Redes de Computadores
1º Projeto

Lara Bastos up202108740
Lia Sobral up202108741

28 de Outubro de 2023

Índice

Sumário

O primeiro projeto da Unidade Curricular de Redes de Computadores consistiu no desenvolvimento de uma camada de ligação de dados, bem como uma camada de aplicação para seu teste.

Após a conclusão do projeto, obtemos um protocolo de ligação de dados via porta série eficiente e que vai de encontro com todos os objetivos pedidos, que permita uma comunicação fiável e com camadas bem definidas e independentes.

Introdução

O objetivo principal do projeto foi a aplicação dos conteúdos lecionados nas aulas teóricas de forma a implementar um protocolo de ligação de dados para transferir ficheiros.
Este relatório, explora assim tópicos desde a arquitetura do protocolo até à sua eficiência, estando divididos nas seguintes secções:

- Arquitetura:
  - Blocos funcionais e Interfaces
- Estrutura do código:
  - APIs, principais estruturas de dados, principais funções e a 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, se possível.
- Eficiência do protocolo de ligação de dados
  - Caraterização estatística da eficiência do protocolo, efetuada recorrendo a medidas sobre o código desenvolvido. A caraterização teórica de um protocolo Stop&Wait, que deverá ser empregue como termo de comparação, encontra-se descrita nos slides de Ligação Lógica das aulas teóricas.
- Conclusões
  - Síntese da informação apresentada nas secções anteriores;
  - Reflexão sobre os objetivos de aprendizagem alcançados.

Arquitetura

Blocos Funcionais

O projeto foi desenvolvido em duas camadas independentes e bem definidas, a camada de aplicação (*ApplicationLayer)* e a camada de ligação de dados (*DataLinkLayer)*.

1. Camada de Ligação de Dados

Esta camada encontra-se implementada nos ficheiro *datalinklayer.h* e *datalinklayer.c*. É responsável por iniciar e terminar ligação, bem como o envio das tramas por parte do transmissor e validação e posterior envio de tramas de aceitação ou rejeição por parte do recetor.

2. Camada de Aplicação

A camada da aplicação, por sua vez, recebe input do utilizador, nomeadamente o ficheiro a transferir, o número de transmissões e tempo de *timeout* desejados. De seguida, abre o ficheiro, fraciona-o sequências de bytes e utiliza a API da camada de ligação de dados para o envio e receção de pacotes. Por fim é responsável também por fechar o ficheiro.

Interface

A execução do código é feita em dois terminais, sendo um o transmissor e o outro o recetor. O utilizador pode também escolher que ficheiro transferir e em que ficheiro escrever. Para este efeito, após compilar, devem ser utilizados os dois comandos seguintes, devendo iniciar-se pelo recetor e de seguida o transmissor:

./bin/main /dev/ttyS11 rx penguin-received.gif
./bin/main /dev/ttyS10 tx penguin.gif

Estrutura do Código

Camada da Aplicação

Funções principais

- applicationLayer: Função principal que divide o ficheiro em pacotes, construindo os seus header e envia-os 1 a 1 para a camada de ligação de dados, utilizando a sua API. Do lado do recetor, recebe os pacotes 1 a 1 e vai escrevendo-os no ficheiro final.
- createStartEndPacket: Constrói os pacotes de controlo (de início e de fim).
- fileData: Vai buscar os bytes do ficheiro.
- removeHeader: Retira o header do pacote de dados recebidos, para obter os bytes que devem ser escritos no ficheiro.

Camada de Ligação de Dados

Funções principais

- llopen: Começa pelo estabelecimento da ligação. De seguida, caso se tratar do transmissor, envia um SET e espera por receber um UA. Se, por sua vez, for o recetor aguarda por receber um SET para depois enviar uma UA.
- llwrite: Constrói a trama a partir do pacote recebido e envia-a para o recetor. De seguida, espera por receber uma trama de controlo. Caso seja de rejeição volta a enviar a trama.
- llread: Aguarda a receção da trama, verificando todos os seus campos. Envia uma trama de aceitação ou rejeição de acordo se receber uma trama com os campos corretos ou incorretos respetivamente. Caso receba trama repetidas envia uma trama de aceitação mas não armazena de novo a trama.
- llclose: Envia um DISC, espera pela receção de um DISC por parte do recetor, enviando de seguida um UA. Por fim, fecha a porta séria terminando a ligação e repondo todos os campos inicias.
- receiver_state_machine: Máquina de estados que aguarda por receber um SET e valida-o, para de seguida enviar um UA.
- transmiter_state_machine: Máquina de estados que aguarda por receber um UA e valida-o.
- control_frame_state_machine: Máquina de estados que espera a receção de uma trama de controlo por parte do recetor.
- alarmHandler: A cada *timeout* do alarme, esta função é chamada e aumenta o contador.
- set_serial_port: Estabelece a ligação entre as portas séries, armazenado as configurações antigas para poderem ser repostas no final da execução.

