Trabalho prático 1

Redes e Computadores

Ligação de dados

Trabalho realizado por:

Nuno Oliveira up201806525@fe.up.pt

Luís Pinto up201806206@fe.up.pt

Sumário

Este trabalho foi realizado no âmbito da disciplina de Redes de Computadores com o objetivo de estudar a transferência de dados através duma aplicação com recurso a um protocolo de ligação de dados. Tanto a aplicação como o protocolo de ligação e o controlo de erros foram implementados por nós.

O trabalho realizado permitiu-nos alargar os conhecimentos sobre a gestão da comunicação entre computadores, os vários parâmetros que influenciam a velocidade e a fiabilidade da transmissão, a gestão dos erros e formas de manter a integridade da informação enviada.

Introdução

O trabalho prático foi dividido em duas grandes partes: desenvolver a aplicação de teste e desenvolver o protocolo de ligação de dados. O protocolo deve fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um cabo de série. Por sua vez, o objetivo da aplicação seria desenvolver um protocolo de aplicação relativamente simples para transferir ficheiros fazendo uso do protocolo de ligação de dados.

Este relatório visa explicar a implementação das funcionalidades já mencionadas bem como a teoria do trabalho sendo composto pelas seguintes secções (por ordem):

- Arquitetura - blocos funcionais e interfaces.
- Estrutura do código - APIs, principais estruturas de dados, principais funções e sua relação com a arquitetura.
- Casos de uso principais - identificação; sequências de chamada de funções.
- Protocolo de ligação lógica - identificação dos principais aspetos funcionais; descrição da estratégia de implementação destes aspetos com apresentação de extratos de código.
- Protocolo de aplicação - identificação dos principais aspetos funcionais; descrição da estratégia de implementação destes aspetos com apresentação de extratos de código.
- Validação - descrição dos testes efetuados com apresentação quantificada dos resultados.
- Eficiência do protocolo de ligação de dados - caraterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido e a caracterização teórica de um protocolo *Stop&Wait.*
- Conclusões - síntese da informação apresentada nas secções anteriores; reflexão sobre os objetivos de aprendizagem alcançados.
- Anexos

Arquitetura

Funcionalmente, tanto a aplicação como o protocolo têm duas maneiras distintas de execução: como emissor do ficheiro e como recetor do mesmo. A interface dispõe de quatro funções. Duas para estabelecer e terminar a ligação e outras duas para enviar e receber dados. Cabe à aplicação fazer uso dessas funções para estabelecer a conexão e transferir o ficheiro.

Existe uma total separação entre a camada de ligação de dados e a aplicação, apenas as funções da API permitem a comunicação entre as duas camadas

Estrutura do código

Principais estruturas de dados:

A *struct* linkLayers é usada pelo protocolo de ligação de dados. Esta contém vários valores importantes como o caminho da porta, *baudrate* da porta série, o número de sequência para usar ao criar e ler tramas , o tempo de espera do alarme, o número de tentativas máximo de envio de uma trama e o *file descriptor* da porta série depois de aberta. As estruturas são gravadas numa lista para que seja possível guardar as configurações das várias ligações de dados, caso a aplicação queira enviar vários ficheiros por várias portas**.**

```
typedef struct linkLayers
{
    char port[20];
    int baudRate;
    unsigned int sequenceNumber;
    unsigned int timeout;
    unsigned int numTransmissions;
    char frame[TRAMA_SIZE];
    int fd;
}linkLayer;
```

É também usado um enum que contém cada estado das várias máquinas de estados usadas neste projeto para simplificar a leitura das mesmas:

```
typedef enum states{
    START = 0,
    FLAGRCV = 1,
    ARCV = 2,
    CRCV = 3,
    BCC = 4,
    STOP = 5
}state;
```

```
typedef enum packetType{
    CONTROL = 0,
    DATA = 1
}packetType;
```

```
typedef union packet_u{
    dataPacket_s d;
    controlPacket_s c;
}packet_u;
```

```
typedef struct dataPacket_s{
    uint8_t seqNumber;
    uint16_t dataSize;
    char *data;
}dataPacket_s;
```

Na camada da aplicação é usada uma union para guardar cada pacote lido como de dados ou de controlo e um enum para facilitar a identificação desse mesmo pacote. Cada pacote será guardado na struct respetiva junto com a informação relevante para ser usada depois.

```
typedef struct controlPacket_s{
    char *fileName;
    uint64_t *fileSize;
    uint8_t end;
}controlPacket_s;
```

Ao longo do código fez-se também uso de múltiplas macros definidas no ficheiro macro.h.

A interface protocolo-aplicação dispõe das seguintes funções:

- int llopen() – estabelece a ligação entre o emissor e recetor.
- int llclose() – termina a ligação entre o emissor e o recetor.
- int llwrite() – escreve para a porta série uma trama e fica à espera de resposta.
- int llread() – lê da porta série uma trama e envia uma resposta de acordo com o que leu.

