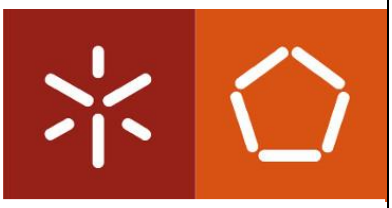

Universidade do Minho

Escola de Engenharia

Relatório Final

Emulação e Simulação de Redes de Telecomunicações

Ano Letivo 2021/2022

Docentes:

Maria João Nicolau

António Costa

José Afonso

Paulo Araújo

Realizado por:

Francisco Neto, a90297

Ricardo Loureiro, a89401

Rui Barbosa, a89370

Vitor Sá, a88606

Índice

Índice de Figuras

Índice de Tabelas

Introdução

No âmbito da unidade curricular de Emulação e Simulação de Redes de Telecomunicações foi-nos proposto o desenvolvimento de uma aplicação que consiga comunicar numa rede local sem fios. Utilizaremos este tipo de comunicação para suportar o desenvolvimento de uma aplicação do tipo "*chating*" de modo a permitir a partilha de ficheiros, imagens e também uma conversação textual em tempo real entre dois computadores pessoais (PC – *Personal Computer*).

Durante o desenvolvimento do projeto iremos aprofundar a camada 1, 2 e 7 do modelo OSI (*Open Systems Interconnection*) que corresponde à camada física, de ligação e de aplicação, respetivamente.

Para a 1º etapa do projeto é pretendido estabelecer uma ligação sem fios entre dois Arduínos, este tipo de comunicação pode ser feito de diversas formas, entre eles RF (radiofrequência), Wi-Fi, Bluetooth, entre muitas outras tecnologias.

Em relação à 2º fase, o principal objetivo é conhecer as características de comunicação entre cada transceiver e o módulo Arduíno, efetuar as ligações necessárias entre os componentes de Arduíno utilizados, conhecer e compreender o impacto dos diferentes parâmetros configuráveis do *transceiver*, desenvolver o código para configurar e transmitir/receber os dados dos respetivos *transceivers* de RF e, por fim, executar os testes experimentais para avaliação do desempenho do sistema, tais como, *delay*, *round-trip time* (RTT), débito máximo e alcance de transmissão (sem erros/perda de informação).

Quanto à fase 3 do projeto, é necessário perceber a noção de trama/pacote, cabeçalho, *payload* e cauda, perceber como representar corretamente a estrutura de um pacote, definir o protocolo de comunicação, especificando e conhecendo as regras de comunicação, tramas e primitivas de serviço a oferecer à camada superior. Devemos ainda definir os tipos de trama e os campos que estas necessitam de contar para proporcionar as funcionalidades desejadas no âmbito da 2º camada do projeto para uma transmissão fiável de dados ponto-a-ponto, com ênfase na deteção e correção de erros, disponibilizar as primitivas às camadas superiores através de API's e, por fim efetuar a transferência fiável de um fluxo de dados entre os dois Arduínos com a ligação RF sem fios.

Relativamente à 4º e última fase do projeto, os objetivos principais passam por entender o último nível do protocolo (camada de aplicação), criar um programa de

conversação e transferência que seja capaz de utilizar o protocolo desenvolvido e, por fim, perceber como interligar as camadas segundo o princípio da independência de camadas. Por exemplo, a camada 2 não deve ter conhecimento de que a informação que está a transportar pertence (ou não) a um ficheiro. No entanto, a camada 7 não deve ter conhecimento sobre as tramas de controlo de erros trocadas no interior da camada 2.

Em suma, para conseguirmos realizar com sucesso todos os objetivos do projeto, é necessário aplicar os conhecimentos estudados em cadeiras lecionadas anteriormente ou atualmente, como é o caso de Redes de Computadores I e Sistemas Operativos, entre outras.

Fundamentos

Modelo OSI

O Modelo OSI (*Open Systems Interconnection*) foi criado em 1984 pela ISO (*International Organization for Standardization*), tem como principal objetivo ser um modelo *standard* para protocolos de comunicação entre diversos tipos de sistema, independentemente da sua constituição, tecnologia ou fabricante, garantindo assim a comunicação *end-to-end* entre eles.

A arquitetura deste modelo divide as redes de computadores em sete camadas para obter camadas de abstração. Cada camada tem funções especificas, verificando se uma dependência de subserviência entre elas, visto que cada uma fornece serviços e funções às camadas superiores, ficando sempre dependente da camada anterior.

As 7 camadas e do Modelo OSI são:

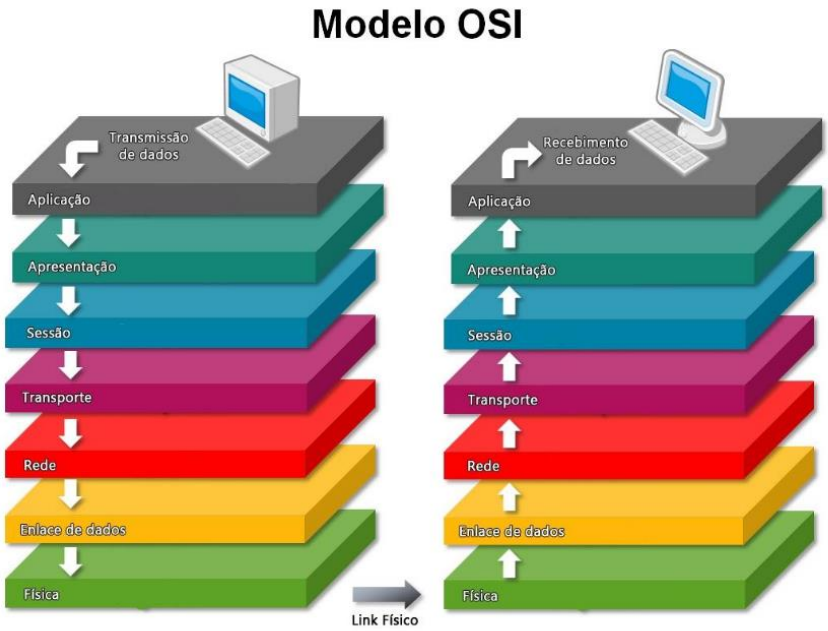

Figura 1 - Modelo OSI

Camada 1 - Camada Física - A camada física é a camada de mais baixo nível no modelo OSI, diz respeito a transmissão e receção de uma sequência de bits provenientes de um meio físico. Esta define especificações elétricas e físicas dos dispositivos. Ela descreve as interfaces elétricas, óticas, mecânicas e funcionais de um meio físico, transporta sinais para todas as camadas superiores sendo também responsável por definir se a transmissão pode ser ou não realizada nos dois sentidos simultaneamente.

Camada 2 - Camada de Ligação (Enlace de Dados) - É nesta camada que se realiza a transferência de informação e dados através de um canal de transmissão, sendo esta também responsável por controlar o fluxo (receção, delimitação e transmissão de quadros), sendo que é nela que se estabelece um protocolo de comunicação entre sistemas diretamente conectados fazendo com que seja possível a deteção e, posteriormente, a correção de erros que possam ocorrer ao nível físico.

Camada 3 - Camada de Rede - Esta camada é responsável por facilitar a transferência de dados entre duas redes diferentes, realiza também o encaminhamento e endereçamento da informação. Para que seja possível o endereçamento, esta camada converte os endereços lógicos (IP) em endereços físicos, depois disto a camada realiza o roteamento de pacotes, para que a informação chegue onde tem de chegar e para que o destino final seja o esperado e o correto.

Camada 4 – Camada de Transporte - Esta camada funciona basicamente como uma interface entre as três camadas superiores e os três inferiores visto que esta camada recebe os dados enviados pela camada de sessão (camada 5), divide-os em porções chamadas segmentos que serão transmitidos para a camada de rede. No recetor, a camada de transporte deteta e elimina erros provenientes das camadas anteriores e garante a integridade do pacote de dados a serem transmitidos e se isso não se verificar solicita uma retransmissão. Os protocolos de transporte usados nesta camada são o TCP e o UDP.

