U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Redes de Computadores

Relatório 1º Trabalho Laboratorial

Mestrado Integrado em Engenharia Informática e Computação

3MIEIC01 | Grupo 6

Francisco Gonçalves | up201704790
José Pedro Baptista | up201705255
José Martins | up201605497

Sumário

Contexto

Este é o 1º trabalho laboratorial da unidade curricular Redes de Computadores e baseia se na implementação de um protocolo de ligação de dados com mecanismo de controlo de erros Stop and Wait.

Conclusões

O projeto foi concluído com sucesso visto o trabalho desenvolvido ser capaz de efetuar o pretendido, isto é, a transferência de ficheiros entre dois computadores com o uso de portas de série, com uma robustez suficiente que permite o envio de qualquer tipo de ficheiros auxiliado de um mecanismo de controlo de erros, assegurando se assim que uma desconexão temporária da ligação ou a existência de um curto circuito não perturbam o envio da informação.

Introdução

Este trabalho tem como principal objetivo a implementação de um protocolo de transferência de dados que permita o envio de informação entre dois computadores através de uma porta de série RS-232.

Neste relatório serão apresentadas todas as escolhas tomadas durante o desenvolvimento deste projeto, sendo este estruturado da seguinte forma:

- Arquitetura: Funcionamento do protocolo e da sua interface;
- Estrutura do código: API, principais funções e sua relação com a arquitetura e principais estruturas de dados,;
- Casos de uso principais: Identificação dos principais casos de uso e sequências de chamada de funções;
- Protocolo de ligação lógica: Identificação dos principais aspetos funcionais e descrição da sua estratégia de implementação;
- Protocolo de aplicação: Identificação dos principais aspetos funcionais e descrição da sua estratégia de implementação;
- Validação: Descrição dos testes efetuados;
- Eficiência do protocolo de ligação de dados: caracterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido. A caracterização teórica de um protocolo Stop&Wait, que deverá ser usada como termo de comparação, encontra-se descrita nos slides de Ligação Lógica das aulas teóricas;
- Conclusões: síntese da informação apresentada nas seções anteriores; reflexão sobre os objectivos de aprendizagem alcançados.

Arquitetura

O código desenvolvido divide-se em duas camadas bem definidas: a camada de ligação de dados e a camada da aplicação.

A camada de ligação de dados assegura-se apenas do funcionamento da transferência de tramas assim como do estabelecimento e terminação da ligação, não existindo qualquer distinção entre pacotes de controlo e de dados. Esta trata de todos os mecanismos necessários como stuffing/destuffing, proteção de tramas, eventuais retransmissões, etc.

A camada da aplicação não conhece os detalhes do protocolo de ligação de dados, apenas a forma como acede ao serviço. Esta efetua o processamento dos pacotes a transportar em tramas de informação, usando o protocolo para os enviar e sem se preocupar como esta transferência é efetuada.

A interface protocolo-aplicação permite assim a troca de informação entre os dois computadores da forma pretendida.

Estrutura do código

API - Principais funções e sua relação com a arquitetura

De forma a estabelecer a comunicação entre as duas camadas referidas anteriormente foram desenvolvidas quatro funções que formam a API protocolo-aplicação:

```
int llopen(char *porta, enum openAS modoAbertura);

int llwrite(int fd, char * buffer, int length);

int llread(int fd, char * buffer);

int llclose(int fd);
```

Principais estruturas de dados

De modo que o protocolo de ligação de dados saiba se tem de correr as funções llopen() e llclose() no modo emissor ou receptor, foi criada a seguinte estrutura que é passada como argumento na função llopen():

```
enum openAS {TRANSMITTER, RECEIVER};
```

Casos de uso principais

Existem dois casos de uso principais sendo estes:

- Correr o programa como emissor (sender.c). Neste caso os argumentos especificados serão a porta de série e o ficheiro a transmitir, ex:

  ./sender /dev/ttyS0 pinguim.gif

- Correr o programa como recetor (receiver.c). Os argumentos especificados serão a porta de série e o nome do ficheiro a receber, ex:

  ./receiver /dev/ttyS0 pinguimCopy.gif

A sequência de chamada das principais funções no caso do emissor e suas finalidades são as seguintes:

- fopen() , abrir o ficheiro que pretendemos transmitir;
- fseek(), obter tamanho do ficheiro;
- llopen() , estabelecer ligação com o recetor;
- llwrite(), enviar a informação do ficheiro;
- fclose(), fechar o ficheiro;
- llclose(), terminar a ligação.

No caso do recetor:

- fopen() , criar o ficheiro que pretendemos receber;
- llopen() , estabelecer ligação com o emissor;
- llread(), receber a informação do ficheiro;
- fwrite(), escrever para o ficheiro criado os dados recebidos;
- fclose(), fechar o ficheiro;
- llclose(), terminar a ligação.