As principais funções da aplicação para além da *main* que as chama são:

- int sendFile() – cria as tramas de controlo e informação relativas ao ficheiro, que são passadas à llwrite() uma a uma para serem enviadas.

- int receiveFile() – chama llread() para receber as tramas de controlo e informação uma a uma e tenta criar o ficheiro.
- int readPacket() – lê uma trama de controlo ou informação. Devolve no argumento packet a union que contém a informação lida
- void controlPacket() – cria um pacote de controlo.
- void dataPacket() – cria um pacote de dados.

O protocolo de ligação de dados para implementar as funções da interface protocolo-aplicação usa também:

- int byteStuff() – aplica o mecanismo de *byte stuffing* a uma trama.
- int byteDeStuff() – realiza a operação inversa de *byte stuffing*.
- int infoPacket() – cria uma trama de informação.
- uint8_t getBCC2() – obtém o valor do BCC2 duma trama.
- int setTermIO() – configura a porta série.
- int setupLinkLayer() - preenche a struct linkLayer.
- int receive() – espera por uma trama de supervisão.
- int send_receive() – envia uma trama de supervisão e aguarda resposta.
- void setHeader() – cria uma trama de supervisão ou não numerada.

Casos de uso principais

Os dois casos de uso são transmissão e receção de dados embora ambos sejam interdependentes.

Transmissão de dados:

- O utilizador corre o programa do tipo:

  ./app <0 para sinalizar o emissor> <caminho do ficheiro a enviar> <numero da porta a usar> [tamanho da trama]

  É chamada a função llopen() que estabelece a ligação entre emissor e recetor. Se não for possível o programa termina e retorna -1. Se llopen() tiver sucesso é chamada a função sendFile() que cria as tramas de controlo e as tramas de informação lendo-as diretamente do ficheiro. Estas tramas são enviadas uma a uma usando llwrite(). Só é enviada a próxima trama quando o llwrite() retornar sucesso (um valor igual ou superior a 0). No fim é chamada a llclose() para terminar a ligação.

  Para o caso da receção de dados o argumento é do tipo:

  ./app <1 para sinalizar o recetor> <numero da porta a usar> [tamanho da trama]

  Depois de chamado o llopen() é chamada a função receiveFile() que chama o llread(), recebendo os conteúdos escritos na porta série. Se as tramas chegarem com sucesso é escrito no ficheiro de destino o seu conteúdo, No final é chamado o llclose() para fechar a conexão.

Exceções incluem falhas no estabelecimento da ligação que implicam terminar a aplicação e erros a tentar enviar/receber o ficheiro que são maioritariamente tratados pelo protocolo de ligação.

Protocolo de ligação lógica

No protocolo de ligação de dados foram implementadas as quatro funções pedidas.

llopen():

Esta função recebe como argumentos o modo em que o processo está correr bem como a porta de onde pretende ler/escrever [5]. Dependendo do modo de execução serão executados blocos de código diferentes. Inicialmente não há diferenças nos blocos pois é necessário preencher a struct linkLayer, abrir efetivamente a porta série e configurá-la. Só depois os blocos diferem, dado que o emissor envia uma trama de supervisão com o valor SET e fica à espera duma resposta com o valor UA usando a função send_receive() [6]. Já o recetor fica inicialmente à espera do SET usando a função receive() e se esta retornar com sucesso escreve a trama com o valor de UA [7]. Se algum destes passos não correr como esperado a função retorna -1 indicando erro caso, contrário a conexão foi estabelecida com sucesso devolvendo o ID da ligação.

llwrite():

Recebendo o ID da ligação, um array que contém um pacote de dados ou controlo e o tamanho do mesmo [8], o llwrite() começa por subscrever o alarme e declarar as variáveis necessárias para tratar esse array [9]. O primeiro passo é obter e escrever o BCC2. A seguir é feito o byte stuffing e com o resultado é criado um pacote de informação adicionando os cabeçalhos adequados. É atualizado o número de sequência na linkLayer. Após tudo isto a trama está finalmente pronta a ser enviada e usando o alarme fica-se à espera duma resposta que deve conter o valor RR durante o tempo guardado na linkLayer [10] . Caso não receba resposta após o sinal do alarme ou a resposta contenha REJ, a função tenta escrever a trama novamente. Este ciclo é repetido o número de vezes guardado na linkLayer out até que receba a resposta positiva. Em caso de sucesso é devolvido o número de caracteres escritos, caso contrário é devolvido -1, indicando erro.

llread():

A esta função é passado o Id da conexão bem como o array onde deve ser escrita a trama lida após esta tenha sido tratada [11]. A primeira coisa que se encontra é uma máquina de estados que lê e guarda a informação da trama até receber a segunda flag (0x7E) que aponta o fim da trama [12]. Após sair da máquina de estados é feito o *destuffing* da informação e verificação de que recebe o mesmo BCC2 que obtém pela trama lida para garantir que não há erros de qualquer natureza [13]. Se tudo correr bem é escrita uma trama de supervisão com o valor RR e retorna o número de carateres lidos para indicar sucesso. Se houver erro em algum destes passos em vez de RR a trama escrita terá o valor REJ [14]. A função retorna à máquina de estados de forma a ler de novo a trama que será recebida.

llclose():

Identicamente ao llopen() esta função recebe o modo de execução e consoante o valor executa um bloco diferente de código [15]. Se for emissor envia uma trama de supervisão com o valor DISC e fica à espera de uma resposta com o mesmo valor para voltar a enviar uma trama desta vez como valor UA e terminar [16]. Já o recetor recebe o DISC e envia outro DISC, terminando de seguida. Ambas as funções usam a função send_receive() para ler e escrever [17].