Camada 5 – Camada de Sessão - Esta camada tem como função exercer o controlo organizado da informação entre duas aplicações, são estas aplicações que definem como será feita a transmissão de dados decidindo assim quando estas comunicações devem começar, terminar ou reiniciar. Percebendo-se assim que é nesta camada que se denota um controlo do diálogo e de sincronização entre os *hosts*.

Camada 6 – Camada de Apresentação- Esta camada tem como função converter os dados recebidos pela camada superior (Camada Aplicação) num formato que possa ser entendido pelo recetor. A camada de apresentação é responsável pela tradução, criptografia e compactação dos dados a serem transmitidos.

Camada 7 – Camada de Aplicação - Esta é a única camada que interage diretamente com os dados do usuário, já que funciona como uma porta de entrada de rede, dando o acesso aos serviços dessa rede. Os protocolos da camada de aplicação incluem o HTTP, SMTP e FTP.

Arquitetura do Sistema

A arquitetura representa estruturalmente os componentes do sistema, utilizadores e ligações.

Uma vez que o objetivo do projeto será uma aplicação de chat entre 2 computadores pessoais, o tipo de comunicação terá de ser bidirecional uma vez que ambos os computadores podem funcionar ora como emissores, ora como recetores. O hardware chave para a comunicação por radiofrequências será o *transceiver* (RF nRF24L01+) que permitirá enviar ou receber dados entre os dois computadores, funcionando como emissor ou recetor respetivamente. Para comandar os *transceivers* utilizaremos 2 Arduínos (Placa ESP32) que estarão ligados a um dos *transceivers* fisicamente, via cabo e correspondentemente ligados a um dos computadores pessoais, tendo assim o hardware necessário para efetuar a comunicação via radiofrequência entre os dois computadores.

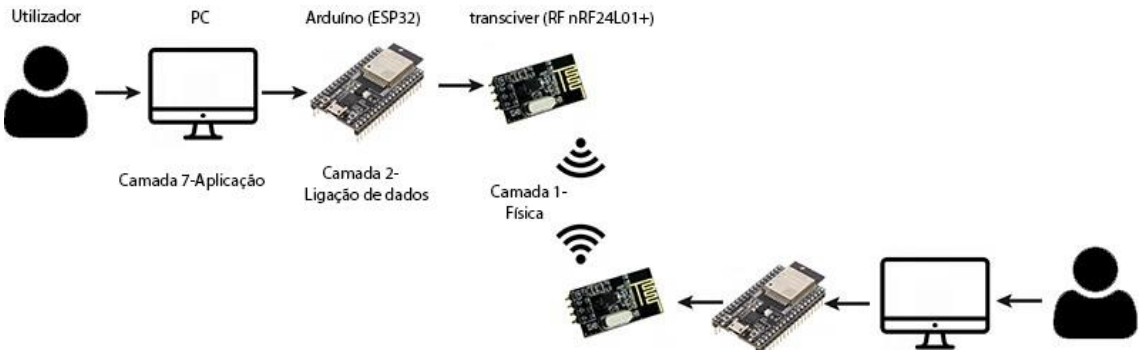

Figura 2 - Arquitetura do Sistema

Transceiver RF (TRF)

O módulo de RF é um pequeno dispositivo eletrónico com capacidade para transmitir e/ou receber sinais de rádio entre dois dispositivos, que são eles a placa ESP32 e as placas *transceiver* (NRF24L01+). Esta comunicação pode ocorrer de duas maneiras, por meio ótico ou por radiofrequência (RF), sendo esta última mais utilizada pois não requer uma linha de visão. A troca de informação no *transceiver* RF incorpora um transmissor e um recetor, podendo ter vários tipos e intervalos, chegando a poder transmitir até 150 metros **[1]**.

A banda de frequência que iremos utilizar no nosso módulo RF está compreendida entre 2,4 e 2,5 GHz. Na imagem a seguir conseguimos visualizar o *transceiver* RF (NRF24L01+) **[2]**.

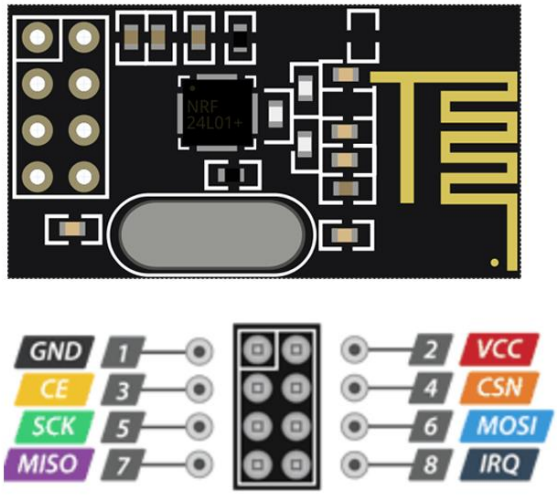

Figura 3 - Pinout TRF
https://diyusthad.com/2020/05/nrf24l01-pinout.html

GND – Terra (0V)	**VCC** - Fonte de alimentação (3,3V)
CE – Ativa o modo RX ou TX	**CSN** - *Chip Select*
SCK – Relógio Serial	**MOSI** - Saída *Master*, Entrada *Slave*
MISO - Entrada *Master*, Saída *Slave*	**IRQ** - Solicitação de interrupção

Tabela 1 - Explicação do Pinout nRF24L01+

Interface SPI

A interface SPI (*Serial Peripheral Interface*) é um protocolo de comunicação/transmissão de dados entre dispositivos periféricos. Neste caso, funcionará como um canal de comunicação utilizado para fazer a ligação entre o módulo TRF e a placa ESP32. Esta interface opera em modo "*full-duplex*", isto significa que os dados são transferidos de uma forma bidirecional e ao mesmo tempo, fazendo com que a velocidade de troca de dados seja mais elevada. Utiliza também uma arquitetura "*master-slave*" que consiste no *Master* começar por enviar um bit na linha MOSI (*Master Output Slave Input*) para o dispositivo *Slave*, e simultaneamente, o dispositivo *Slave*, envia um bit na linha MISO (*Master Input Slave Output*) para o dispositivo Master **[3]**.

As transmissões de dados envolvem deslocamento de registos onde os dados são enviados continuamente e simultaneamente, do *Master* para o *Slave*. Após a troca de bits estar concluída, o *Master* e o *Slave* trocam valores de registo. Se houver necessidade de transmissão de mais dados, os registos são recarregados e o processo é repetido. A transmissão é feita durante um determinado número de ciclos de relógio onde o sinal de *clock* é gerado e controlado pelo *Master*, tal como se observa na figura seguinte **[4]**.

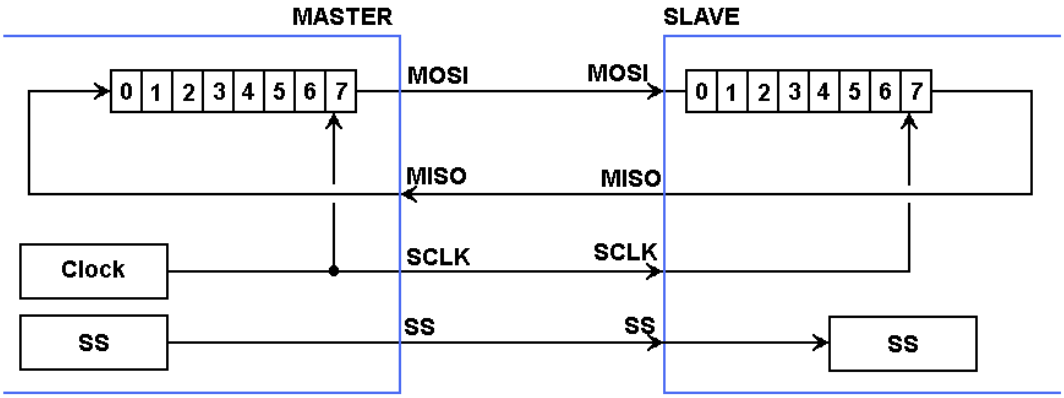

Figura 4- Ligações entre o Master e o Slave
https://www.embarcados.com.br/spi-parte-1/

Controlo de Acesso ao Meio

Uma rede de área local é um conjunto de hardware e software que permite estabelecer comunicações entre computadores. No entanto esta rede irá criar problemas e é necessário um controlo de acesso ao meio (MAC) para fazer com que os ficheiros não tenham erros de transmissão. O termo MAC (*Media Access Control*) pertence à 2º camada do modelo OSI que permite a conexão de diversos computadores numa rede **[5]**.

