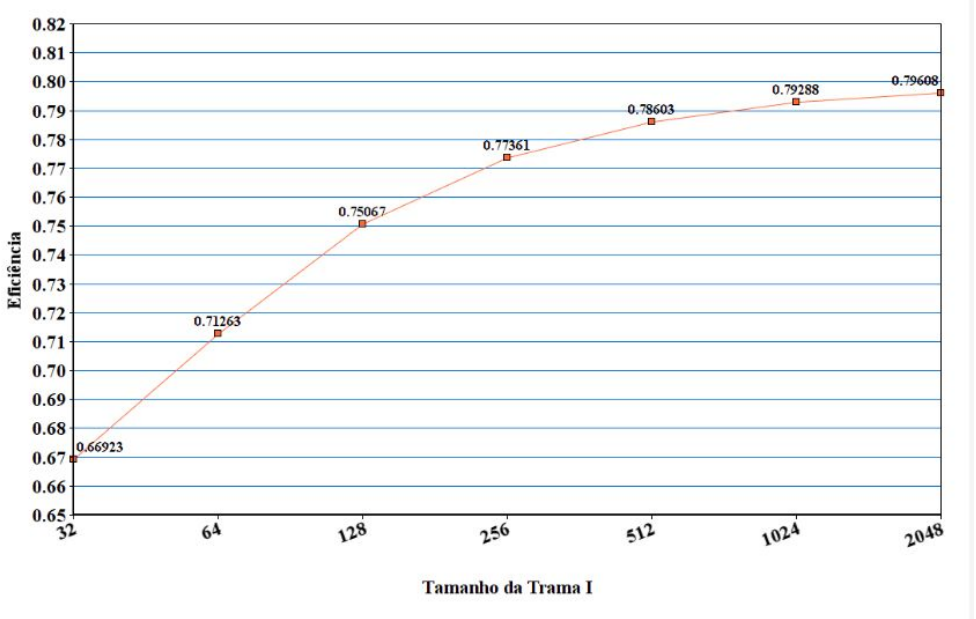
Variação do FER:

Com este gráfico podemos ver que há uma grande queda na eficiência à medida que aumentamos a probabilidade de erro apenas no Bcc2. Isto deve-se ao facto de quando ocorre erro aqui é necessário reenviar a trama.



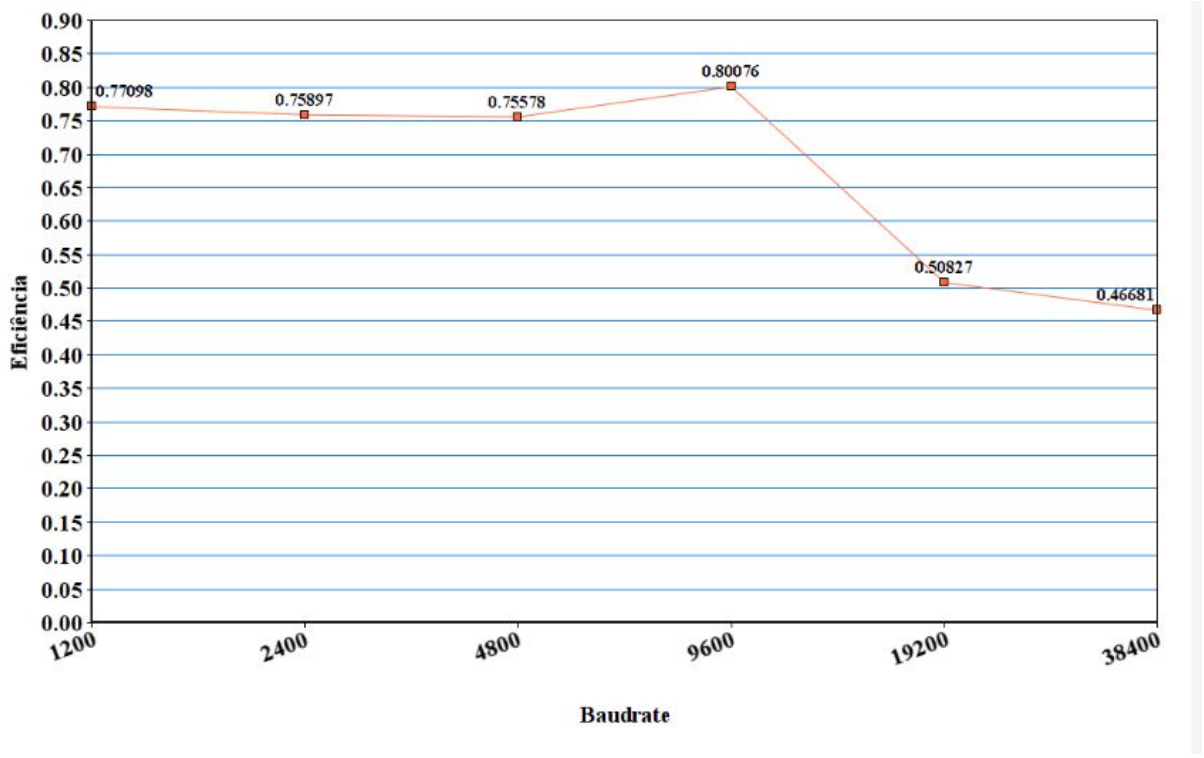
Variação do tamanho das tramas I:

Com o seguinte gráfico podemos visualizar que quanto maior o trama de informação mais eficiente se torna a aplicação devido ao facto de ao ser enviada mais informação de uma vez, são necessárias que cada vez menos tramas sejam enviadas e daí o programa seja mais eficiente.