As funções send_receive() e receive(): estas funções são semelhantes, ambas fazem uso duma máquina de estados para ler uma trama de supervisão e retornam 0 caso recebam o

valor esperado (passado como segundo argumento). A diferença está no facto da primeira função escrever uma trama de supervisão com o valor recebido no terceiro argumento e só depois entra na máquina de estados.

O uso de alarme nas funções send_receive() e llwrite(): O alarme é usado da mesma maneira nas duas funções para se poder esperar um determinado tempo por uma resposta, guardado na linkLayer [18]. Assim nas duas é subscrito o alarme tendo como handler do sinal a função atende(). Esta função quando executada coloca uma flag a 1 e itera a variável que contém o número de tentativas de leitura. A função que está a usar o alarme apenas lê da porta série, leitura esta feita num loop infinito. Caso a flag seja posta a 1 pelo alarme é escrita de novo na porta séria a informação sobre a qual se espera confirmação de receção.

De forma a tornar a configuração mais intuitiva, foram fornecidas formas de alterar o baudRate e o tamanho da trama. No ficheiro headers/link.h estão definidas as constantes TRAMA_SIZE e BAUDRATE,ERROR_PROB e PROPAGATION_DELAY. . A TRAMA_SIZE define o valor default do tamanho da trama, este valor é usado como default caso não seja especificado o tamanho de trama nos argumentos da consola. Caso seja especificado na consola, o valor da constante TRAMA_SIZE é ignorado. O baudrate necessita de ser configurado no código, alterando a constante BAUDRATE. Caso seja necessário alterar a probabilidade de erros ou o atraso de propagação basta alterar as constantes ERROR_PROB e PROPAGATION_DELAY.

Protocolo de aplicação

sendFile(): Esta função recebe a linkLayer a passar ao llwrite() e o nome do ficheiro a enviar [20]. Com esta informação o primeiro passo é abrir o ficheiro para que se possa lê-lo e criar o primeiro pacote de controlo com as informações relevantes para indicar o início do ficheiro passando-o para a função llwrite() [21]. Após confirmado o envio do pacote de controlo está na altura de ler o ficheiro bloco a bloco. É criado um pacote de dados com o bloco lido, e este é enviado pela função llwrite() ficando à espera do retorno [22]. O próximo bloco só é enviado após o protocolo de ligação terminar com sucesso. Cada bloco tem blocos de tamanho definido no inicio do programa tirando o último que pode ter menos. Depois de enviado o ficheiro na integra é enviado outro pacote de controlo que simboliza o fim do ficheiro [23].

receiveFile(): [24] O receiveFile() consiste num ciclo que chama continuamente llread() para ler bytes recebidos, usando uma máquina de estados. Depois de lidos os bytes de um pacote é usada a função readPacket() que processa os pacotes que recebe. Se o pacote for de controlo que marca o início de um ficheiro, é criado um ficheiro com o nome indicado no pacote [25], se for um pacote de dados, os dados são escritos no ficheiro [26].

Validação

Testes efetuados:

- Envio do pinguim.gif presente no moodle com 10968B
- Envio de um png com 504KB
- Envio de um ficheiro de vídeo com 8.8MB
- Geração de ruído na ligação
- Interrupção da ligação
- Introdução de erros aleatórios nas tramas

- Variação dos valores do baudrate, tamanho da trama, probabilidade de geração de erros aleatórios

Resultados:

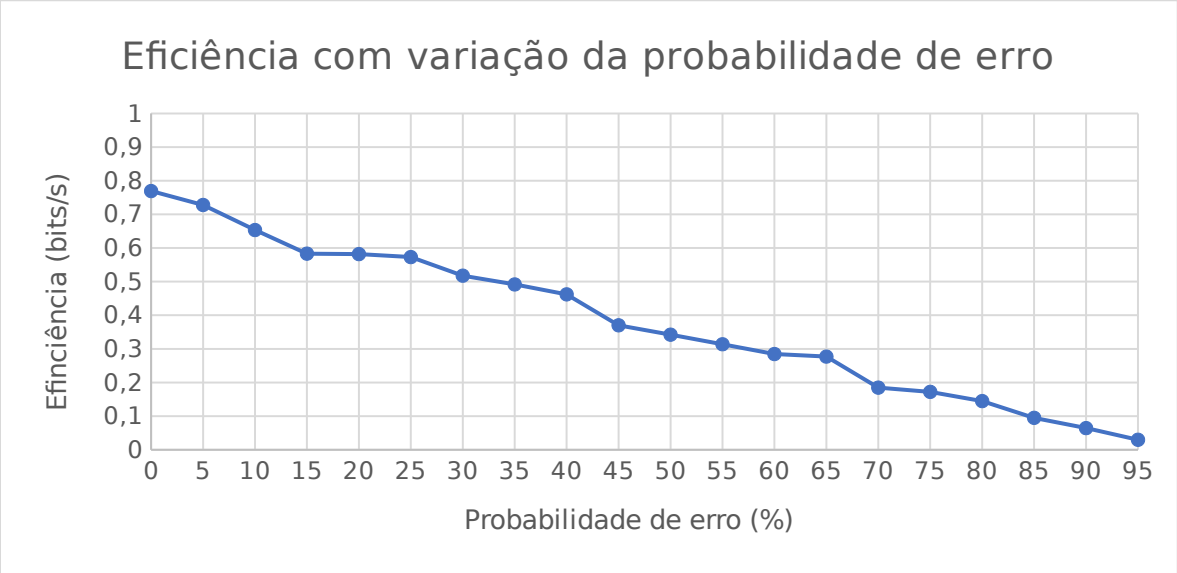
Com uma trama de tamanho 512 bytes e velocidade de propagação de 115200 bits/s um ficheiro vazio, o gif, png e o vídeo mencionados demoram respetivamente 0.015s [1], 1,021s [2], 46,262s [3], 782.175s [4]. O valor teórico para o gif seria 0.76s, valor que corresponde a 75% do valor obtido mas é preciso ter em conta que para alem dos cabeçalhos é feito byte stuffing o que aumenta o valor real de bytes a enviar. O protocolo responde bem também à geração de ruído e interrupções na ligação devido ao controlo de erros e ao mecanismo Stop&Wait conseguindo obter o ficheiro intacto. Funciona também com qualquer tamanho da trama desde que seja superior a 0 e inteiro.

Eficiência do protocolo de ligação de dados

No calculo da eficiência por default foi usado o tamanho da trama a 256 bytes, um baudrate de 38400 bits/s, sem atrasos no processamento da trama e 10% de probabilidade de geração de erro na mesma. Importante também mencionar que a eficiência será sempre inferior a 1 por vários motivos, como, por exemplo, escrever mais bytes do que realmente existem devido a byte stuffing e cabeçalhos, aos tempos de processamento e de espera por respostas e os erros introduzidos aleatoriamente. Para diminuir os erros estatísticos cada valor usado na tabela é resultado da média de 5 medições. De notar que utilizando as equações associadas ao protocolo Stop&Wait, estes valores default definem o teto máximo de eficiência como 0.9, Assumindo o tempo de propagação 0 e probabilidade de erro 10%(0.1).

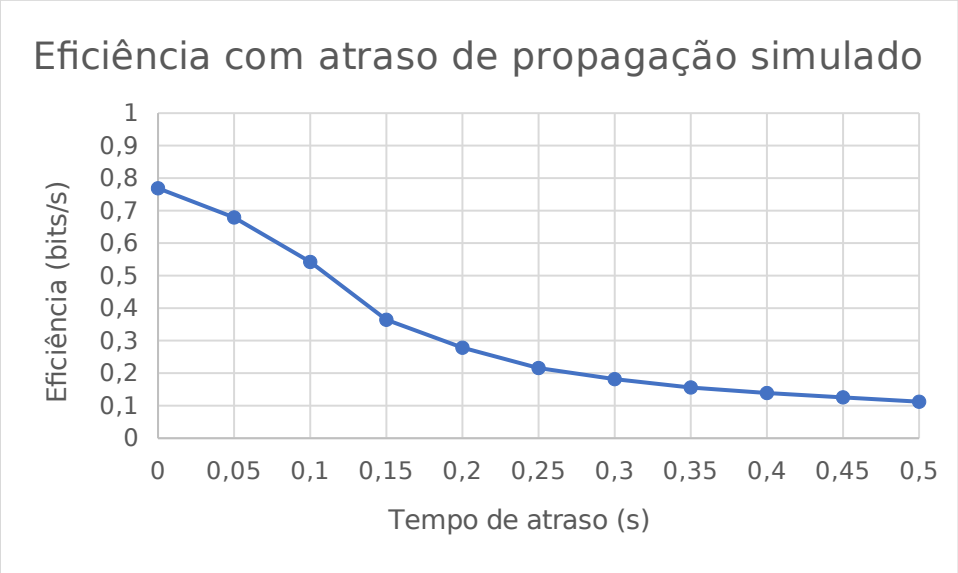
Estão em anexo os valores específicos associados aos gráficos a seguir apresentados.