No entanto, existem protocolos que têm de ser cumpridos, entre eles o CSMA (*Carrier Sense Multiple Access*) e o seu subprotocolo CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*)

Relativamente ao CSMA é um protocolo que emite um sinal antes de enviar algum pacote para saber se está desocupado. Se isso acontecer ocorre a transmissão da trama, senão atrasa a sua transmissão. Por sua vez, as colisões não são totalmente evitadas, elas ocorrem quando dois ou mais *hosts* decidem iniciar a transmissão sem terem informação do canal **[6]**.

Já em relação ao subprotocolo CSMA/CD funciona de forma diferente do CSMA e em vez de enviar um sinal, escuta o canal para saber se há transmissão de alguma trama, porém este continua a escutar enquanto transmite. *Collision Detection* significa que havendo uma colisão, a transmissão é interrompida por completo e o canal é totalmente libertado por um tempo indeterminado antes de voltar a transmitir **[7]**.

Parâmetros a Considerar para Testes

Para a realização de testes iremos ter em consideração vários parâmetros/fatores que podem afetar a transmissão de dados de emissor para recetor. Os parâmetros que iremos ter em conta são:

- ***Round-Trip Time*** é o tempo em que o emissor demora a enviar uma mensagem para o recetor e a receber essa mesma mensagem. Basicamente é a soma do tempo de envio do pacote de dados de emissor para recetor mais o tempo de transmissão desse mesmo pacote de dados do recetor para emissor novamente. Os valores do RTT podem variar dependendo do tamanho do pacote de dados, da velocidade de transmissão e consequentemente do tempo de transmissão que é a relação entre os dois últimos parâmetros referidos.

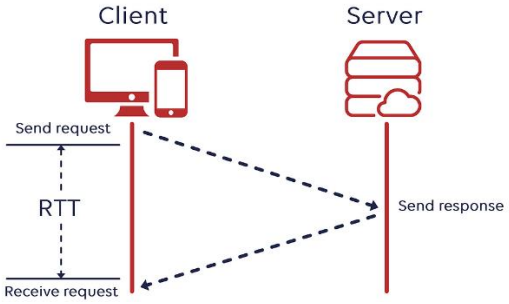

Figura 5- Round Trip Time
https://www.stormit.cloud/post/what-is-round-trip-time-rtt-meaning-calculation

- ***Delay*** é o tempo que demora a mensagem a ser transmitida de emissor para recetor, como neste caso, o tempo de processamento pode ser desprezado podemos considerar que o *delay* é metade do *RTT*.

- **Débito Máximo (*Throughput*)** é uma unidade que define a quantidade de dados transmitidos entre dois ou mais sistemas num intervalo de tempo. Esta unidade normalmente é medida em *bits/kilobits/megabits/gigabits* por segundo dependendo do tamanho dos dados a ser transmitidos **[8]**.

- **Alcance da Transmissão** é uma distância limite para a qual se consegue estabelecer uma conexão estável sem erros e sem perdas de pacotes. Para que exista uma otimização do alcance e do débito máximo é imprescindível uma boa configuração do ESP32 e dos respetivos transístores.

Noção de Trama/Pacote

Para uma rede/conexão ser mais eficiente existe uma necessidade de dividir a informação a ser transmitida em pacotes. Cada pacote carrega a informação que o vai ajudar a chegar corretamente ao seu destino, para isso o pacote será enviado pela melhor rota possível sendo todos os outros pacotes oriundos da mesma informação original irão seguir essa mesma rota.

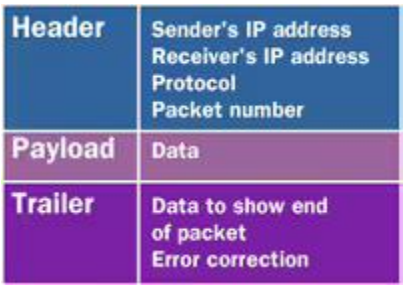

Figura 6- Estrutura da Trama

Um pacote é constituído por três partes:

- **Cabeçalho** (*Header*): Contém informações acerca da data a ser transportada, tal como o tamanho total do pacote, o número identificativo do pacote (no caso de haver uma sequência de pacotes a ser enviados) e por fim o destino e a origem desse mesmo pacote.
- ***Payload***: Representa o corpo do pacote, onde se encontra as informações que realmente se quer transmitir até ao seu destino final.
- **Cauda** (*Tail ou Trailer*): É constituída por alguns bits que irão alertar o receptor se já chegou ao fim do pacote a ser recebido. Pode também conter alguns bits para deteção e correção de erros, o mecanismo mais usado nestas situações é o *Cyclic Redundancy Check*, mais conhecido como CRC. **[9]**

Protocolo de Comunicação

Existem vários protocolos de comunicação para o controlo de erros, no entanto estamos mais acostumados a dois que foram lecionados na UC de Redes de Computadores I, sendo eles o *Stop-and-Wait* e *Sliding Window*. O grupo optou por escolher o protocolo *Stop-and-Wait* e irá ser explicado em detalhe de seguida.

Stop-and-Wait

No mecanismo *Stop-and-Wait*, o emissor envia uma trama de cada vez e espera pela confirmação positiva (ACK). A seguinte trama de dados só é enviada após ter recebido a confirmação.

Para tentar diminuir os erros que costumam ocorrer, pois embora seja enviada uma de cada vez, eles também acontecem, numeram-se as tramas usando 0 e 1. Em certos casos a retransmissão de trama de dados dá origem a pacotes duplicados, sendo por esta razão importante a numeração.

Esta técnica é bastante funcional quando uma mensagem é fragmentada em poucas tramas, mas de grande dimensão. Porém, se o tamanho das tramas for muito grande haverá uma maior probabilidade de erro e uma utilização mais elevada dos recursos, como os buffers e os processadores, fazendo com que o desempenho da ligação tenda a piorar. Caso o tamanho das tramas seja muito pequeno, a eficiência do *Stop-and-Wait* pode diminuir drasticamente, tornando-se inadequado a sua utilização. Na figura abaixo temos um exemplo gráfico de como funciona o *Stop-and-Wait*. **[10]**

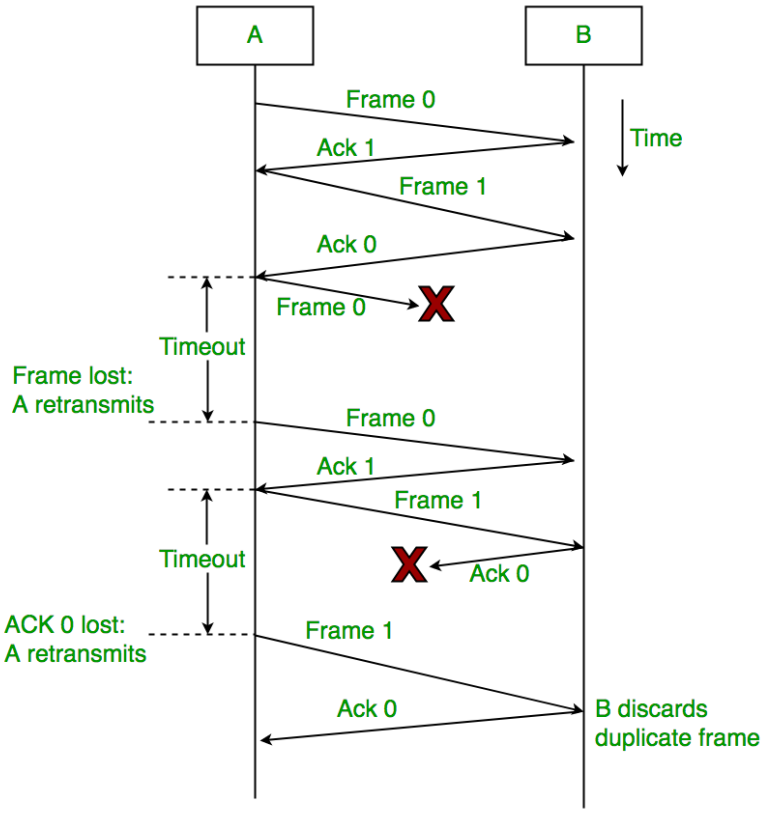

Figura 7 - Mecanismo do Stop-and-Wait

Controlo de erros

Checksum

É uma técnica utilizada para detetar erros, mais predominante na Internet e é denominada como somas de verificação.