Protocolo de ligação lógica

Foram desenvolvidos os seguintes principais aspetos funcionais:

- Estabelecimento da ligação: Através da função llopen();
- Envio de dados: Através da função llwrite();
- Receção de dados: Através da função llread();
- Terminação da ligação: Através da função llclose();
- Stuffing e destuffing de dados: Através das funções stuff() e deStuff();
- Controlo de erros: Através do cálculo do BCC2 e com as funções calcBCC2(), stuffBCC2(), deStuffBCC2().
- Mecanismo de retransmissão: Através da função atende().

Os extratos de código que retratam estas funcionalidades encontram se no Anexo I.

Protocolo de aplicação

Foram desenvolvidos os seguintes principais aspetos funcionais:

- Criação dos pacotes de controlo para sinalizar o início e o fim da transferência do ficheiro;
- Criação dos pacotes de dados contendo fragmentos do ficheiro a transmitir;
- Envio (sender.c) e receção (receiver.c) dos pacotes de controlo e de dados;
- Leitura da informação do ficheiro a transmitir (sender.c) e consequente escrita após receção para um novo ficheiro (receiver.c);

Os extratos de código que retratam estas funcionalidades encontram se no Anexo II.

Validação

De modo a verificar o correto funcionamento do código desenvolvido foram feitos vários testes sendo que todos foram completados com sucesso. De entre os testes efetuados destacam se:

- Envio de diferentes tipos de ficheiros (.gif, .jpeg, .png, .h, .txt, ...);
- Envio de ficheiros com diversos tamanhos;
- Variação do tamanho das tramas enviadas;
- Interrupção da ligação entre as portas de série;
- Criação de ruído na ligação.

Eficiência do protocolo de ligação de dados

- Variação de FER (Frequência/Probabilidade de erro no envio de trama)
- Variação do tamanho de trama I
- Variação de baudrate (C)
- Variação de tempo de propagação

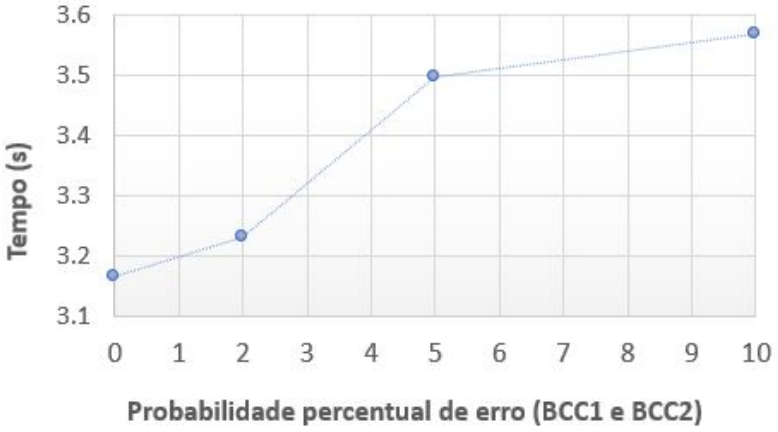
Estes 4 testes foram registados e tratados dados concretos de maneira a estudar melhor o envio. Foram repetidas medições pelo menos duas vezes sempre de maneira a ter resultados mais fiáveis.

Teste de FER

VARIAR FER		
Tamanho Ficheiro	10968	Bytes
Baudrate (C)	38400	
Tamanho Trama I	256	Bytes
Nº Tramas	44	

% ERRO BCC1	% ERRO BCC2	Tempo (s)	Tempo Médio	Nº REJs	S (R/C)
0	0	3.166155	3.165049	0	0.7219
		3.163943		0	
2	2	3.158761	3.231854	0	0.707
		3.304947		2	
5	5	3.223992	3.497123333	1	0.6534
		3.523067		4	
		3.744311		8	
10	10	3.230432	3.567893667	2	0.6404
		3.81031		9	
		3.662939		7	

x	y
0	3.165049
2	3.231854
5	3.497123333
10	3.567893667



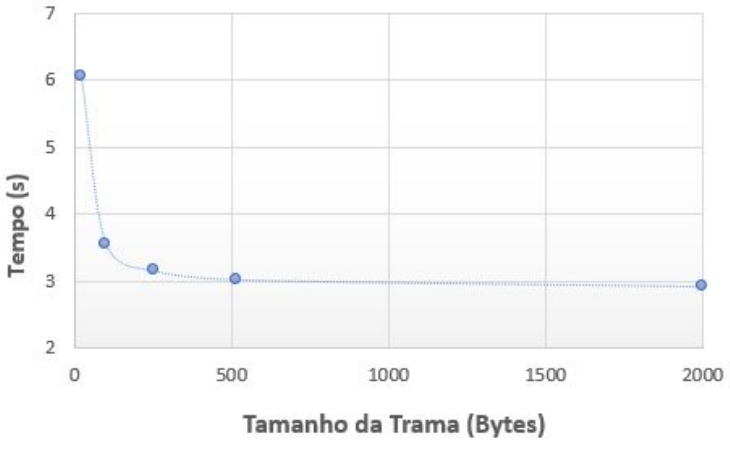
Através da criação de uma função que gera uma variável booleana (true ou false) mediante um número entre 0 e 100, criámos uma probabilidade de BCC1 e BCC2 conterem erros com o retorno dessa função. Com os resultados conclui-se que estes possíveis erros aumentam ligeiramente o tempo de envio, diminuindo S (R/C).