Variação do FER: [27]



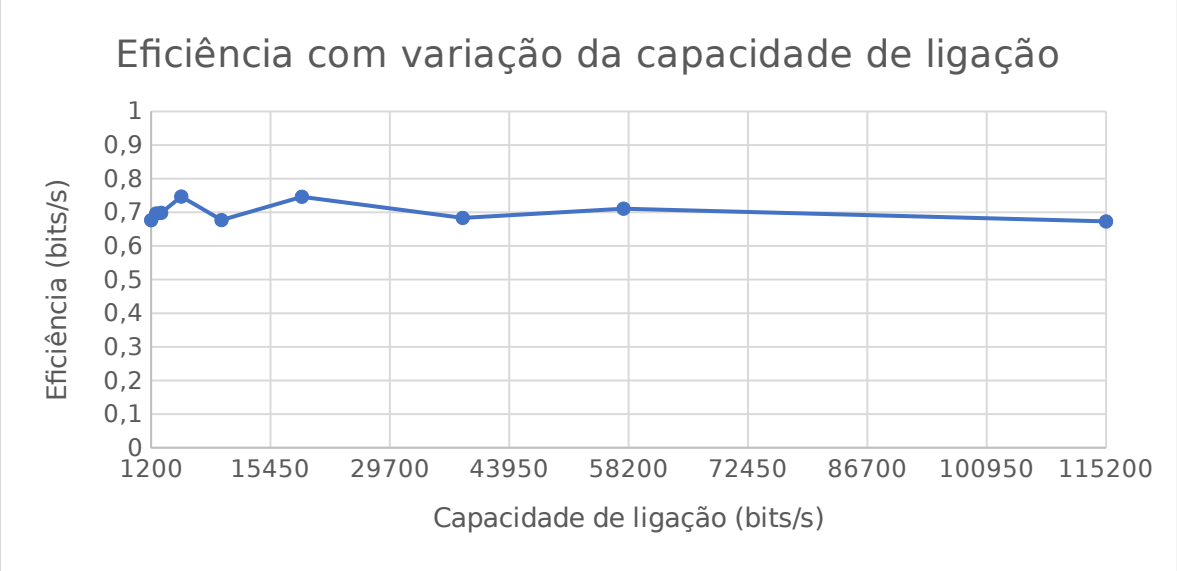
A eficiência diminui quanto maior for a probabilidade de erro dado que a quantidade de tramas a reenviar aumenta proporcionalmente, em média. No entanto, existe a mesma probabilidade de cada trama reenviada ter erro, o que justifica os valores extremamente baixos de eficiência.

Variação de T_prop (atrasos de propagação simulados): [28]



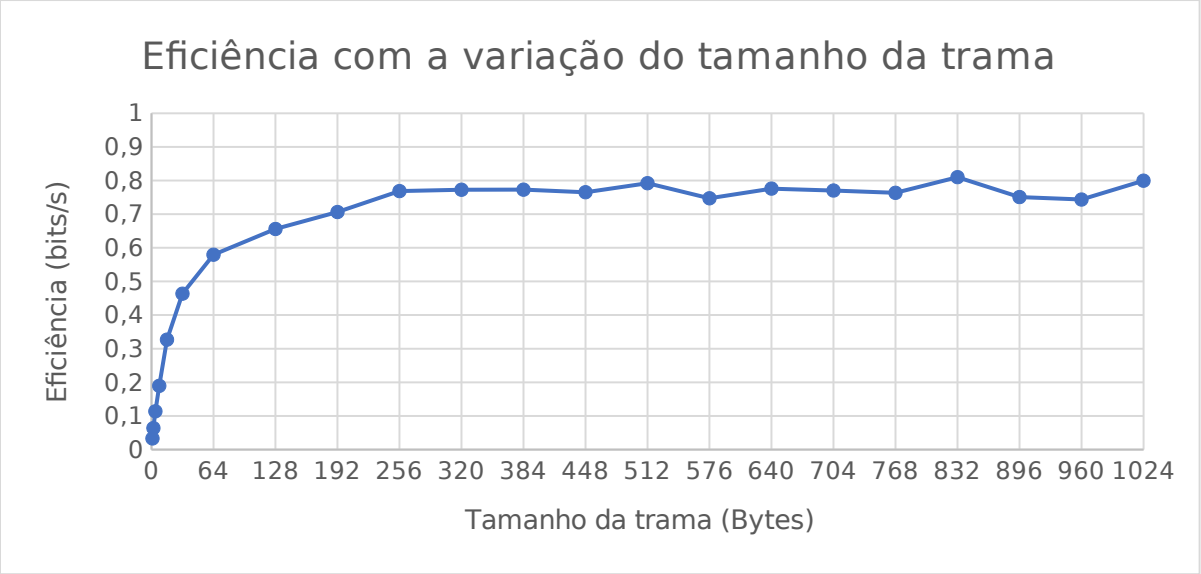
Havendo atrasos na propagação, o número efetivo de bits lidos por segundo diminui dado que a percentagem de tempo em que o protocolo está a processar dados e enviar os mesmos diminui uma vez que passa parte do tempo à espera de receber informação, sem a receber na realidade.