Alguns sistemas de rede enviam um *checksum* por cada pacote para facilitar o trabalho do recetor na deteção de erros. Se das somas de verificação resultarem apenas 1's, é sinal de que "não" houve erro na transmissão dos pacotes, caso apareçam 0's verifica-se que na transmissão dos pacotes ocorreram erros. Na imagem seguinte, observa-se um exemplo da soma de verificação do *checksum*. **[11]**

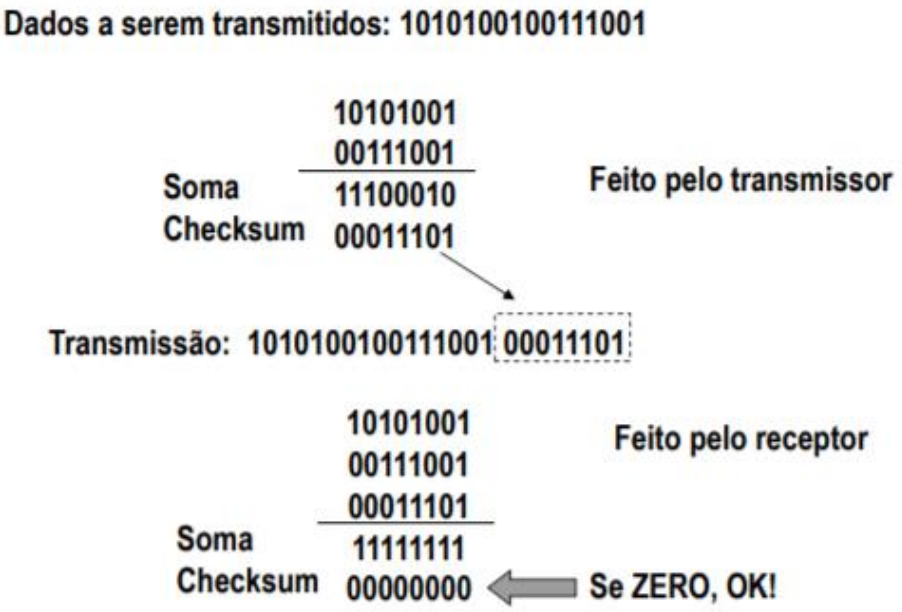

Figura 8 - Método de verificação da Checksum
http://www.inf.ufrgs.br/~asc/redes/pdf/aula06.pdf

*Cyclic Redundancy Check (*CRC*)*

O CRC é um método muito usado para detetar erros na transmissão digital de informação/dados, logo iremos usá-lo para a verificar a existência de erros no nosso protocolo. Sendo um tipo de *Checksum* e tal como ele, o CRC anexa à trama a ser enviada um número fixo de bits. Este método apresenta uma maior complexidade em termos de cálculos, porém aparenta ser mais eficaz na deteção de erros, e apenas pode apresentar debilidades quando o número de erros for maior á capacidade implementada pelo código de deteção. O processo CRC, de um modo geral, é representado através de polinómios de uma variável, com coeficientes binários.**[12]**

CRC-32: $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

100000100110000010001110110110111 (33bits, para dar resto 32)

Figura 9 – Exemplo de um polinómio gerador G(x)
Slides de Redes de Computadores I,Nicolau Pinto,Maria João.

A sequência de bits a proteger (dados que realmente queremos enviar) é divida pelo número de bits fixos definidos pelo polinómio gerador.

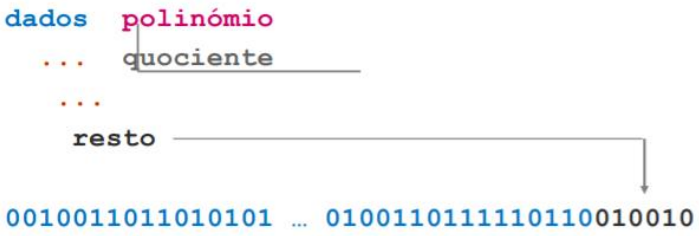

Figura 10 - Método Matemático do CRC
Slides de Redes de Computadores I,Nicolau Pinto,Maria João

Por fim o resto da divisão será anexado à mensagem a ser transmitida, como redundância.

Por exemplo, queremos a enviar a mensagem que contém os seguintes bits: 10110.

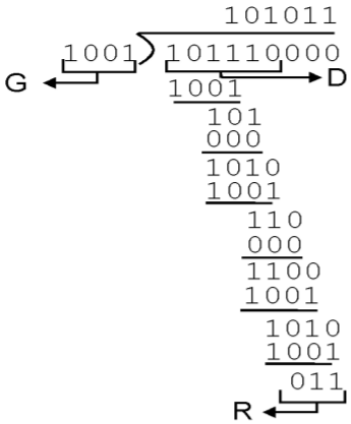

Figura 11 - Exemplo de uma verificação de redundância cíclica (CRC)
Slides de Redes de Computadores I,Nicolau Pinto,Maria João

Depois de realizarmos a verificação de redundância cíclica, chegamos á conclusão que a mensagem a ser enviada será a sequência de bits *10111011*, os bits *101110* da mensagem a enviar mais os 3 bits *011* que advém do resto da divisão.

Desenvolvimento

Recursos e Tecnologias necessárias

Para a realização deste projeto será necessário:

Hardware:

- 2 computadores pessoais – interface final para o utilizador, desenvolvimento do código, realização dos relatórios, pesquisas e testes;
- 2 Placas Arduíno ESP32 – comunicação com os *transceivers*;
- 2 *Transceivers* RF nRF24L01+ – ligação/comunicação via radiofrequência;
- 2 Cabos Micro-USB – ligação entre os computadores pessoais e as placas Arduíno;

Software:

- Microsoft Word – realização dos relatórios;
- Arduíno IDE – comunicação com a placa ESP32;
- InteliJ – desenvolvimento da aplicação em JAVA;
- GanttProject – planeamento do projeto em diagramas de Gantt;
- Discord – realização do projeto a distância entre o grupo;
- Github- sincronização do código entre o grupo.

Figura 12 - Logotipo dos softwares

Esquema de Ligação ESP32-TRF

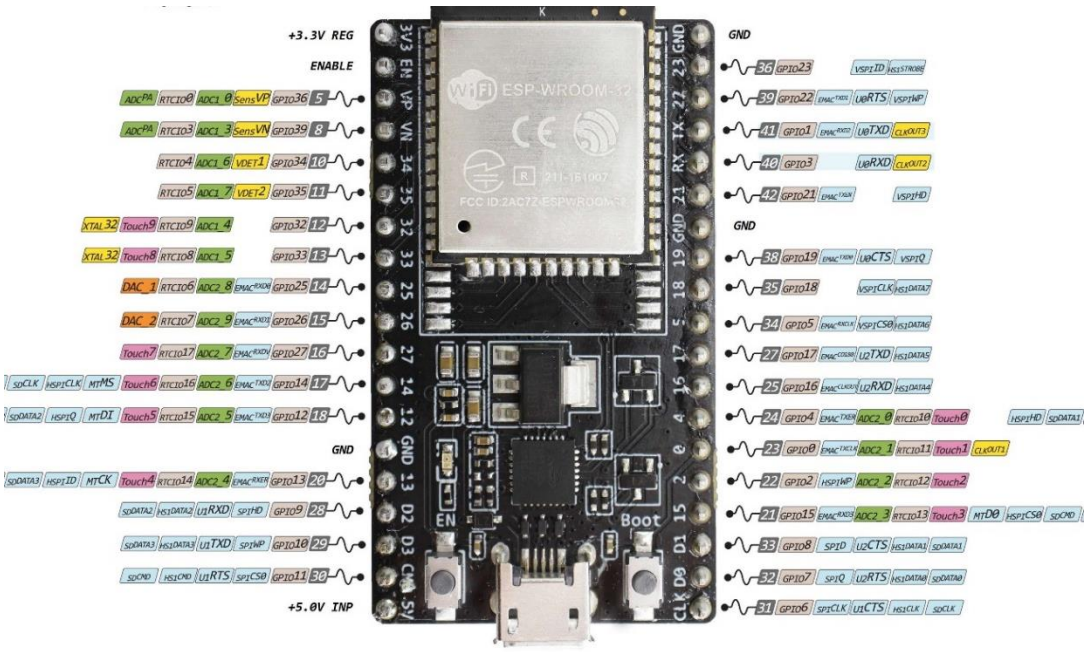

Figura 13-Pinout ESP32
https://blog.eletrogate.com/conhecendo-o-esp32-introducao-1/