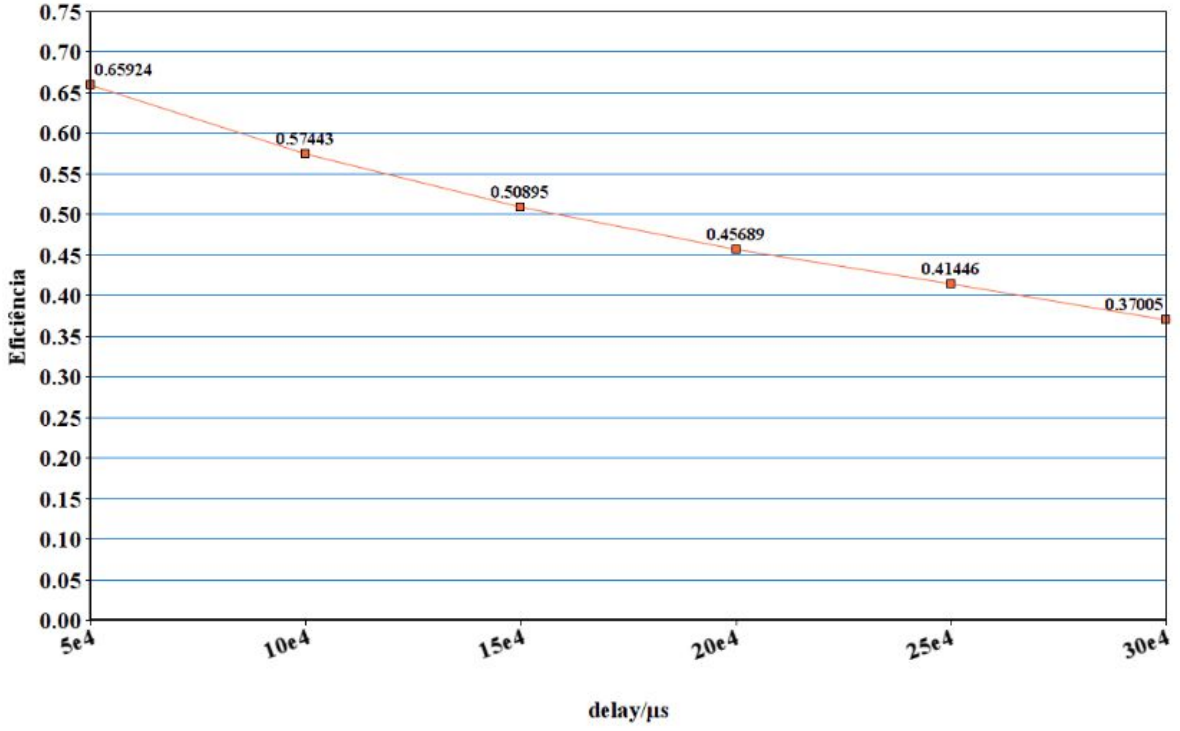
Variação da Capacidade de Ligação ou C:

Com este gráfico vemos que temos quebras na eficiência para os valores mais altos de *Baudrate*.



Variação do tempo de propagação:

Com o seguinte gráfico podemos ver que com o aumento do tempo de propagação, a eficiência da aplicação diminui como seria de esperar.



No nosso projeto foi implementado o protocolo *Stop and Wait* para conseguir detetar e resolver erros na transferência dos dados. Quando o emissor envia tramas fica à espera de uma resposta que poderá ser RR0/RR1 caso os dados tenham sido recebidos sem erros no recetor, ou REJ0/REJ1 caso o oposto tenha ocorrido para que o emissor reenvie a trama. O 0 e 1 estão sincronizados a trocar de forma alternada para conseguir um melhor tratamento dos dados duplicados. Esta foi a parte onde tivemos mais dificuldades e que achamos que a aplicação poderia estar melhor. Quando ocorre ruído na transferência e pacote de dados recebido não têm o BCC2 correto este é descartado e enviado um REJ ao emissor. Este então fica preso na máquina de estados até haver um timeout e tenta novamente reenviar a trama o que é suposto acontecer. O que falha na nossa aplicação é quando o bcc2 realmente confere, o recetor escreve os dados e envia o RR mas caso haja ruído e este RR não for corretamente recebido pelo emissor, será reenviado um trama que já foi recebido pelo recetor. Após várias tentativas não conseguimos resolver o problema de gestão de dados duplicados a tempo da entrega.

Conclusão

O tema abordado foi o protocolo de ligação de dados que consiste em elaborar um mecanismo que nos permite transferir dados entre dois sistemas de forma eficiente e viável através do uso de cabos de porta de série.

Para além da aprendizagem da estruturação dos pacotes de dados e tramas que enviamos, um dos pontos focados no desenvolvimento do projeto foi a independência entre camadas onde, por exemplo, na camada de ligação e envio de dados (protocolo.c) não é feita qualquer tipo de processamento sobre o cabeçalho dos pacotes, apenas o envio, pois essa função é exclusivamente da camada da aplicação (app.c).

Para concluir, o projeto foi desenvolvido com sucesso (à exceção de um dos pontos referidos na validação) tendo contribuído para um aprofundamento da matéria a nível teórico e principalmente prático por todos os membros do grupo.