Variação da capacidade de ligação: [29]



Com a observação do gráfico pode-se concluir que a capacidade de ligação não afeta substancialmente a eficiência do protocolo de ligação de dados.

Variação do tamanho da trama: [30]



Como seria de esperar a eficiência do protocolo aumenta com o aumento do tamanho da trama. A partir de 256 bytes nota-se um estagnar da eficiência porque os cabeçalhos passam a representar uma pequena percentagem do que é realmente escrito. Para exemplificar, quando a trama tiver tamanho 1, são escritos 11 bytes de cada vez, 10 dos quais são cabeçalhos. Nota-se também umas variações mínimas à medida que se aumenta a trama pois apesar do peso do cabeçalho diminuir, sempre que houver um erro, a quantidade de bytes a escrever aumenta. Assim, de maneira a maximizar a eficiência o tamanho da trama deverá ser um meio termo entre o menor peso dos cabeçalhos possível e a menor quantidade de bytes a escrever em caso de erro para cada situação (tamanho da trama).

O protocolo ARQ usado é o Stop and Wait. Do lado do transmissor, é enviada a trama e espera-se a resposta de confirmação. Do lado do recetor, analisa-se a trama recebida e envia-se uma resposta consoante a validade dos dados recebidos. Caso o transmissor não receba a confirmação num espaço de tempo estipulado, a trama é de novo enviada. Este protocolo é caracterizado pela sua simplicidade apesar de trazer várias limitações associadas à baixa eficiência.

Conclusões

O trabalho realizado permitiu alargar os conhecimentos sobre a gestão da comunicação entre computadores, os vários parâmetros que influenciam a velocidade e a fiabilidade da transmissão, a gestão dos erros e formas de manter a integridade da informação recebida.

Permitiu também conhecer a base para os protocolos de ligação e de aplicação usados para qualquer acesso a recursos na internet, tal como a relação entre a velocidade real e a influência de valores como os atrasos de propagação e a quantidade de erros ocorridos que juntos definem a verdadeira velocidade de qualquer ligação.

Foi também necessário melhorar os conhecimentos em na linguagem de programação C nos campos de gestão de memória, escrita em ficheiros e gestão de apontadores de forma a gerar código eficiente e que introduzisse o menor *overhead* possível.

Anexos

```
netedu@linus25:~/RCOM$ time ./build/app 0 img/emptyFile 0
FileName: img/emptyFile
FileSize: 0

real    0m0.015s
user    0m0.000s
sys     0m0.000s
```

Imagem 1: tempo de transferência de um ficheiro vazio

```
netedu@linus25:~/RCOM$ time ./build/app 0 img/pinguim.gif 0
FileName: img/pinguim.gif
FileSize: 10968

real    0m1.021s
user    0m0.000s
sys     0m0.000s
```

Imagem 2: Tempo de transferência de um ficheiro com 10960B

```
netedu@linus25:~/RCOM$ time ./build/app 0 img/monte.png 0
FileName: img/monte.png
FileSize: 504466

real    0m46.262s
user    0m0.000s
sys     0m0.092s
```

```
netedu@linus21:~/RCOM$ time ./build/app 0 img/video.webm 0
FileName: img/video.webm
FileSize: 8832677
Closing Connectin

real    13m2.175s
user    0m0.216s
sys     0m0.384s
```

Imagem 4: Tempo de Transferência de ficheiro com 8832677B

```
/**
 * Function used to establish the connection between sender and receiver
 * @param porta value of the port to open
 * @param flag contains the value in which llopen should operate (TRANSMITTER / RECEIVER)
 * @return index of the linkLayer to use, -1 in case of error
 */
int llopen(int porta,deviceType flag);
```

Imagem 5: Cabeçalho da função llopen()

```
if (flag == TRANSMITTER)
{
    setupLinkLayer(&linkNumber[linkLayerNumber], porta, BAUDRATE, 0, 3, 3);
    linkNumber[linkLayerNumber].fd = open(linkNumber[linkLayerNumber].port, O_RDWR | O_NOCTTY);
    if (setTermIO(&newtio, &oldtio, &linkNumber[linkLayerNumber], 1, 0))
        return -1;

    if (send_receive(&linkNumber[linkLayerNumber], UA, SET))
    {
        return -1;
    }
}
```

Imagem 6: Bloco do Transmissor, enviar SET e esperar UA

```
else if (flag == RECEIVER)
{
   setupLinkLayer(&linkNumber[linkLayerNumber], porta, BAUDRATE, 0, 3, 3);
    linkNumber[linkLayerNumber].fd = open(linkNumber[linkLayerNumber].port, O_RDWR | O_NOCTTY);
    if (setTermIO(&newtio, &oldtio, &linkNumber[linkLayerNumber], 1, 0))
        return -1;

    char set[5];
    setHeader(FLAG, ADDRESS, UA, set);
    while (receive(&linkNumber[linkLayerNumber], SET))
    {
    }
    write(linkNumber[linkLayerNumber].fd, set, 5);
}
```