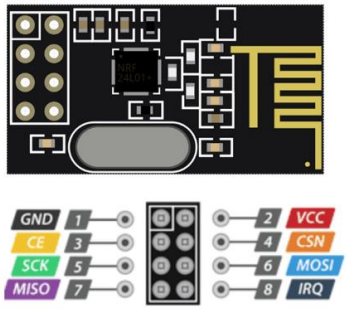

Figura 14- Pinout TRF
https://diyusthad.com/2020/05/nrf24l01-pinout.html

Para a implementação da interface SPI tivemos de realizar algumas ligações entre a placa ESP-32, que atuará como Master, e o *transceiver* RF (nRF24L01+) que atuará como Slave. Para que isto seja possível fizemos as seguintes ligações:

- Ligamos o GND (pino 1 da imagem 4) ao GND da ESP32
- Ligamos o VCC (pino 2 da imagem 4) ao 3V3 da ESP32
- Ligamos o CE (pino 3 da imagem 4) ao GPIO 17 da ESP32
- Ligamos o CSN (pino 4 da imagem 4) ao GPIO 16 da ESP32
- Ligamos o SCK (pino 5 da imagem 4) ao GPIO 18 da ESP32
- Ligamos o MOSI (pino 6 da imagem 4) ao GPIO 23 da ESP32
- Ligamos MISO (pino 7 da imagem 4) ao GPIO 19 da ESP32

Para a implementação desta fase do projeto tivemos de desenvolver um método no Arduino IDE usando a libraria RF-24.h, para que ocorra uma comunicação eficiente por radiofrequência entre a placa ESP32 e os respetivos *transceivers* desabilitando todas as funcionalidades de deteção/correção de erros tal como foi pedido pelos docentes.

Primeiramente, depois de fazermos a ligação entre os Arduíno e os respetivos *transceivers*, decidimos que iriamos utilizar um Arduíno como emissor, que ficará responsável por enviar as tramas nos vários testes a ser realizados, e um segundo Arduíno que funcionará como emissor. Esta condição só não se verificará na parte do *chat* visto que desenvolvemos um programa de *chat* bidirecional onde tanto um Arduíno como outro alternam o seu estado entre emissor/recetor. Para isso, tivemos de escolher um canal de transmissão e através da consulta do *datasheet* do *transceiver* de RF **[12]** constatamos que para o RF24 existem 125 canais disponíveis para escolha, sendo que a frequência dos canais varia entre 2.4GHz até 2.525 GHz.

A frequência de cada canal pode ser calculada através da seguinte expressão:

$$f = 2400 + RF\_CH[MHz]$$

O grupo definiu que iriamos usar o canal 100, e através da fórmula apresentada acima concluímos que a frequência para a qual o nRF24L01 irá trabalhar será a 2500MHz.

Formato das tramas

De seguida, definimos o formato das tramas que iríamos utilizar durante o desenvolvimento do nosso programa, no entanto este formato será explicado detalhadamente a seguir. Começamos por atribuir 8 bits ao cabeçalho, 30 bytes para o p*ayload* e 8 bits para o CRC, ou seja, é mais eficiente para o controlo de erros utilizar apenas 8 bits.

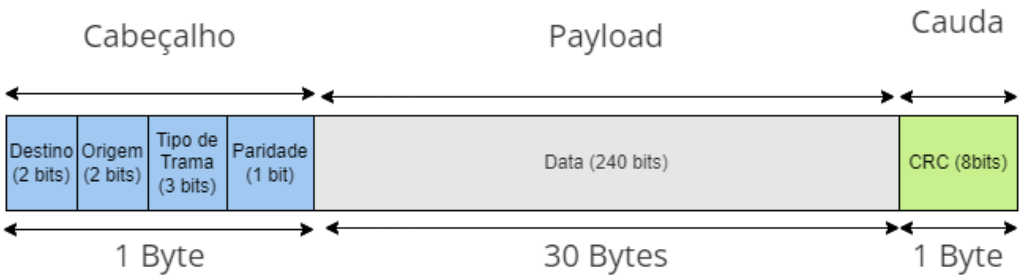

Figura 15 - Estrutura da Trama

O cabeçalho das tramas será subdividido em quatro partes, decidimos alocar 2 bits, tanto para origem da trama, como para o destino desta. Com 3 bits conseguimos identificar os vários tipos de trama, conseguindo até identificar oito tipos de tramas diferentes. Porém, apenas teremos de identificar se a trama em questão, é a primeira ou última trama do pacote de dados a ser enviado, se a trama é um ACK ou NACK proveniente do recetor, ou então se é uma trama intermédia. Sendo que temos 5 tipos de tramas distintas tivemos de alocar 3 bits. Por fim, atribuímos um bit ao campo paridade de forma a termos uma melhor gestão e controlo de erros.

Os dados que realmente queremos que sejam transmitidos irão constar no *payload*, sendo que se o pacote for maior que 30 bytes, esse pacote será subdivido em partes mais pequenas de 30bytes de forma a agilizar o processo de transmissão. Para terminar, a cada a trama será adicionado o CRC que é constituído por oito bits, pois caso haja erros de transmissão, a trama perdida ou corrompida possa ser reenviada novamente, evitando assim a perda de pacotes.

Cada trama a ser transmitida será construída no emissor, o payload será constituído pelos dados enviados para a porta emissora, e depois tanto o cabeçalho como a cauda serão adicionados automaticamente com valores que já predefinimos.

Fluxograma do “chat” (Fase 2)

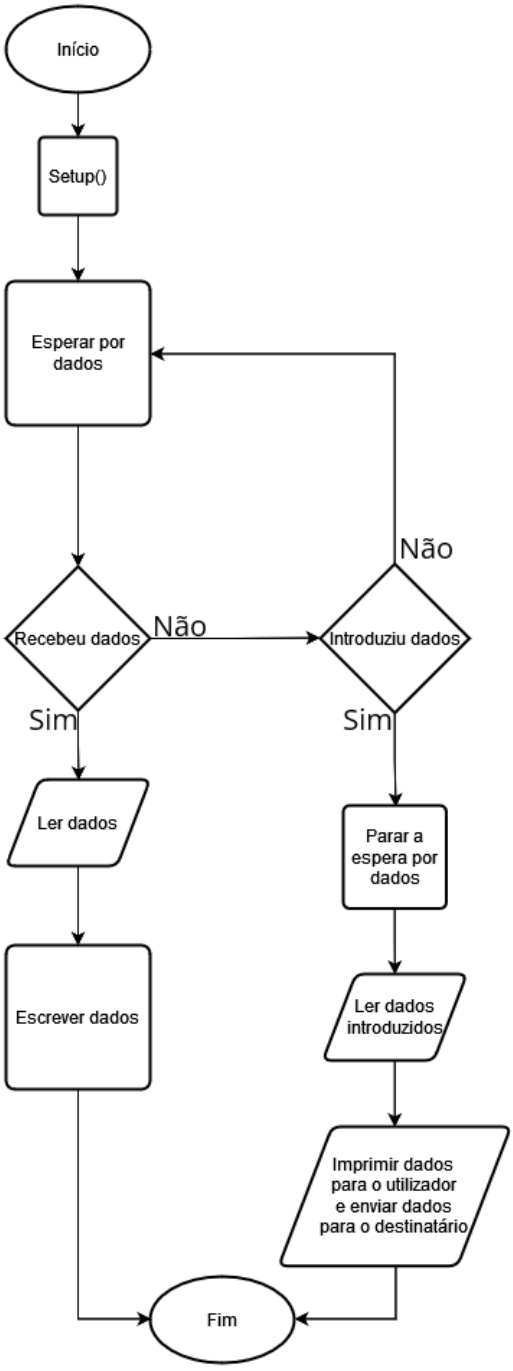

Figura 16 - Fluxograma do chat da Fase 2

Fluxograma do Round-Trip-Time (Fase 2)