Teste de Variação de tamanho trama I

VARIAR TAMANHO TRAMA I		
Tamanho Ficheiro	10968	Bytes
Baudrate (C)	38400	
ERRO BCC1	0	%
ERRO BCC2	0	%

Tamanho Trama	Tempo (s)	Tempo Médio	Nº Tramas	S (R/C)
21	6.053192	6.050118	524	0.3777
	6.047044			
100	3.553648	3.5535375	111	0.643
	3.553427			
256	3.150793	3.1509045	44	0.7252
	3.151016			
515	3.023234	3.023152667	23	0.7558
	3.023139			
	3.023085			
2000	2.923388	2.923454333	7	0.7816
	2.923431			
	2.923544			

21	6.050118
100	3.5535375
256	3.1509045
515	3.023152667
2000	2.923454333



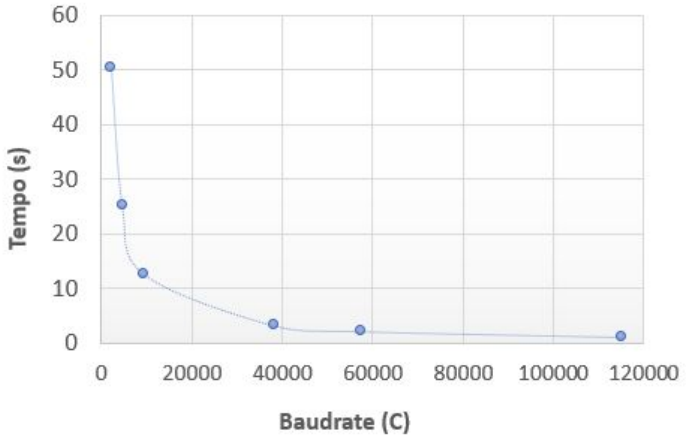
Fazendo uso da macro *Trama_I_Size* controlámos o tamanho de cada trama I. Após a realização de testes conclui-se que quanto maior for a trama, mais rápido é o envio (melhoramento não linear), resultando assim um valor de S(R/C) significativamente melhor. A relação aparenta ser hiperbólica (y=(1/x)).

Teste de variação de baudrate (C)

VARIAR BAUDRATE (C)		
Tamanho Ficheiro	10968	Bytes
Tamanho Trama I	256	Bytes
Nº Tramas	44	
ERRO BCC1	0	%
ERRO BCC2	0	%

BAUDRATE (C)	Tempo (s)	Tempo Médio	S (R/C)
2400	50.300991	50.3012145	0.726821417
	50.301438		
4800	25.163836	25.163202	0.726457626
	25.162568		
9600	12.581629	12.581104	0.726486324
	12.580579		
38400	3.151423	3.1512425	0.725110809
	3.151062		
57600	2.103446	2.102951	0.724378901
	2.102456		
115200	1.059607	1.057577	0.720199727
	1.055547		

2400	50.3012145
4800	25.163202
9600	12.581104
38400	3.1512425
57600	2.102951
115200	1.057577



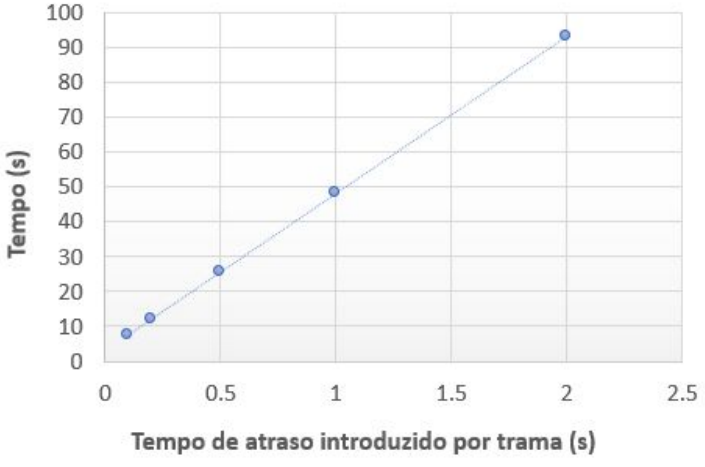
Variando o baudrate (C - capacidade da ligação, bit/s) utilizando a macro do programa, conseguimos observar que quanto maior o baudrate, mais rápido é o envio. No entanto, não se verifica uma variação significativa de S, pois apesar de C crescer, R também cresce em proporções muito semelhantes. A relação aparenta ser hiperbólica (y=(1/x)).