Imagem 7: Bloco do Recetor, esperar o SET e enviar UA

```
/**
 * Function used to create and write a frame to the port
 * @param fd index of the linkLayer to use
 * @param buffer array of chars to proccess
 * @param length size of buffer
 * @return number of chars written, -1 in case of error
 */
int llwrite(int fd, unsigned char * buffer, int length);
```

Imagem 8: Cabeçalho da função llwrite()

```
uint8_t *packet = malloc(length + 1);
uint8_t *stuffedPacket = malloc((length + 1) * 2 + 5);
int u = buffer[0];
conta = 1;
flag = 0;
(void)signal(SIGALRM, atende);
```

Imagem 9: Subscrever alarme e declarar variáveis em llwrite()

```
write(linkNumber[fd].fd, stuffedPacket, length + 5);
alarm(linkNumber[fd].timeout);
unsigned char answer[5];
while (conta <= linkNumber[fd].numTransmissions)
{
    unsigned char out[2] = {RR, REJ};
    int r = receive2(&linkNumber[fd], out, 2);
```

Imagem 10: Escrever trama e esperar resposta(RR ou REJ)

```
/**
 * Function used to read and process data from the port
 * @param fd index of the linkLayer to use
 * @param buffer array to write what is read after it is processed
 * @return number of chars read, -1 in case of error
 */
int llread(int fd, unsigned char *buffer);
```

Imagem 11: Cabeçalho função llread()

```
case 0:
    if (aux == FLAG)
    {
        state = 1;
    }
    break;
case 1:
    if (aux == SNDR_COMMAND)
    {
        A = aux;
        state = 2;
    }
    else
    {
        erro = true;
        if (aux == FLAG)
            state = 1;
        else
            state = 0;
    }
    break;
case 2:
    if (linkNumber[fd].sequenceNumber == aux)
    {
        C = aux;
        changeSeqNumber(&linkNumber[fd].sequenceNumber);
    }
    else
    {
        duplicado = true;
    }
    state = 3;
    break;
case 3:
    if ((A ^ C) == aux)
    {
        state = 4;
    }
    else
    {
        erro = true;
        if (aux == FLAG)
            state = 1;
        else
            state = 0;
    }
    break;
case 4:
    if (aux != 0x7e)
        data[i++] = aux;
    else
    {
        flagReached = true;
        state = 0;
    }
    break;
}
```

Imagem 12: Exemplo da Máquina de estados

```
bcc2 = getBCC2(data, size - 1);

if (bcc2 != data[size - 1])
{
    erro = true;
}
```

Imagem 13: Obter o bcc2 e comparar com o bcc2 recebido

```
setHeader(FLAG, SNDR_COMMAND, REJ, answer);
write(linkNumber[fd].fd, answer, 5);
erro = false;
i = 0;
changeSeqNumber(&linkNumber[fd].sequenceNumber);
flagReached = false;
```

Imagem 14: Envio do REJ

```
/**
 * Function used to close the connection between sender and receiver
 * @param ll index of the linkLayer to use
 * @return 0 in case of success, -1 otherwise
 */
int llclose(int ll);
```

Imagem 15: Cabeçalho da função llclose()

```
if (global_flag == TRANSMITTER)
{
    int a = send_receive(&linkNumber[linkLayerNumber], DISC, DISC);
    char set[5];
    setHeader(FLAG, ADDRESS, UA, set);
    write(linkNumber[linkLayerNumber].fd, set, 5);
}
```

Imagem 16: Bloco do transmissor em llclose(), escrever DISC, esperar DISC e enviar UA

```
else if (global_flag == RECEIVER)
{
    send_receive(&linkNumber[linkLayerNumber], UA, DISC);
}
```

Imagem 17: Bloco do recetor em llclose(), leitura de DISC e envio de DISC

```
else if (flag && conta <= linkNumber[fd].numTransmissions)
{
    flag = 0;
    //printf("Flag %d %d\n", flag, conta);
    write(linkNumber[fd].fd, stuffedPacket, length + 5);
    alarm(linkNumber[fd].timeout);
    printf("Waiting for answer\n");
}
```

Imagem 18: Exemplo de utilização do alarme

```
#define TRAMA_SIZE 256
#define BAUDRATE B115200
//error probability in percentage
#define ERROR_PROB 0
//delay in microseconds
#define PROPAGATION_DELAY 0
```

Imagem 19: Variáveis de configuração

```
/**
 * Function used to read a file, create all packets necessary and pass them to llwrite() one by one
 * @param linkLayerNumber Value to set baudrate
 * @param file pointer to a char array that contains the name of the target file
 * @return -1 if there is any error, 0 if it is successfull
 */
int sendFile(int linkLayerNumber, char *file);
```