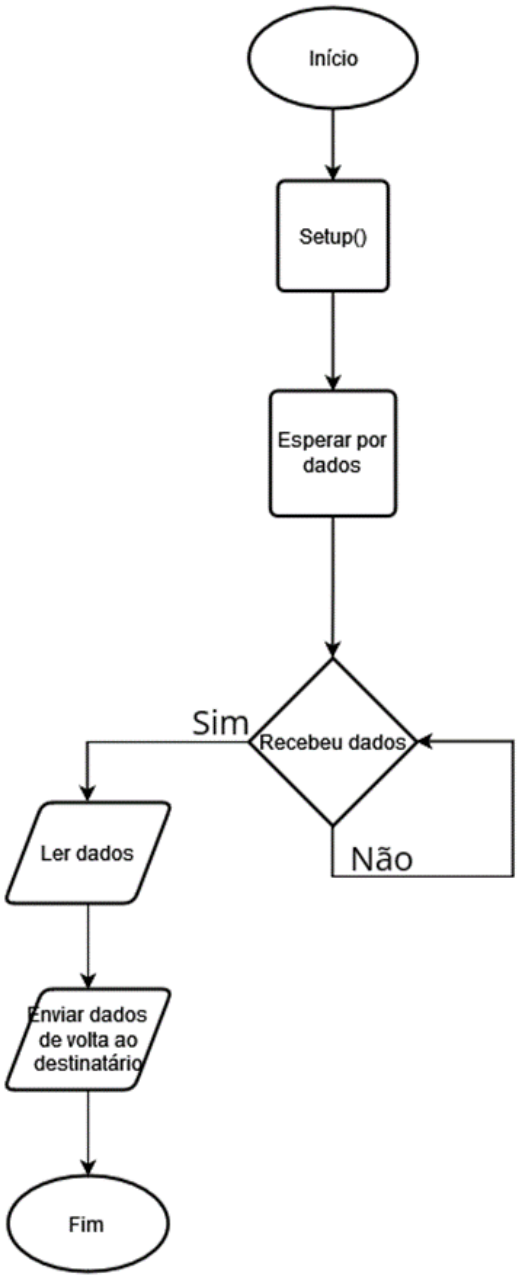

Figura 17 - Fluxograma Emissor RTT da Fase 2

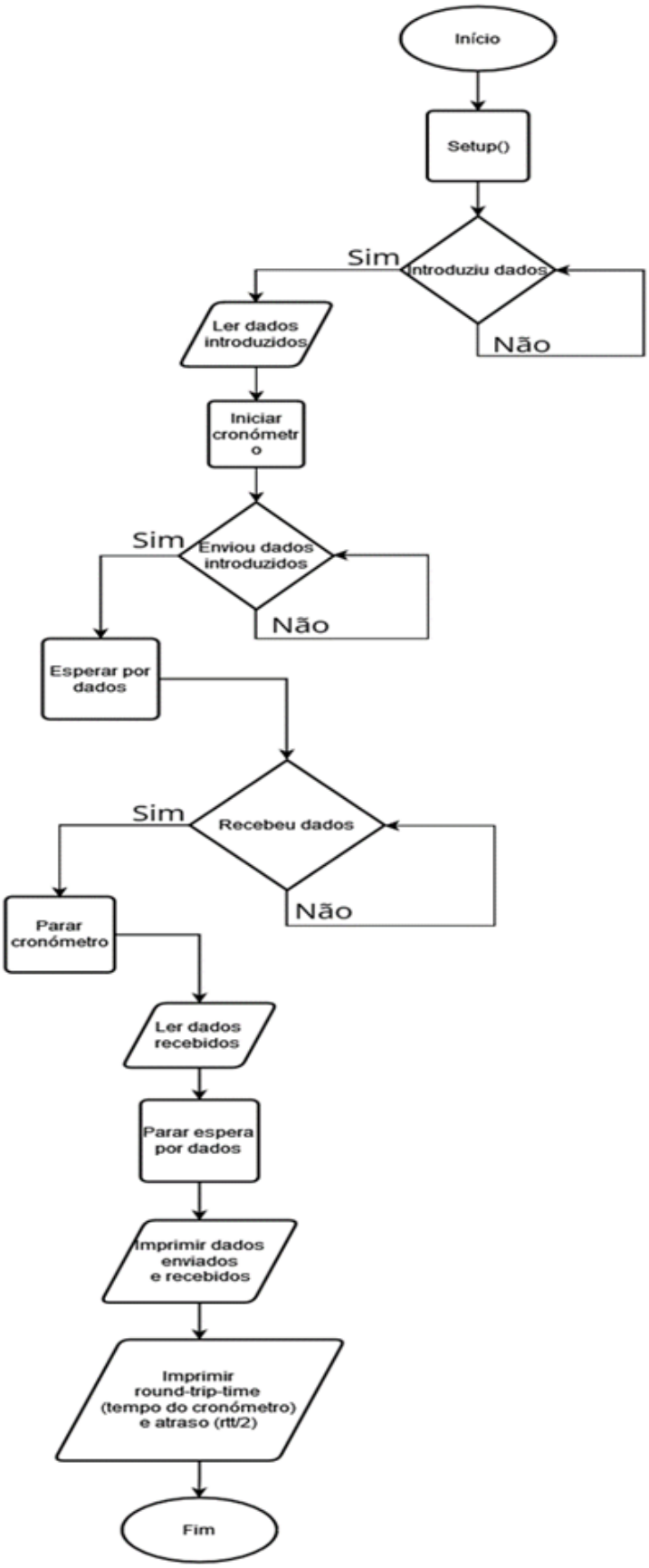

Figura 18 - Fluxograma Recetor RTT da Fase 2

Fluxograma do “chat” (Fase 4)