Teste de variação de Tempo de propagação

VARIAR T_PROP		
Tamanho Ficheiro	10968	Bytes
Tamanho Trama I	256	Bytes
Nº Tramas	44	
ERRO BCC1	0	%
ERRO BCC2	0	%
Baudrate (C)	38400	

Tempo Atraso (s)	Tempo (s)	Tempo Médio	S (R/C)
0.1	7.654926	7.654902	0.298501535
	7.654878		
0.2	12.154955	12.1551405	0.187986309
	12.155326		
0.5	25.654729	25.6551715	0.089065863
	25.655614		
1	48.155286	48.155274	0.04745067
	48.155262		
2	93.155842	93.1558835	0.024528778
	93.155925		

0.1	7.654902
0.2	12.1551405
0.5	25.6551715
1	48.155274
2	93.1558835

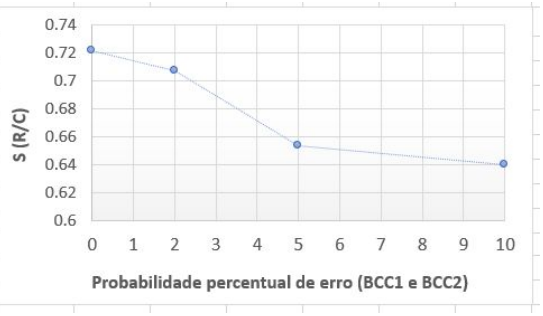


Fazendo uso da função usleep() da biblioteca unistd.h, introduzimos um atraso (em μs) em cada chamada de llwrite (cada envio de trama I). Com os valores registados e gráfico traçado, podemos observar que a relação entre o atraso em segundos e o tempo de envio é linear.

Em cima estão descritos gráficos com o tempo em y. Em baixo podem ser vistos os 4 gráficos mas com S=R/C no lugar do tempo(s). Deste modo podemos também ver como se relaciona a eficiência do envio com os valores variados de cada teste.

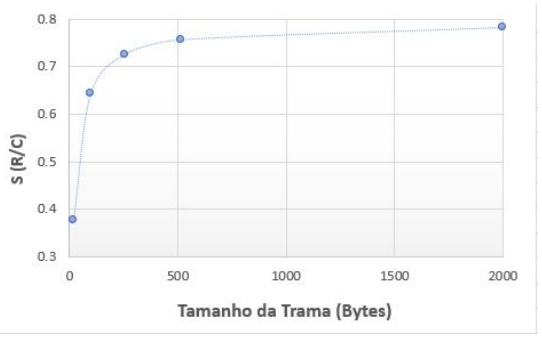
Teste de FER

x	y
0	0.7219
2	0.707
5	0.6534
10	0.6404



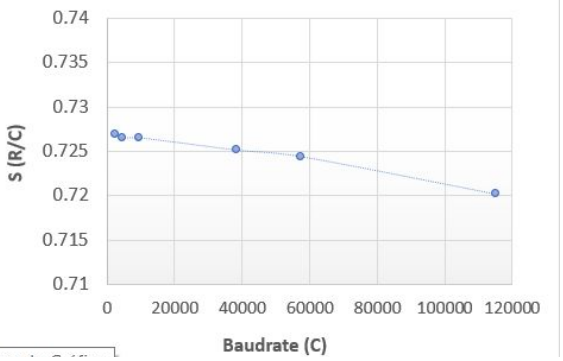
Teste de Variação de tamanho trama I

x	y
21	0.377
100	0.643
256	0.7252
515	0.7558
2000	0.7816



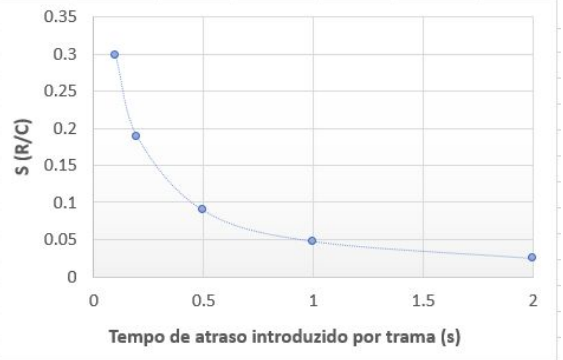
Teste de variação de baudrate (C)

x	y
2400	0.726821417
4800	0.726457626
9600	0.726486324
38400	0.725110809
57600	0.724378901
115200	0.720199727



Teste de variação de Tempo de propagação

x	y
0.1	0.298501535
0.2	0.187986309
0.5	0.089065863
1	0.04745067
2	0.024528778



Conclusões

A solução implementada divide se em duas camadas, camada de ligação de dados, sendo esta a de mais baixo nível e que faz a interação direta com a porta de série, apenas se assegurando com o mecanismo de transferência de dados, e, camada da aplicação, camada de mais alto nível que trata da parte de recolher a informação do ficheiro a enviar assim como de criar o ficheiro recebido, utilizando o protocolo de ligação de dados para fazer a transferência.

Os testes efetuados permitiram também ter uma noção prática do tipo de erros e atrasos que podem ser encontrados, que limitam a eficiência das transferências.

O grupo considera que todos os objetivos de aprendizagem foram alcançados e bem consolidados ficando assim com um bom conhecimento acerca de canais de comunicação, controlo da ligação de dados e modelos de erro e atraso.

Anexos

Anexo I - Protocolo de ligação lógica

llopen

```
int llopen(char *porta, enum openAS modoAbertura)
{
  /*
    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.
  */
  modo = modoAbertura;
  int fd = open(porta, O_RDWR | O_NOCTTY);
  if (fd < 0)
  {
    perror(porta);
    exit(-1);
  }

  if (modo == TRANSMITTER)
  {

    if (tcgetattr(fd, &oldtio) == -1)
    { /* save current port settings */
      perror("tcgetattr");
      exit(-1);
    }

    bzero(&newtio, sizeof(newtio));
    newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
    newtio.c_iflag = IGNPAR;
    newtio.c_oflag = OPOST;

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

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

```
    tcflush(fd, TCIOFLUSH);

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

    char SET[5];
    SET[0] = FLAG;
    SET[1] = A_ANS;
    SET[2] = C_SET;
    SET[3] = (A_ANS ^ C_SET);
    SET[4] = FLAG;

    unsigned char byte;
    (void)signal(SIGALRM, atende); // instala  rotina que atende
interrupcao

    do
    {
      // SEND SET
      for (size_t i = 0; i < 5; i++)
        write(fd, &SET[i], 1);

      printf("SET sent\n");

      alarm(3);
      flag = FALSE;

      int state = Start;

      while (state != STOP && !flag)
      {
        read(fd, &byte, 1); // read byte
        handleStateOfEstablishment(&state, byte, C_UA);
      }
    } while (flag && conta < 4);

    if (flag && conta == 4)
      return -1;
    else
    {
```

```
      printf("UA received\n");
      alarm(0);
      flag = FALSE;
      conta = 1;
    }
  }

  else if (modo == RECEIVER)
  {
    tcgetattr(fd, &oldtio);

    bzero(&newtio, sizeof(newtio));
    newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
    newtio.c_iflag = IGNPAR;

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

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

    tcflush(fd, TCIOFLUSH);
    tcsetattr(fd, TCSANOW, &newtio);

    // RECEIVE SET
    unsigned char byte;
    int state = Start;

    while (state != STOP)
    {
      read(fd, &byte, 1); // read byte
      handleStateOfEstablishment(&state, byte, C_SET);
    }

    printf("SET received\n");

    // SEND UA
    char UA[5];
    UA[0] = FLAG;
    UA[1] = A_ANS;
    UA[2] = C_UA;
    UA[3] = (A_ANS ^ C_UA);
    UA[4] = FLAG;
```

```
    for (size_t i = 0; i < 5; i++)
      write(fd, &UA[i], 1);

    printf("UA sent\n");
  }
  else
    return -1;

  return fd;
}
```

llwrite

```
int llwrite(int fd, char *buffer, int length)
{
  // BYTE STUFFING com buffer
  size_t dataNewSize = 0;
  size_t BCC2NewSize = 0;

  unsigned char *BCC2 = (unsigned char *)malloc(sizeof(unsigned char *));
  char *dataStuffed = stuff(buffer, length, &dataNewSize, BCC2);
  char *newBCC2 = stuffBCC2(*BCC2, &BCC2NewSize);

  // SEND I
  int tramaSize = 5 + dataNewSize + BCC2NewSize;
  char I[tramaSize];
  I[0] = FLAG;
  I[1] = A_ANS;

  if ((Ns % 2) == 0) I[2] = C_Ns0;
  else I[2] = C_Ns1;

  I[3] = I[1] ^ I[2];

  for (size_t i = 0; i < dataNewSize; i++)
    I[4 + i] = dataStuffed[i];

  for (size_t i = 0; i < BCC2NewSize; i++)
    I[4 + dataNewSize + i] = newBCC2[i];

  I[tramaSize - 1] = FLAG;
```

```
  // SEND I Trama
  int tramaSent = FALSE;
  (void)signal(SIGALRM, atende); //instala rotina que atende interrupcao


  do
  {
    if(conta > 3)
        return -1;

    for (size_t i = 0; i < tramaSize; i++)
      write(fd, &I[i], 1);

    printf("I trama sent with Ns = %d\n", Ns);
    alarm(3);
    flag = FALSE;

    // RECEIVE C
    unsigned char byte;
    int state = Start;

    if ((Ns % 2) == 0)
    {
      while (state != STOP && !flag)
      {
          read(fd, &byte, 1); // read byte
          handleStateOfTransmissionSender(&state, byte, C_RR_Nr1,
C_REJ_Nr1);
          if (state == C_RR)
          {
              printf("RR received\n");
              tramaSent = TRUE;
          }

          if (state == C_REJ)
          {
              printf("REJ received\n");
              flag = TRUE;
          }
      }
    }
```

```
    else
    {
        while (state != STOP && !flag)
        {
            read(fd, &byte, 1); // read byte
            handleStateOfTransmissionSender(&state, byte, C_RR_Nr0,
C_REJ_Nr0);
            if (state == C_RR){
                printf("RR received\n");
                tramaSent = TRUE;
            }

            if (state == C_REJ){
                printf("REJ received\n");
                flag = TRUE;
            }
        }
    }

  } while (!tramaSent || (flag && conta < 4));

  if(conta > 3) return -1;
  else{
    alarm(0);
    flag = FALSE;
    conta = 1;
  }

  printf("\n");
  Ns++;
  return length;
}
```

llread

```
int llread(int fd, char *buffer)
{
  // RECEIVE I
  unsigned char byte;
  int state = Start;
  char data[MAX_LEN] = "";
  size_t dataSize = 0;
  int nrByteReceived = 0;
```

```
int descartarTrama = FALSE;



while (state != STOP)
{
  read(fd, &byte, 1); // read byte
  nrByteReceived++;

  if (state == DATA && byte != FLAG)
  {
    data[dataSize] = byte;
    dataSize++;
  }

  if (nrByteReceived == 3)
  {
    if (((Nr % 2) == 1) && byte == C_Ns1)
      descartarTrama = TRUE;
    else if (((Nr % 2) == 0) && byte == C_Ns0)
      descartarTrama = TRUE;
  }

  if (descartarTrama)
  {
    if ((Nr % 2) == 1)
      handleStateOfTransmissionReceiver(&state, byte, C_Ns1);
    else
      handleStateOfTransmissionReceiver(&state, byte, C_Ns0);
  }
  else
  {
    if ((Nr % 2) == 1)
      handleStateOfTransmissionReceiver(&state, byte, C_Ns0);
    else
      handleStateOfTransmissionReceiver(&state, byte, C_Ns1);
  }

  if (dataTransferOver)
  {
    if (byte == C_DISC)
    {
      disconnect = 1;
```

```
      printf("DISC received\n");
    }
    else
      printf("# FINAL BYTE: 0x%02x\n", byte);
  }
}

if (disconnect == 1)
  return 0;
else
{
  // destuffing data
  size_t dataNewSize = 0;
  char *dataDeStuffed = deStuff(data, dataSize, &dataNewSize);
  char BCC2Received = dataDeStuffed[dataNewSize - 1];
  char BCC2Calculated = calcBCC2(dataDeStuffed, dataNewSize - 1);

  // Se BCC2 valid
  if (BCC2Received == BCC2Calculated)
  {
    // enviar RR
    unsigned char RR[5];
    RR[0] = FLAG;
    RR[1] = A_ANS;

    if ((Nr % 2) == 1)
      RR[2] = C_RR_Nr1;
    else
      RR[2] = C_RR_Nr0;

    RR[3] = RR[1] ^ RR[2];
    RR[4] = FLAG;

    for (size_t i = 0; i < 5; i++)
      write(fd, &RR[i], 1);

    if (descartarTrama != TRUE)
    {
      Nr++;

      if (!TramaAlreadyReceived)
      {
        printf("TRAMA RECEIVED\n");
```

```
        for (int i = 0; i < dataNewSize - 1; i++)
        {
          buffer[i] = dataDeStuffed[i];
          printf("%x |", buffer[i]);
        }

        printf("\n\n");

        if (buffer[0] == '3')
          dataTransferOver = TRUE;

        TramaAlreadyReceived = FALSE;
        return dataNewSize - 1;
      }
      else
      {
        TramaAlreadyReceived = FALSE;
        return -2;
      }
    }
    else
    {
      TramaAlreadyReceived = TRUE;
      Nr--;
      return -2;
    }
  }
  else
  {
    // enviar REJ
    unsigned char REJ[5];
    REJ[0] = FLAG;
    REJ[1] = A_ANS;

    if ((Nr % 2) == 1)
      REJ[2] = C_REJ_Nr1;
    else
      REJ[2] = C_REJ_Nr0;

    REJ[3] = REJ[1] ^ REJ[2];
    REJ[4] = FLAG;
```

```
      for (size_t i = 0; i < 5; i++)
        write(fd, &REJ[i], 1);

      return REJ_SENT;
    }
  }
}
```

llclose

```
int llclose(int fd)
{
  if (modo == RECEIVER)
  {
    int state = Start;
    unsigned char byte;

    // SEND DISC
    char DISC[5];
    DISC[0] = FLAG;
    DISC[1] = A_SND;
    DISC[2] = C_DISC;
    DISC[3] = DISC[1] ^ DISC[2];
    DISC[4] = FLAG;

    for (size_t i = 0; i < 5; i++)
      write(fd, &DISC[i], 1);

    printf("DISC sent\n");

    // RECEIVE UA
    state = Start;
    while (state != STOP)
    {
      read(fd, &byte, 1); // read byte
      handleStateOfTermination(&state, byte, C_UA);
    }
    printf("UA received\n");

    sleep(1);

    tcsetattr(fd, TCSANOW, &oldtio);
    close(fd);
```

```
    return fd;
}
else if (modo == TRANSMITTER)
{
    // SEND DISC
    char DISC[5];
    DISC[0] = FLAG;
    DISC[1] = A_ANS;
    DISC[2] = C_DISC;
    DISC[3] = DISC[1] ^ DISC[2];
    DISC[4] = FLAG;

    for (size_t i = 0; i < 5; i++)
        write(fd, &DISC[i], 1);

    printf("DISC sent\n");

    int state = Start;
    unsigned char byte;

    // RECEIVE DISC
    while (state != STOP)
    {
        read(fd, &byte, 1); // read byte
        handleStateOfTermination(&state, byte, C_DISC);
    }
    printf("DISC received\n");

    // SEND UA
    char UA[5];
    UA[0] = FLAG;
    UA[1] = A_SND;
    UA[2] = C_UA;
    UA[3] = UA[1] ^ UA[2];
    UA[4] = FLAG;

    for (size_t i = 0; i < 5; i++)
        write(fd, &UA[i], 1);

    printf("UA sent\n");

    sleep(1);
```

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

stuff

```
char *stuff(const char *dados, size_t currentSize, size_t *newSize,
unsigned char *BCC2)
{
  char *temp = (char *)malloc(2 * currentSize * sizeof(char));
  size_t t = 0;
  *BCC2 = dados[0];

  for (size_t i = 0; i < currentSize; i++)
  {
    if (i != 0)
      *BCC2 ^= dados[i];

    char byteAtual = dados[i];

    if (byteAtual == FLAG)
    {
      temp[t] = ESCAPE;
      temp[t + 1] = xorFLAG;
      t += 2;
    }
    else if (byteAtual == ESCAPE)
    {
      temp[t] = ESCAPE;
      temp[t + 1] = xorESCAPE;
      t += 2;
    }
    else
    {
      temp[t] = byteAtual;
```

```
      t++;
    }
  }

  *newSize = t;
  dados++;
  return temp;
}
```

deStuff

```
char *deStuff(const char *dados, size_t currentSize, size_t *newSize)
{
  char *temp = (char *)malloc(currentSize * sizeof(char));
  size_t t = 0;

  for (size_t i = 0; i < currentSize; i++)
  {
    char byteAtual = dados[i];

    if (byteAtual == ESCAPE)
    {
      char proximoByte = dados[i + 1];

      if (proximoByte == xorFLAG) //It was a FLAG
      {
        temp[t] = FLAG;
        t++;
      }
      else if (proximoByte == xorESCAPE) //It was a ESCAPE
      {
        temp[t] = ESCAPE;
        t++;
      }

      i++;
    }
    else
    {
      // temp = (char *) malloc (sizeof(char *));
      temp[t] = byteAtual;
      t++;
    }
```

```
  }
  *newSize = t;
  return temp;
}
```

calcBCC2

```
char calcBCC2(const char *dados, size_t currentSize)
{
  char BCC2 = dados[0];
  for (size_t i = 1; i < currentSize; i++)
    BCC2 ^= dados[i];

  return BCC2;
}
```

stuffBCC2

```
char *stuffBCC2(const unsigned char BCC2, size_t *newSize)
{
  char *temp;

  if (BCC2 == FLAG)
  {
    *newSize = 2;
    temp = (char *)malloc(2 * sizeof(char *));
    temp[0] = ESCAPE;
    temp[1] = xorFLAG;
  }
  else if (BCC2 == ESCAPE)
  {
    *newSize = 2;
    temp = (char *)malloc(2 * sizeof(char *));
    temp[0] = ESCAPE;
    temp[1] = xorESCAPE;
  }
  else
  {
    *newSize = 1;
    temp = (char *)malloc(sizeof(char *));
    temp[0] = (char)BCC2;
  }
```

```
  return temp;
}
```

deStuffBCC2

```
unsigned char *deStuffBCC2(const unsigned char *BCC2)
{
  unsigned char *temp;
  unsigned char primeiroByte = BCC2[0];

  if (primeiroByte == ESCAPE)
  {
    unsigned char segundoByte = BCC2[1];

    if (segundoByte == xorFLAG) //It was a FLAG
    {
      temp = (unsigned char *)malloc(sizeof(unsigned char *));
      temp[0] = FLAG;
    }
    else if (BCC2[1] == xorESCAPE) //It was a ESCAPE
    {
      temp = (unsigned char *)malloc(sizeof(unsigned char *));
      temp[0] = ESCAPE;
    }
  }

  else
  {
    temp = (unsigned char *)malloc(sizeof(unsigned char *));
    temp[0] = primeiroByte;
  }

  return temp;
}
```

atende

```
int flag = FALSE, conta = 1;
void atende(int sig) // atende alarme
{
  printf("alarme # %d\n", conta);
  flag = TRUE;
  conta++;
}
```

Anexo II - Protocolo de aplicação

Criação dos pacotes de controlo para sinalizar o início e o fim da transferência do ficheiro e envio

```
// Create Start and End control package
  char startControl[10] = "20";
  char endControl[10] = "30";
  int controlLengt;
  /**
   *  2^8 = 256
   *  2^16 = 65536
   *  2^32 = 16777215
   */


  if (fileSize < 256)
  {
    append(startControl, '1');
    append(startControl, (unsigned char)fileSize);
    append(endControl, '1');
    append(endControl, (unsigned char)fileSize);
    controlLengt = 4;
  }
  else if (fileSize < 65536 && fileSize >= 256)
  {
    append(startControl, '2');
    append(startControl, (unsigned char)(fileSize >> 8));
    append(startControl, (unsigned char)fileSize);
    append(endControl, '2');
    append(endControl, (unsigned char)(fileSize >> 8));
    append(endControl, (unsigned char)fileSize);
    controlLengt = 5;
  }
  else if (fileSize < 16777215 && fileSize >= 65536)
  {
    append(startControl, '3');
    append(startControl, (unsigned char)(fileSize >> 16));
    append(startControl, (unsigned char)(fileSize >> 8));
    append(startControl, (unsigned char)fileSize);
    append(endControl, '3');
    append(endControl, (unsigned char)(fileSize >> 16));
```

```
    append(endControl, (unsigned char)(fileSize >> 8));
    append(endControl, (unsigned char)fileSize);
    controlLengt = 6;
  }
  else
  {
    printf("File Size too big!\n");
    return 1;
  }
```

```
// Send Start Control Package

  int llwrite_return_value;

  llwrite_return_value = llwrite(fd, startControl, controlLengt);
    if(llwrite_return_value == -1)
    {
        printf("Timeout!\n");
        return -1;
    }
```

```
  // Send End Control Package
  llwrite_return_value = llwrite(fd, endControl, controlLengt);

  if(llwrite_return_value == -1)
    {
        printf("Timeout!\n");
        return -1;
    }
```

Criação dos pacotes de dados contendo fragmentos do ficheiro a transmitir e envio

```
// Get File Bytes
  char *fileInfo = (char *)malloc(fileSize);
  fread(fileInfo, sizeof(char), fileSize, filefd);

  int nrIterationsNeeded = fileSize / TRAMA_I_SIZE;
  int remainderBytes = fileSize % TRAMA_I_SIZE;

  int i = 0;
  int nrFileByte = 0;

  // Send file data
  for (; i < nrIterationsNeeded; i++)
  {
    char data[2 * TRAMA_I_SIZE + 4];
    data[0] = 1;
    data[1] = i % (TRAMA_I_SIZE - 1);
    data[2] = TRAMA_I_SIZE / 256;
    data[3] = TRAMA_I_SIZE % 256;

    for (int j = 0; j < TRAMA_I_SIZE; j++)
    {
      data[4 + j] = fileInfo[nrFileByte];
      nrFileByte++;
    }

    llwrite_return_value = llwrite(fd, data, (TRAMA_I_SIZE + 4));

    if(llwrite_return_value == -1)
    {
        printf("Timeout!\n");
        return -1;
    }
  }

  // Send remainder of bytes
  if (remainderBytes)
  {
    char data[remainderBytes + 10];
    data[0] = 1;
```

```
    data[1] = i % (TRAMA_I_SIZE - 1);
    data[2] = 0; // é preciso alterar de acordo com valor definido em
cima (256)
    data[3] = remainderBytes;

    for (int j = 0; j < remainderBytes; j++)
    {
      data[4 + j] = fileInfo[nrFileByte];
      nrFileByte++;
    }

    llwrite_return_value = llwrite(fd, data, 4 + remainderBytes);

    if(llwrite_return_value == -1)
    {
        printf("Timeout!\n");
        return -1;
    }
  }
```

Leitura da informação do ficheiro a transmitir

```
 // Get File Bytes
  char *fileInfo = (char *)malloc(fileSize);
  fread(fileInfo, sizeof(char), fileSize, filefd);
```

Receção dos pacotes de controlo e escrita para um novo ficheiro

```
// READ INFORMATION
    char buffer[MAX_LEN];
    int llread_return_value; // if == 0, DISC was received

    // Get Start Control Package
    llread_return_value = llread(fd, buffer);

    do
    {
        llread_return_value = llread(fd, buffer);

        if (llread_return_value != REJ_SENT)
            if (buffer[0] == 1) // Write data to file
                for (int i = 4; i < llread_return_value; i++)
                    fwrite(&buffer[i], 1, 1, filefd);
```

```
        if (llread_return_value == -1)
            return 1;

    } while (llread_return_value != 0);
```