Imagem 20: Cabeçalho da função sendFile()

```
if (llwrite(linkLayerNumber, CTRLPacket, packetSize) == -1)
{
    return -1;
}
```
Imagem 21: Chamada a llwrite() com escrita de um control packet que simboliza o inicio da transmissão

```
for (int sequenceNumber = 0; sequenceNumber < localSize; sequenceNumber++)
{
    readSize = fread(&dataPack[4], 1, trama_size, fd);
    dataPacket(dataPack, sequenceNumber % 255, readSize);
    if (llwrite(linkLayerNumber, dataPack, readSize + 4) == -1)
        return -1;
}
```
Imagem 22: Ciclo de leitura de ficheiro segmentado e escrita no terminal

```
controlPacket(CTRL_END, CTRLPacket, &size, file);

if (llwrite(linkLayerNumber, CTRLPacket, packetSize) == -1)
{
    return -1;
}
```
Imagem 23: Chamada a llwrite() com control packet que simboliza o fim da transmissão

```
/**
 * Function used to create a file with what it receives from llread()
 * @param linkLayerNumber Value to set baudrate
 * @return -1 if there is any error, 0 if it is successfull
 */
int receiveFile(int linkLayerNumber);
```
Imagem 24: Cabeçalho da função receiveFile()

```
else if (packetType == CONTROL)
{
    totalSize = *packet.c.fileSize;
    fd = open(packet.c.fileName, O_RDWR | O_NOCTTY | O_CREAT, 0777);
}
```

Imagem 25: Criação/abertura de ficheiro pelo recetor para guardar a informação recebida

```
else if (packetType == DATA)
{
    acumSize += (dataSize - 4);
    if (acumSize / totalSize * 100 > lastP)
    {
        unsigned long curTime = time(NULL);
        float rest = (totalSize - acumSize) * (curTime - iniTime) / acumSize;
        unsigned long acumTime = curTime - iniTime;
        printf("Tempo Restante: %fs\n", rest);
        printf("Tempo Decorrido: %lds\n", acumTime);
        printf("Percentagem: %f%%\n\n", lastP);
 fflush(stdout);
        lastP = (int)(acumSize / totalSize * 100) + 1;
    }
    dataPacket_s *dataPacket = &(packet.d);
    write(fd, dataPacket->data, dataPacket->dataSize);
}
```

Imagem 26: Escrita de informação no ficheiro criado

FER	Eficiência	Tempo
0	0,76936	2,97
5	0,72794	3,139
10	0,65342	3,497
15	0,58321	3,918
20	0,58187	3,927
25	0,57297	3,988
30	0,51767	4,414
35	0,49161	4,648
40	0,46208	4,945
45	0,3701	6,174
50	0,34217	6,678
55	0,3137	7,284
60	0,28459	8,029
65	0,277	8,249
70	0,18468	12,373
75	0,1721	13,277
80	0,14471	15,79
85	0,09492	24,074
90	0,06432	35,527
95	0,02962	77,138

Imagem 27: Valores de tempo e eficiência usados no gráfico de variação do FER,
Atraso = 0s;
baudrate = 38400 bits/s;
trama size = 512B

Atraso	Eficiência	Tempo
0	0,76873	3,344
0,05	0,67898	3,786
0,1	0,5421	4,742
0,15	0,36411	7,060
0,2	0,27824	9,239
0,25	0,21571	11,917
0,3	0,18145	14,167
0,35	0,15584	16,495
0,4	0,13877	18,525
0,45	0,12545	20,492
0,5	0,11199	22,955

Imagem 28: Valores de tempo e eficiência usados no gráfico de variação do atraso de propagação,
% Erro = 10%;
baudrate = 38400 bits/s;
trama size = 512B

C	Eficiência	Tempo
1200	0,67664	108,063
1800	0,6964	69,998
2400	0,6984	52,348
4800	0,74685	24,476
9600	0,67689	13,503
19200	0,74624	6,124
38400	0,68331	3,344
57600	0,71084	2,143
115200	0,67285	1,132

Imagem 29: Valores de tempo e eficiência usados no gráfico de variação do baudrate,
% Erro = 10%;
atraso = 0s;
trama size = 512B

Tamanho	Eficiência	Tempo
1	0,03305	77,787
2	0,06383	40,274
4	0,11372	22,604
8	0,18946	13,568
16	0,32689	7,864
32	0,46359	5,545
64	0,57949	4,436
128	0,65594	3,919
192	0,70641	3,639
256	0,76873	3,344
320	0,77266	3,327
384	0,77289	3,326
448	0,76529	3,359
512	0,79218	3,245
576	0,74727	3,440
640	0,77615	3,312
704	0,77034	3,337
768	0,76348	3,367
832	0,81016	3,173
896	0,75099	3,423
960	0,7436	3,457
1024	0,79957	3,215

Imagem 30: Valores de tempo e eficiência usados no gráfico de variação do tamanho da trama,
% Erro = 10%;
atraso = 0s;
baudrate = 38400 bits/s