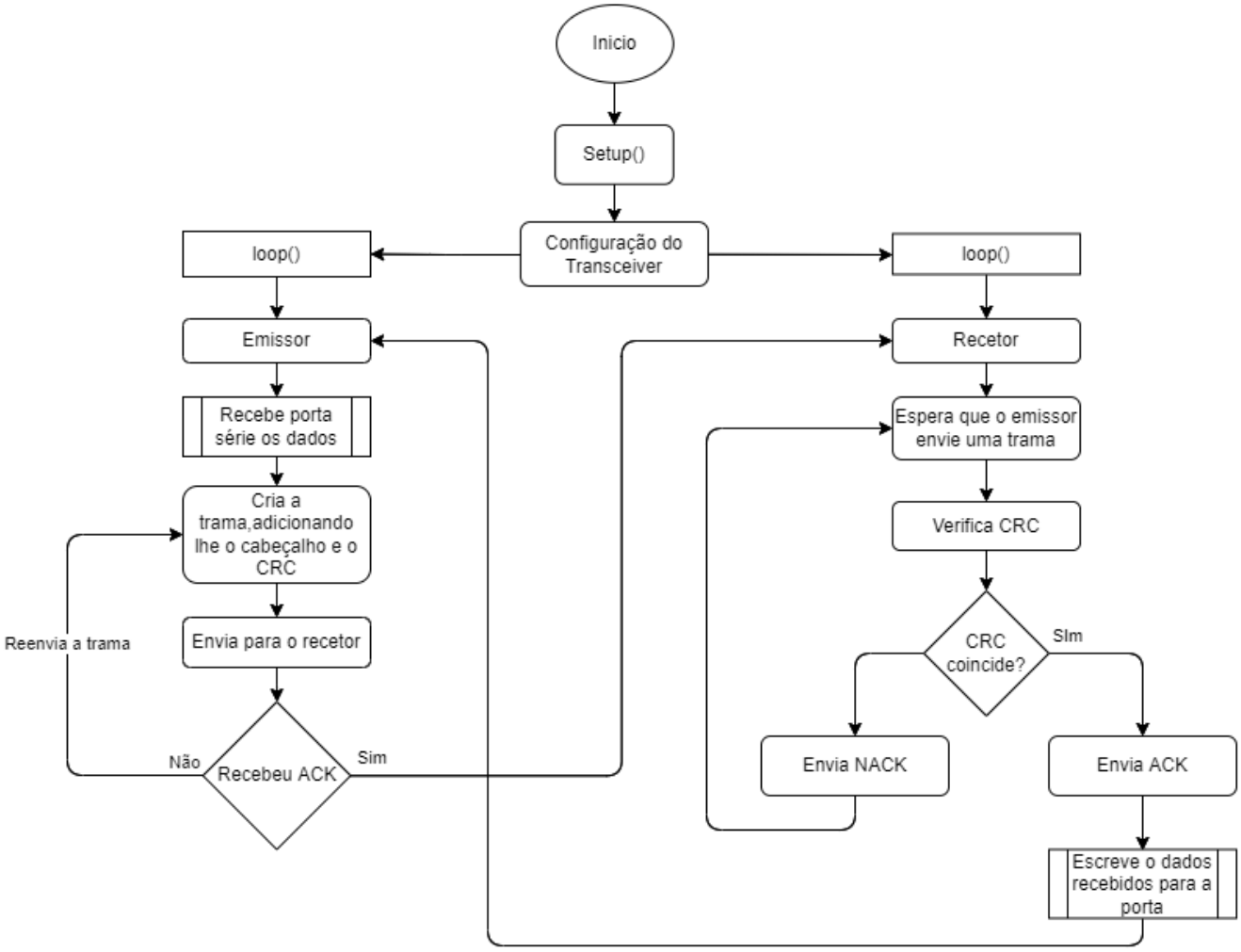

Figura 19 - Fluxograma do chat da Fase 4

Fluxograma do “file” (Fase 4)

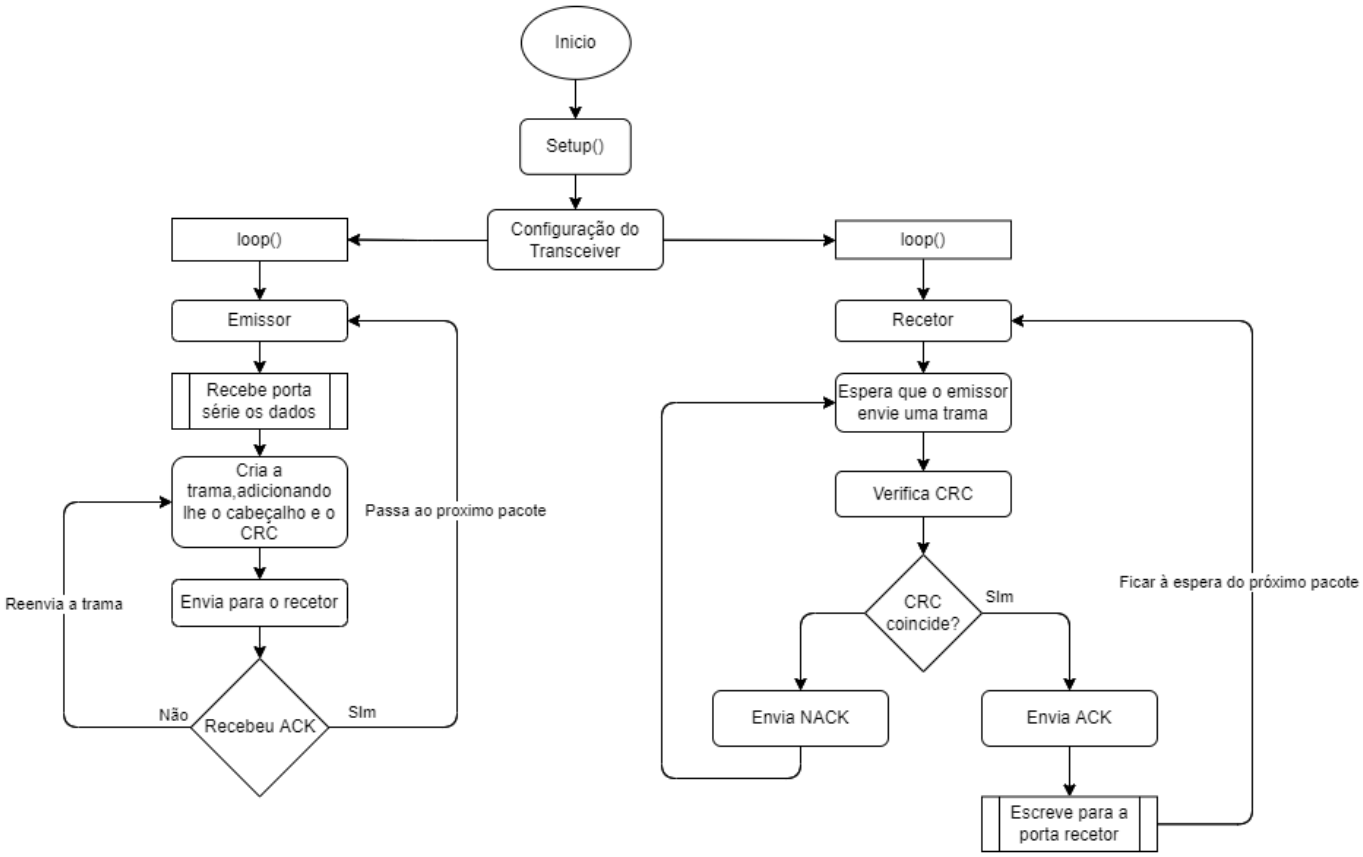

Figura 20 - Fluxograma do file da Fase 4

Fluxograma da Camada de Aplicação (Fase 4)