Estrutura de Dados

- LinkLayerRole: Identifica se computador que está a executar o código é o recetor ou o transmissor.
- LinkLayer: Identifica os vários parâmetros definidos pelo utilizador (*timeout, baudRate, nRetransmissions* etc.)
- LinkLayerStateMachine: Estados possíveis na totalidade das máquinas de estado implementadas.

Casos de uso principais

Caso o transmissor seja iniciado em primeiro lugar, irá transmitir a mensagem de início de envio a cada *timeout* segundos, *nRetransmission*s vezes (por defeito 4 segundos e 3 vezes). Caso o recetor não seja ligado entretanto, a conexão é fechada.

Após ser iniciado o **recetor** vai ter a seguinte sequências de chamada de funções:

1. applicationLayer()
2. llopen()
3. receiver_state_machine()
4. set_serial_port()
5. llread()
6. removeHeader()

O **transmissor**, por sua vez, chama as seguintes funções:

1. applicationLayer()
2. llopen()
3. createStartEndPackett()
4. fileData()
5. transmiter_state_machine()
6. set_serial_port()
7. llwrite()
8. control_frame_state_machine()
9. createStartEndPacket()
10. llclose()

Protocolo de ligação de dados

A camada de ligação de dados é a responsável por enviar e receber dados pela porta série, permitindo assim a comunicação entre os dois computadores. Através da implementação de um alarme, na eventualidade de ocorrer uma falha na transmissão que resulta na não receção de alguma resposta, a trama volta a ser enviada. Esta camada é composta por 4 funções principais: **llopen**, **llwrite**, **llread** e **llclose**.

A função **llopen**, começa por estabelecer a ligação com a porta série e guardar as configurações antigas. De seguida, do lado do transmissor, é enviado um SET que indica que vai ser iniciado o envio dos dados do ficheiro. Fica assim à espera de receber a resposta por parte do recetor para garantir que a ligação foi corretamente efetuada. O recetor, ao receber esta trama de supervisão, envia o UA.

A função **llwrite** é chamada de seguida. Esta apenas é usada pelo emissor e é responsável pelo envio das tramas. Começa por receber um pacote da camada de aplicação, transformá-lo numa trama, criando um *header* e BCC2, e enviá-la para o recetor. De forma a garantir que o recetor não vai terminar de ler a trama mais cedo, antes da transformação do pacote numa trama, realizamos o *stuffing*. Qualquer byte que seja idêntico à *flag* ou ao ESC, é substituído pelo ESC seguido do valor do byte^0x20. Após o envio, o transmissor aguarda por uma resposta. Caso esta seja negativa, a trama volta a ser enviada *nRetransmissions* vezes.

Por sua vez, a função **llread** realiza a leitura dos dados recebidos na porta sérias. Esta apenas é usada pelo recetor. Começa por realizar o *destuffing*, ou seja, reverter o mecanismo explicado anteriormente. De seguida, ocorre a validação dos *header* e BCCs, de forma a verificar que a trama foi recebida corretamente e garantir a integridade da informação. Se estes campos não estiverem corretos é enviada uma resposta de rejeição. Se a trama já tiver sido recebida pelo recetor, é enviada uma resposta de aceitação mas a trama não é armazenada.

Por fim, quando todo o ficheiro tiver sido enviado ou quando já tiver enviado a trama nRetransmissions vezes sem resposta, o transmissor chama **llclose**. Nesta, envia uma trama DISC e espera até receber uma de volta para enviar um UA, terminando assim a ligação.

Protocolo da Aplicação

A camada da aplicação, por sua vez, é responsável por receber e escrever informação no ficheiro. Para este fim, utiliza as funções da API fornecida pela LinkLayer para receber e escrever informação no ficheiro. Uma vez que as camadas devem ser independentes, esta apenas é responsável por abrir, ler e escrever dados no ficheiro.

Começa por chamar o **llopen** para estabelecer a ligação. De seguida, o transmissor abre o ficheiro, determina o seu tamanho e armazena toda a sua informação. O primeiro pacote a ser enviado é de controlo e indica o início da transmissão. O ficheiro é fragmentado em segmentos de tamanho MAX_PAYLOAD (valor arbitrário de 1000 mas pode ser alterado), transformado num pacote de dados e enviado através da função **llwrite**.

O recetor, através da função **llread**, vai recebendo pacote a pacote, extrai a informação necessária e escreve os bytes no ficheiro.

Após todo o ficheiro ser enviado com sucesso, ou ocorrer algum erro na transmissão que provoque a função **llwrite** retornar -1, o transmissor invoca o **llclose**, que irá desligar a ligação e repor as configurações originais da porta série.

Validação

Durante o desenvolvimento do projeto e com o objetivo de assegurar a implementação precisa do protocolo que foi criado, foram realizados diversos testes, abrangendo uma variedade de cenários. Os testes realizados incluíram:

- Transferência de ficheiros com tamanhos diferentes.
- Transferência de pacotes de dados com tamanhos variados.
- Simulação de rejeição de tramas com a introdução de erros.
- Transferência de ficheiros com diferentes *baudrates*.
- Interrupção da Porta Série.
- Introdução de ruído na porta série
- Conexão do transmissor antes ou após a conexão do recetor.
- Conexão do transmissor sem conexão do recetor.

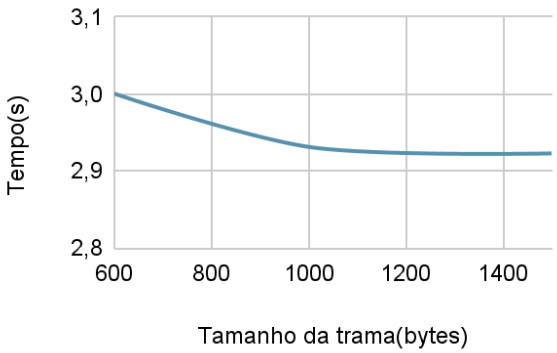
Graças à nossa abordagem abrangente e aos testes realizados, conseguimos assegurar a transferência bem-sucedida dos ficheiros, mesmo quando ocorreram erros no Header, no BCC2 e em casos de tramas repetidas. Isso traduziu-se em dois ficheiros diferentes com bytes idênticos no final do processo.

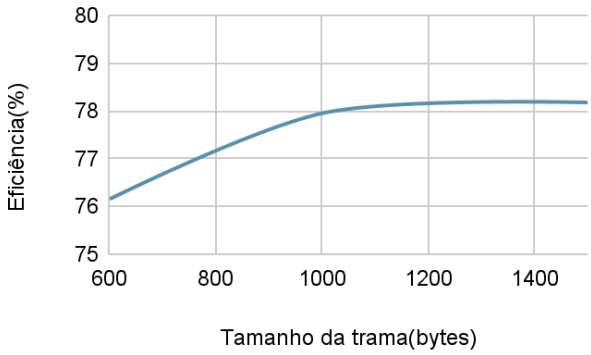
Eficiência do protocolo de ligação de dados

Variação do tamanho da trama

Com um ficheiro de 10 968 bytes e um *baudrate* fixo de 38400 bits/s, observamos os seguintes resultados ao variar o tamanho da trama de informação:

Tamanho da trama (bytes)	Tempo (s)	T frame (Kbits/s)	Eficiência (%)
600	3.000448	29.243633	76.155
1000	2.93136156	29.932848	77.95
1500	2.92289	30.019584	78.176



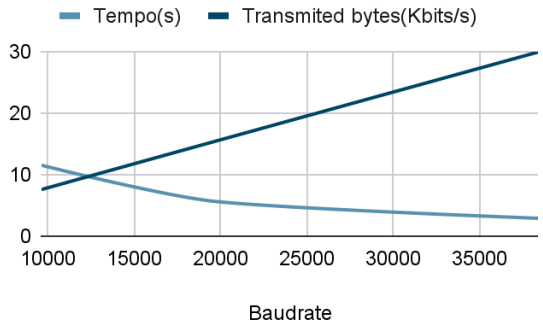


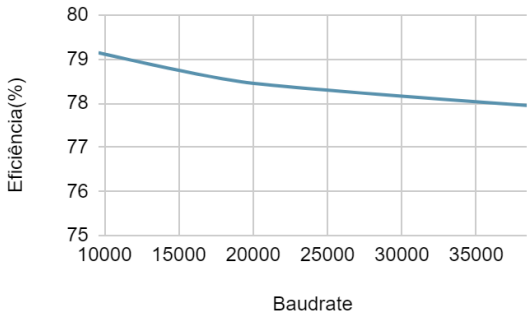
Como podemos observar, o tempo total de transferência do ficheiro é inversamente proporcional ao tamanho de cada trama, visto que, com a diminuição do tamanho da trama, um maior número de transmissões terá de ser efetuado. Por sua vez, a eficiência apresenta valores mais elevados quando o valor do tamanho da trama aumenta.

Variação do *baudrate*

Com um ficheiro de 10 968 bytes e uma trama de informação de tamanho fixo de 1000 bytes, observamos os seguintes resultados ao variar o *baudrate*:

Baudrate	Tempo (s)	T frame (Kbits/s)	Eficiência (%)
9600	11,5482836	7,598012246	79,146
19200	5,82254612	15,069696	78,488
38400	2,93136156	29,932848	77,95



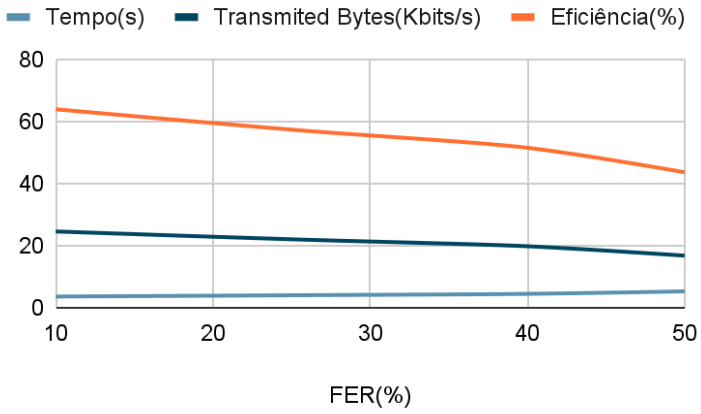


Conforme observado, existe uma relação inversa entre o tempo total de transferência do ficheiro e a velocidade de propagação no cabo (baudrate). A eficiência do protocolo diminui progressivamente à medida que o *baudrate* aumenta, uma vez que, embora a velocidade de propagação das tramas aumente, o tempo necessário para transmitir essa mesma informação também se estende.

Variação da taxa de erros

Com um ficheiro de 10968 bytes, um baudrate de 38400 bits/s e tamanho da trama de 1500 bytes, ao variar o Frame Error Ratio, observamos os seguintes resultados:

FER (%)	Tempo (s)	T frame (Kbits/s)	Eficiência (%)
10	3.5762611	24.5351213	63.89
25	3.98665173	22.009	57.32
40	4.43841512	19.769	51.48
50	5.2419419	16.7388349	43.6



Cada vez que uma trama apresenta um erro, o protocolo utilizado provoca a retransmissão da mesma, o que aumenta o tempo de transferência de cada pacote de dados e por sua vez, todo o

tempo global de todo o processo. Consequentemente, a eficiência é inversamente proporcional à taxa de erros.

Conclusão

O trabalho consistiu na implementação de duas camadas de um protocolo de ligação de dados. Ambas as camadas foram desenvolvidas de forma a serem 100% independente para que nenhuma tivesse a necessidade de compreender a implementação da outra. Desta forma compreendemos melhor como funciona a comunicação entre camadas e a importância de estarem bem definidas.

Conseguimos também consolidar os conhecimentos lecionados na UC acerca de tramas e pacotes e como ocorre a divisão de um ficheiro nestes. Aprofundamos ainda conceitos como mecanismos de stuffing e de deteção de erros, nomeadamente não receção das tramas e envio de tramas repetidas.

Por fim, na fase final, conseguimos compreender como é que a variação do *baudrate*, do tamanho da trama e do FER influenciam a eficiência do projeto.

Uma vez que atingimos todos os objetivos esperados, e o nosso projeto lidou eficazmente com todos os testes e perturbações realizadas na apresentação, acreditamos que a realização deste foi bastante bem sucedida.