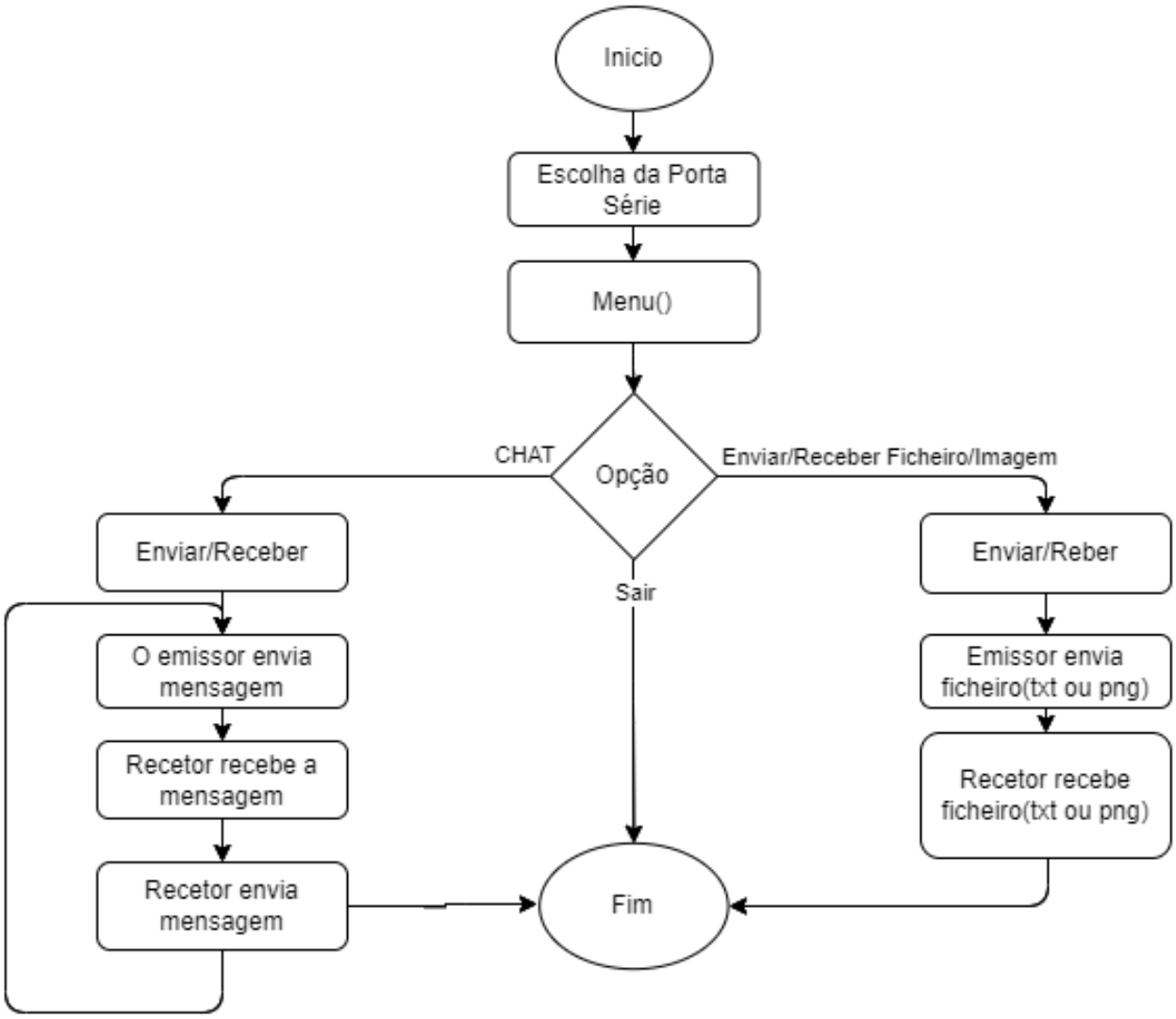

Figura 21 – Fluxograma da Camada de Aplicação da Fase 4

Testes

Round-Trip Time (RTT)

O RTT é o tempo que um pacote demora a enviar de uma estação emissora até uma estação recetora . O grupo decidiu utilizar um dos Arduínos como emissor e o outro como recetor, sendo que o primeiro irá ficar responsável por enviar as tramas de cada teste e o segundo fica responsável por as receber. As taxas de transmissão vão variar entre 3 valores para cada teste, ou seja, 250 Kbps, 1 Mbps e 2Mbps.

Para o cálculo do valor teórico do RTT, primeiro calculamos o tempo de transmissão e de seguida multiplicamos o tempo de transmissão por 2 para conseguir obter o RTT, tal como mostram as expressões seguintes:

$$T_{trans} = \frac{L}{R}$$

$$\text{RTT} = 2 * T_{trans}$$

RTT	Teórico (µs)	Prático (µs)
250 Kbps	4608	3830
1 Mbps	1024	1584
2 Mbps	512	1202

Tabela 2 - Valores do RTT da Fase 2

Débito (*Throughput*)

O *throughput* é a quantidade de dados transmitidos entre dois ou mais sistemas, ou seja, vamos enviar 1000 pacotes de um Arduíno para o outro, onde consideramos um como o emissor e outro como o recetor.

Para os testes efetuados pelo grupo, as taxas de transmissão vão variar entre 3 valores, 250Kbps, 1Mbps e 2Mbps, respetivamente.

Data Rate	Prático (Kbps)
250Kbps	173.084
1Mbps	466.302
2Mbps	639.980

Tabela 3 - Valores do Débito da Fase 2

Alcance de Transmissão

Para a realização dos testes de alcance, foram enviados 1000 pacotes entre o emissor e o recetor a diferentes distâncias para assim comparar os pacotes que se iam perdendo conforme a distância ia aumentando.

Para cada distância foram efetuados dois ensaios, sendo depois calculado o valor médio e posteriormente o *Lost Packet Rate (LPR)* como apresentado na tabela abaixo.

Distância (m)	**1**	**10**	**20**
Lost Packets	0	34	143
LPR (%)	0	3,4	14,3

Tabela 4 - Valores práticos do alcance da Fase 2

Distância (m)	**1**	**10**	**20**
Lost Packets	0	0	10
LPR (%)	0	0	1,0

Tabela 5 - Valores práticos do alcance da Fase 3

Discussão de Resultados

Relativamente ao *Round-Trip Time* realizamos testes para três taxas de transmissão diferentes, 250Kbps, 1Mbps e 2Mbps. Observamos discrepâncias dos valores teóricos para os práticos, isto é, os valores práticos, à exceção do último, foram inferiores aos teóricos, para uma taxa de transmissão de 250 Kbps obtemos 4068 microssegundos para o valor teórico e 3830 microssegundos para o prático. Isto pode acontecer por várias razões, o facto de não estar a ser testado nas condições corretas, haver interferências de ruídos, telemóveis ou outros tipos de aparelhos tecnológicos que estejam presentes no local do teste e também os atrasos que existem no software e hardware do Arduíno.

Em relação aos testes realizados para o débito, observamos que o valor prático foi bastante inferior ao esperado, ou seja, a diferença de um valor para o outro é demasiado elevada. No entanto, quanto maior for a data rate, o débito irá também ter um valor mais elevado.

Por último, realizamos os testes de alcance de transmissão com 3 distâncias diferentes, onde enviamos 1000 pacotes entre o emissor e o recetor. Efetuaram-se 2 testes para cada distância, 1m, 10m e 20m, sendo depois calculada a média entre os valores obtidos. A respeito destes ensaios, concluímos que quanto maior for a distância entre os dois Arduínos, maior será a quantidade de pacotes que não chegam com sucesso ao recetor, tal como era de esperar.

Relativamente aos testes de alcance de transmissão da 3ºfase, onde foi implementado o protocolo *Stop-and-wait*, foi notável a melhoria no desempenho e fiabilidade da comunicação entre os dois módulos Arduínos. Através da tabela (tabela 5), observa-se que nesta fase, houve uma mudança notória para uma distância de 10 e 20 metros, onde não se perderam praticamente pacotes nenhuns, devido às retransmissões, controlo de fluxo e controlo de erros.

Conclusão

Para a realização deste projeto foi necessário aprofundar e adquirir conhecimentos em várias áreas, assim como foi necessário aplicar os conhecimentos lecionados tanto em cadeiras de anos anteriores como em cadeiras que estamos a ter neste ano letivo, como é o caso de Redes de Computadores I, em que necessitamos do modelo OSI, entre outras coisas.

A placa NRF24 foi utilizada pelo grupo para efetuar as comunicações necessárias entre os transceivers, onde foi necessário desenvolver códigos relativos para cada fase do projeto.

Apesar das adversidades que foram surgindo ao longo da execução, a persistência e o trabalho de equipa foi sempre uma das principais qualidades deste grupo, onde tentávamos reunir várias vezes por semana para debater as ideias e atribuir as tarefas a cada membro.

Em suma, consideramos que este projeto prático possuía um grau de dificuldade elevado, mas que foi bastante útil para o nosso crescimento cognitivo, onde aprendemos como comunicar entre placas através de radiofrequência. Ainda de realçar, concluímos todas as fases com sucesso e alcançamos todos os objetivos delineados inicialmente.

Referências Bibliográficas

1. https://en.wikipedia.org/wiki/RF_module
2. http://wiki.sunfounder.cc/index.php?title=NRF24L01_Wireless_Transceiver_Module
3. https://pt.wikipedia.org/wiki/Serial_Peripheral_Interface
4. https://repositorio.ul.pt/bitstream/10451/27954/1/ulfc121684_tm_Jo%C3%A3o_Sabino.pdf
5. https://pt.wikipedia.org/wiki/Media_Access_Control
6. https://www.geeksforgeeks.org/carrier-sense-multiple-access-csma/
7. https://www.hardware.com.br/termos/csma-cd
8. https://www.techtarget.com/searchnetworking/definition/throughput
9. https://computer.howstuffworks.com/question525.htm
10. https://paginas.fe.up.pt/~mricardo/02_03/rcd/teoricas/ligdados_v4.pdf e Slides da aula teórica de Redes de Computadores I
11. https://edisciplinas.usp.br/pluginfile.php/5809966/mod_resource/content/1/REDES_Aula09.pdf
12. https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf