Universidade do Minho
Escola de Engenharia

Laboratórios de Telecomunicações e Informática I

ENGENHARIA DE TELECOMUNICAÇÕES E INFORMÁTICA

2020/2021

(Docentes: José Augusto Afonso, Nuno Vasco Moreira Lopes)

7 de janeiro de 2020

Relatório

FASE 3

Inês Barreira Marques – a84913@alunos.uminho.pt

José Pedro Fernandes Peleja - a84436@alunos.uminho.pt

Rui Filipe Ribeiro Freitas - a84121@alunos.uminho.pt

Tiago João Pereira Ferreira - a85392@alunos.uminho.pt

Índice

Índice de figuras **4**

Índice de tabelas **5**

Lista de abreviaturas **6**

Introdução **7**

Controlo da Ligação Lógica **8**

1. Fundamentos 8
   - 1.1. Trama/Pacote 8
   - 1.2. Protocolos de Comunicação 10
     - 1.2.1. *Stop-and-wait* 10
     - 1.2.2. *Sliding Window* 11
   - 1.3. Controlo de erros 12
     - 1.3.1. *Checksum* 12
     - 1.3.2. CRC 13
2. Desenvolvimento 15
   - 2.1. Fluxograma 16
   - 2.2. Formato da trama 17
3. Testes 18
   - 3.1. Round-trip time 18
   - 3.2. Alcance 19
   - 3.3. PER (Packet Error Rate) 19
4. Discussão de resultados 20

Conclusão **21**

Referências **22**

Autoavaliação **23**

Índice de figuras

Figura 1 - Constituição da trama. 8
Figura 2 - Estrutura do cabeçalho. 9
Figura 3 - Estrutura do payload. 9
Figura 4 - Estrutura da cauda. 9
Figura 5 - Mecanismo Stop-and-wait. 10
Figura 6 - Mecansimo Sliding-window. 11
Figura 7 - Janela Deslizante. 11
Figura 8 - Cálculo do Checksum. 12
Figura 9 - Exemplo de polinómio gerador 13
Figura 10 - Exemplo de cálculo de CRC. 13
Figura 11 - Verificação do cálculo do CRC. 14
Figura 12 - Fluxograma. 16
Figura 13 - Formato da trama de dados. 17

Índice de tabelas

Tabela 1 - Comparação valores RTT. 18
Tabela 2 - Valores alcance (pacotes retransmitidos). 19
Tabela 3 - Valores do PER (em %). 19

Lista de abreviaturas

LTI – *Laboratórios de Telecomunicações e Informática*

ACK – *Acknowledgement*

NACK – Not *Acknowledgement*

CRC – *Cyclic Redundancy Check*

FCS – *Frame Check Sequence*

Introdução

No âmbito da Unidade Curricular de LTI I foi-nos proposto o desenvolvimento de uma aplicação que permita a conversação em modo texto, em tempo real, entre dois computadores pessoais. Pretende-se que para além da possibilidade de ambos os utilizadores conversarem em modo texto possam enviar os mais variados ficheiros incluindo imagens.

Para a realização deste projeto iremos utilizar o modelo OSI que foi devidamente explicado na fase 1. As camadas que utilizaremos neste projeto serão as camadas 1, 2 e 7 que correspondem à camada física, à camada de ligação de dados e à camada de aplicação onde nesta segunda fase foca-se particularmente na segunda.

Este projeto está dividido em 4 fases em que cada uma terá uma entrega de um relatório e demonstração prática da aplicação em funcionamento, com exceção da última fase que para além disso terá também uma apresentação oral.

O presente relatório corresponde à fase 3 em que os objetivos principais passam por 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 concebendo as regras de comunicação, tramas e primitivas de serviço a oferecer à camada superior e definir os tipos de tramas e campos que necessitam de conter de modo a proporcionar as funcionalidades desejadas no âmbito da camada 2 do projeto para uma transmissão fiável de dados ponto-a-ponto, com ênfase na deteção e correção de erros. Para além disto devemos disponibilizar nesta fase as primitivas às camadas superiores através de API's e efetuar a transferência fiável de um fluxo de dados entre as duas placas ESP32 com a ligação RF sem fios.

De modo a sermos capazes de cumprir com os objetivos deste projeto semestral devemos pôr em prática conhecimentos adquiridos noutras unidades curriculares, nomeadamente Redes de Computadores I, Sistemas Operativos, Paradigmas de Programação, entre outras.

Controlo da Ligação Lógica

Após o final da fase anterior o grupo ficou com uma vasta noção na transferência de dados entre dois computadores através de radiofrequência, no entanto com alguns erros. Para esta terceira fase ficou mais claro o significado de uma trama e como controlar/corrigir esses erros. Para além disto foi estabelecido um protocolo de comunicação para a transferência de dados.

Desta forma, para que os erros possam ser controlados e corrigidos teremos de implementar protocolos e funções que permitam realizar as tarefas necessárias.

1. Fundamentos

1.1. Trama/Pacote

As tramas são pequenas partições de um ficheiro a ser enviado pela rede. Estas são constituídas por três divisões: o cabeçalho, o *payload* e a cauda.

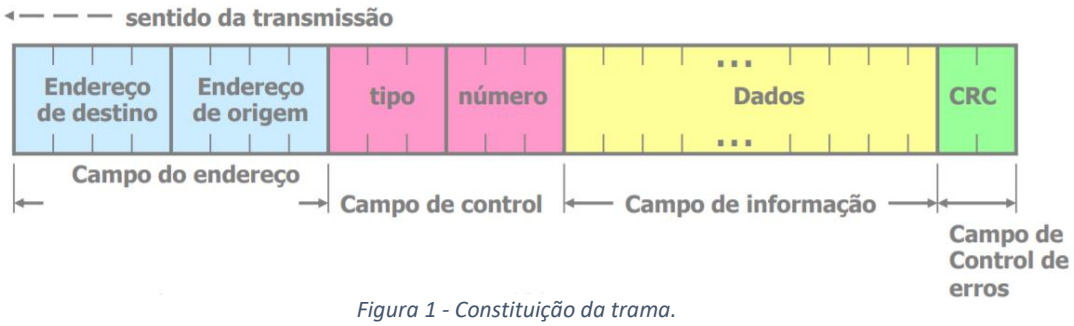

Figura 1 - Constituição da trama.

Como vemos ilustrado na figura 1 a trama está dividida em diferentes secções, sendo que o campo de endereço e o campo de controlo pertencem ao cabeçalho, o campo de informação/dados pertencem ao *payload* e o CRC pertence a cauda. Toda esta informação estará dentro dos 32 bytes da trama.

Vantagens do uso de tramas em vez dos ficheiros:

- O facto de a quantidade de dados a ser transmitida ser mais pequena, faz com que a deteção e correção de erros seja mais eficaz, pois não é necessário o reenvio do ficheiro total e apenas da trama onde ocorreu o erro;
- A velocidade de transmissão é maior devido à divisão do ficheiro em tramas mais pequenas, tornando o tempo de transmissão de cada trama menor.
- O tempo de processamento é também menor pois o processamento de pacotes menores é realizado mais rapidamente, visto que a trama se trata de um pacote menor que um ficheiro.

Como referido anteriormente a trama/pacote deve estar devidamente dividida em 3 secções e com os devidos nomes, tamanhos e conteúdo de cada campo (devidamente estruturado).

Assim sendo, na primeira divisão será o cabeçalho que terá o campo do endereço que irá apresentar dentro dele os endereços de destino e de origem, sem esquecer que também irá ter o campo de controlo onde estão definidos o tipo e o número da trama. O cabeçalho irá ocupar uma quantidade de *bytes* definida em código dos 32 *bytes* disponíveis.

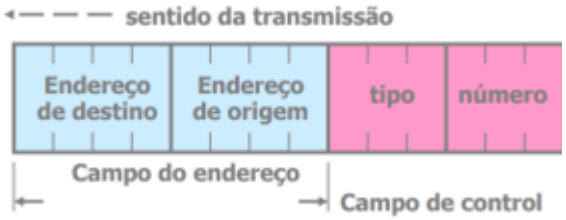

Figura 2 - Estrutura do cabeçalho.

De seguida está presente o *payload* que é o campo de informação, dentro deste estarão os dados que pretendemos enviar presentes na trama. Este também irá ocupar um número de bytes definido em código.

Figura 3 - Estrutura do payload.

Finalmente a cauda que será o campo de controlo de erros, aqui estará presente o CRC que corresponde ao controlador dos erros presentes na trama.

Figura 4 - Estrutura da cauda.

1.2. Protocolos de Comunicação

Existem diversas técnicas relativas ao controlo de fluxo e de erros, sendo estas importantes para que não haja uma sobrecarga na estação que recebe os dados. Para esta fase temos de escolher uma e implementá-la, como estamos mais familiarizados com 2 técnicas lecionadas na UC de Redes de Computadores vamos optar por uma dessas onde explicaremos melhor cada uma delas em seguida.

1.2.1. *Stop-and-wait*

No mecanismo de *stop-and-wait*, após a transmissão de uma trama, a estação emissora aguarda pela confirmação da estação recetora (ACK) para transmitir a trama seguinte. Esta técnica funciona bem quando queremos enviar poucas tramas de grande dimensão, no entanto quando as tramas são de tamanho muito grande é maior a probabilidade de existirem erros, é maior a ocupação dos recursos como os buffers e os processadores e o desempenho da ligação tende a piorar. Se a mensagem a enviar for fragmentada em tramas de pequena dimensão o *stop-and-wait* torna-se inadequado devido ao tempo que leva a enviar uma trama de cada vez e esperar pela sua confirmação.

Na figura 5 podemos ver o processo a decorrer ao longo do tempo em que ao ser enviado uma trama do emissor, este espera pelo ACK (*acknowledgement*) vindo do recetor e só depois envia a trama seguinte. Vários problemas podem ocorrer, quer sejam erros na trama ou o recetor estar ocupado. Se ocorrerem erros na trama quando esta chega ao recetor, existem duas soluções, ou é enviado pelo recetor um NACK (not *acknowledgement*) ou então através da implementação de um *timeout* que serve para contar até um tempo pré-definido e caso esse tempo seja atingido o emissor volta a enviar o mesmo pacote. Se o recetor estiver ocupado, apenas envia o ACK quando for possível. **[1]** Este foi o protocolo escolhido pelo nosso grupo para a implementação.

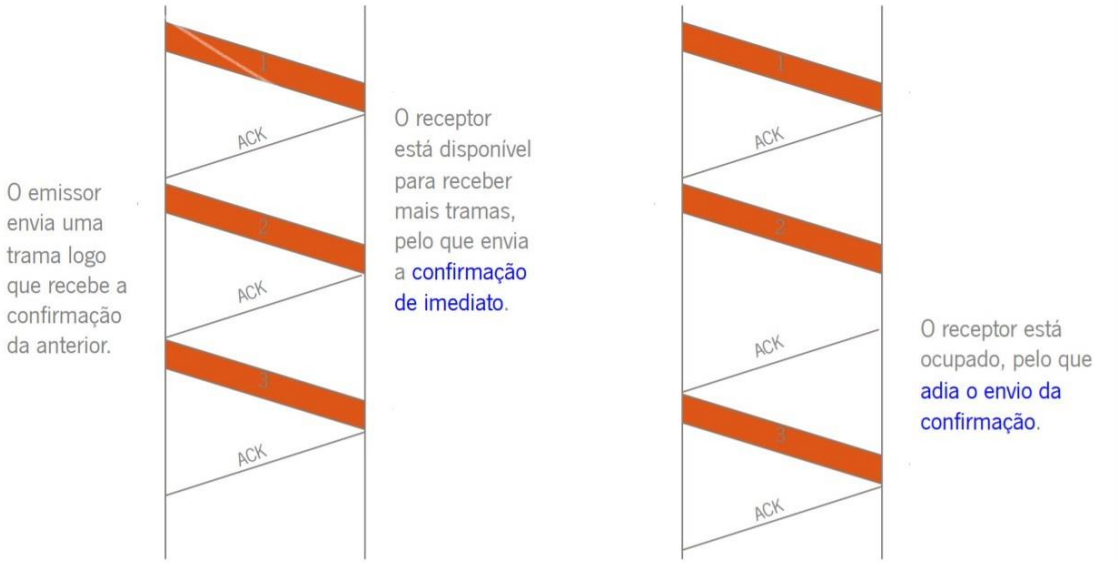

Figura 5 - Mecanismo Stop-and-wait.

1.2.2. *Sliding Window*

Os mecanismos de janela deslizante (*sliding window*) permitem fazer uma utilização mais eficiente da ligação. Neste mecanismo em específico, o emissor pode enviar um número de tramas pré-definido e espera pela confirmação dessas tramas. Ou seja, neste processo apenas são enviadas mais tramas quando são recebidas todas as confirmações das tramas previamente enviadas como podemos observar na figura 6.

De modo a garantir que as tramas e as confirmações não excedem a capacidade da janela deslizante, os pacotes e os *ACK's* são numerados de 0 a W, permitindo um maior controlo de fluxo e uma transferência mais eficiente. **[1]**

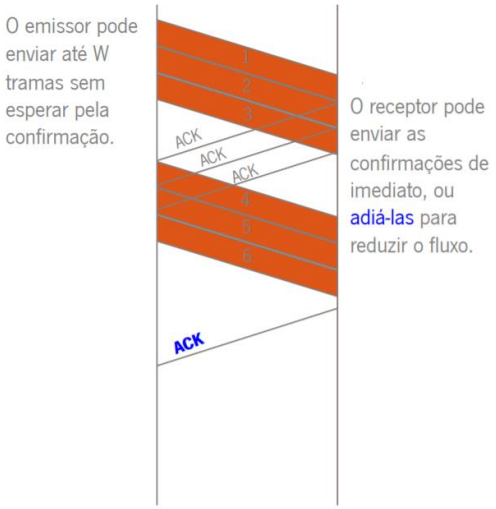

Figura 6 - Mecansimo Sliding-window.

Através da figura 7 conseguimos ver o porquê deste mecanismo se denominar de "janela deslizante", após o envio das W tramas do emissor para o recetor, a partir do momento que começa a receber as confirmações provenientes do recetor, esta desliza uma unidade para a direita e envia a trama seguinte.

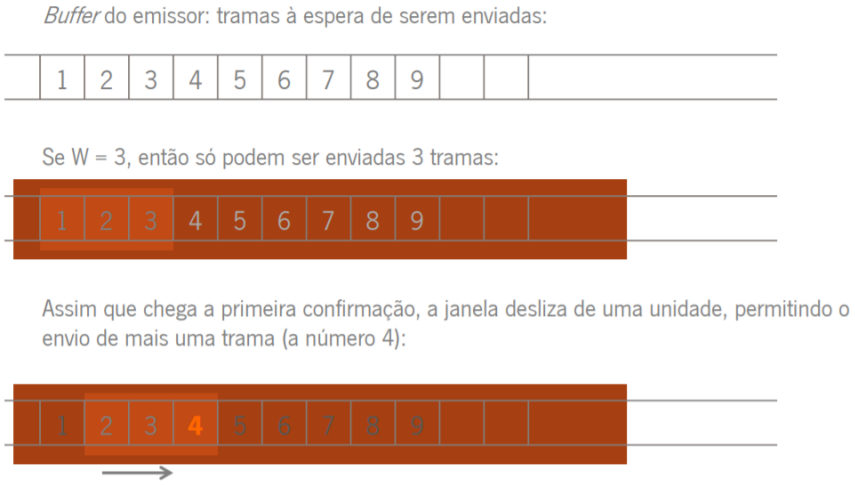

Figura 7 - Janela Deslizante.

1.3. Controlo de erros

1.3.1. *Checksum*

No emissor, a mensagem a ser enviada é dividida em partes iguais e de forma sequencial, após ter-se a mensagem toda divida, somam-se todas as parcelas, depois ao resultado obtido no final dessas somas faz-se o complemento para 1, que é passar os 1's para 0's e os 0's para 1's. Esse complemento é anexado ao final da mensagem que é depois enviada.

No recetor, faz-se o mesmo processo que no emissor, de dividir a mensagem em tamanhos iguais e em seguida somar novamente as sequencias incluindo o complemento para 1 do resultado da soma do emissor. O complemento da soma do recetor será só 0's, o que significa que não ocorreu corrupção de dados, no entanto pode acontecer de haver corrupção de dados e não serem detetados pelo *checksum*, ao serem alterados alguns bits na sequência a soma pode dar a mesma, fazendo com que não sejam detetadas as corrupções e os erros, embora não seja muito habitual este tipo de erros acontecer.

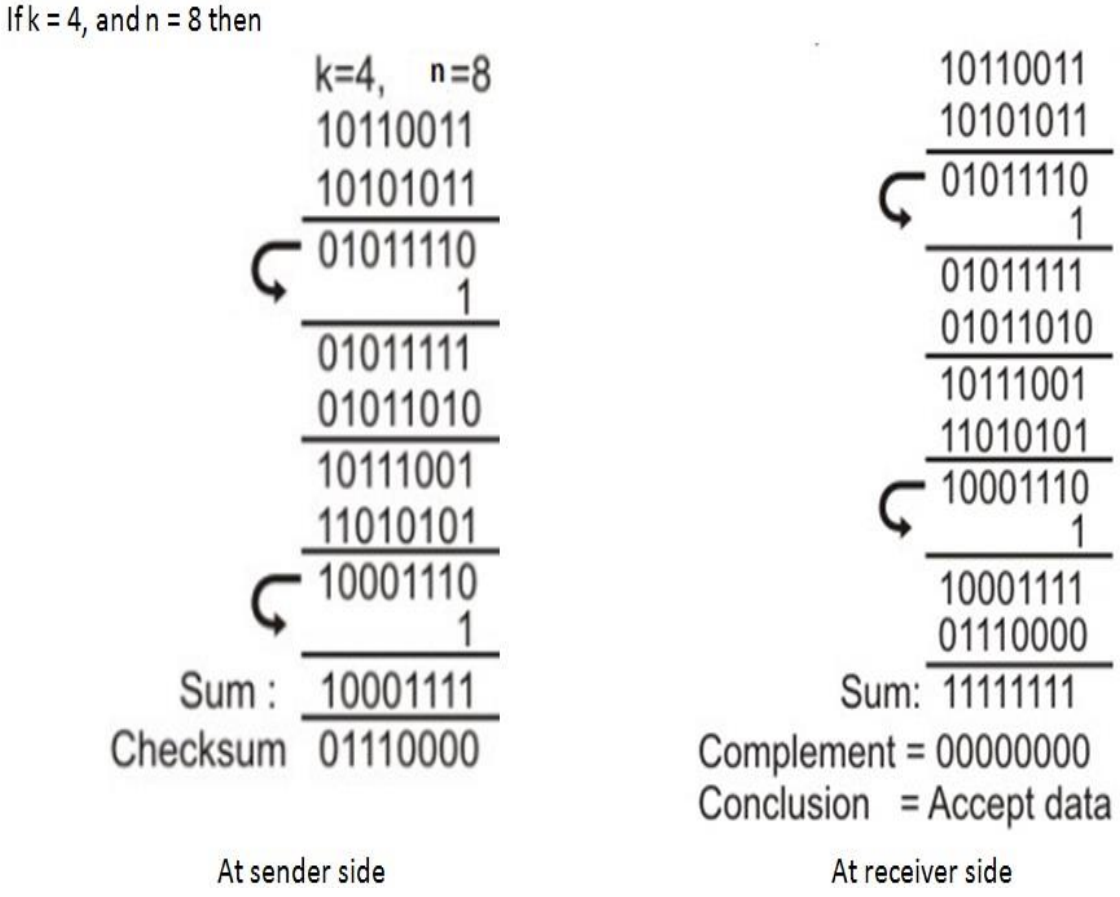

Figura 8 - Cálculo do Checksum.

1.3.2. CRC

Este método apesar de maior complexidade na realização de cálculos revela-se um método mais eficiente e mais adequado para o problema que nos é colocado. Este método é geralmente representado através de polinómios de uma variável de coeficientes binários. Este pode falhar se o número de erros for superior a capacidade de deteção do código.**[1]**

Em suma, para calcular o CRC necessitamos de calcular o FCS, este método consiste em adicionar um conjunto de bits à mensagem a transmitir, a quantidade de bits a adicionar será igual ao grau do polinómio gerador. Para tal, teremos de utilizar também um polinómio gerador que estará predefinido.**[2]**

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

100000100110000010001110110111 (33bits, para dar resto 32)

Figura 9 - Exemplo de polinómio gerador

Como podemos observar neste exemplo o polinómio gerador e de grau 32 o que significa que seriam adicionados 32 zeros ao final da mensagem a enviar.

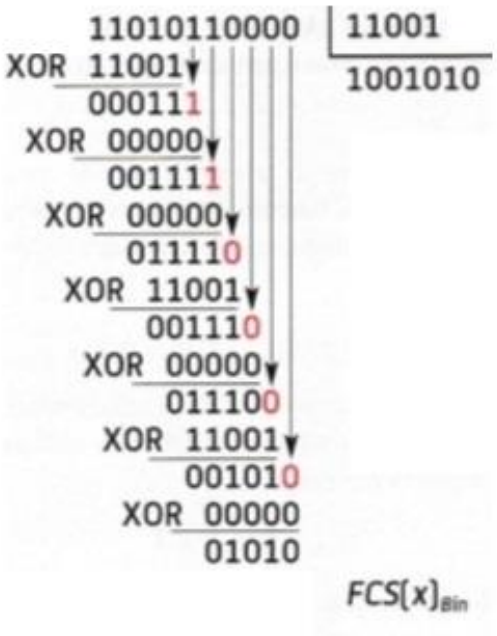

Figura 10 - Exemplo de cálculo de CRC.

Como podemos observar pelo exemplo anterior, para calcular o CRC devemos dividir a mensagem a enviar pelo polinómio gerador até este obter um resto. Esta divisão é uma divisão aritmética polinomial que é o mesmo que aplicar uma *gate* XOR.

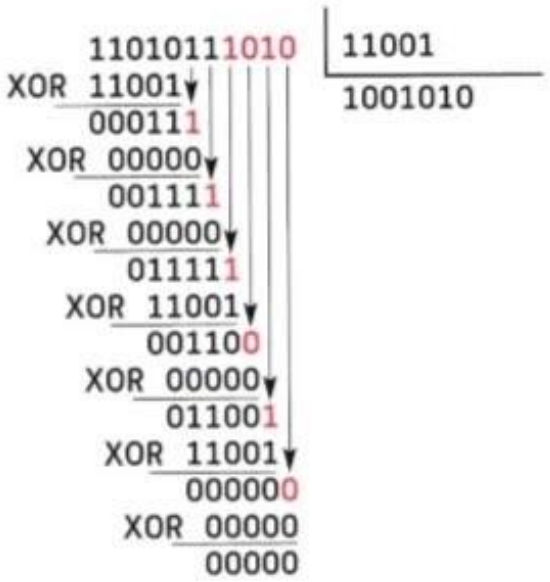

Figura 11 - Verificação do cálculo do CRC.

Finalmente, são substituídos os bits adicionados à mensagem pelo resto da divisão. Para confirmar que não existiram erros, repetiríamos o processo da divisão desta vez já com a mensagem que foi enviada. Se nesta situação o resto for zero significa que não houve erros na transmissão.**[2]**

Para a realização do nosso trabalho iremos utilizar um CRC de apenas 8 bits visto que achamos não ser necessário mais do que isso dado o tamanho reduzido da trama.

2. *Desenvolvimento*

Para esta fase do projeto foi necessário a realização de um novo código para a comunicação entre os *transceivers*, pois tivemos de adicionar o controlo de erros e de fluxo ausentes na fase anterior. Começamos por elaborar um fluxograma presente na figura 12, que retrata o processo completo que tínhamos de elaborar.

Após a elaboração do fluxograma começamos a pensar no formato das tramas que íamos utilizar de modo a que o desempenho do nosso programa fosse o melhor possível. Com isto em consideração decidimos utilizar tramas com 1 byte no cabeçalho, 30 para os dados e 1 de CRC, isto pois desta maneira conseguimos controlar os erros eficientemente e mesmo assim conseguir enviar 30 bytes de dados por pacote, o que faz com que seja uma transferência mais eficiente do que se utilizássemos por exemplo 2 bytes para o CRC. O formato das tramas será explicado em detalhe mais à frente.

Em relação às tramas ACK e NACK decidimos optar por o envio de apenas 1 byte contendo o tipo da trama, os endereços de origem e destino e o número de sequência para o caso de ocorrer erros na transmissão do ACK do recetor para o emissor.

Após termos decidido o formato das tramas e feito o fluxograma passamos à realização do código em que no que toca às linhas de código mais relevantes para a sua execução não mudaram muito da fase passada, apenas tivemos de configurar o *transceiver* e após isso implementar certas funções para o controlo de erros. Para além disso nesta fase tiramos proveito de uma libraria do Arduíno IDE que fazia o cálculo do CRC **[3]**. Através disto no recetor bastou comparar o CRC calculado com o CRC proveniente do emissor e se fossem iguais o recetor enviaria um ACK, caso contrário envia um NACK.

2.1. *Fluxograma*

Este fluxograma apresenta tanto a linha de processo do emissor como do recetor em que o emissor tem como responsabilidade enviar as tramas e esperar pela confirmação das mesmas para passar sem erros para a próxima. No caso do recetor, a este cabe o trabalho de receber as tramas enviadas pelo emissor e compará-las de modo a ver se houve erros ou não no envio, depois disto envia uma confirmação ao emissor para que este possa continuar o seu processo de envio das tramas.

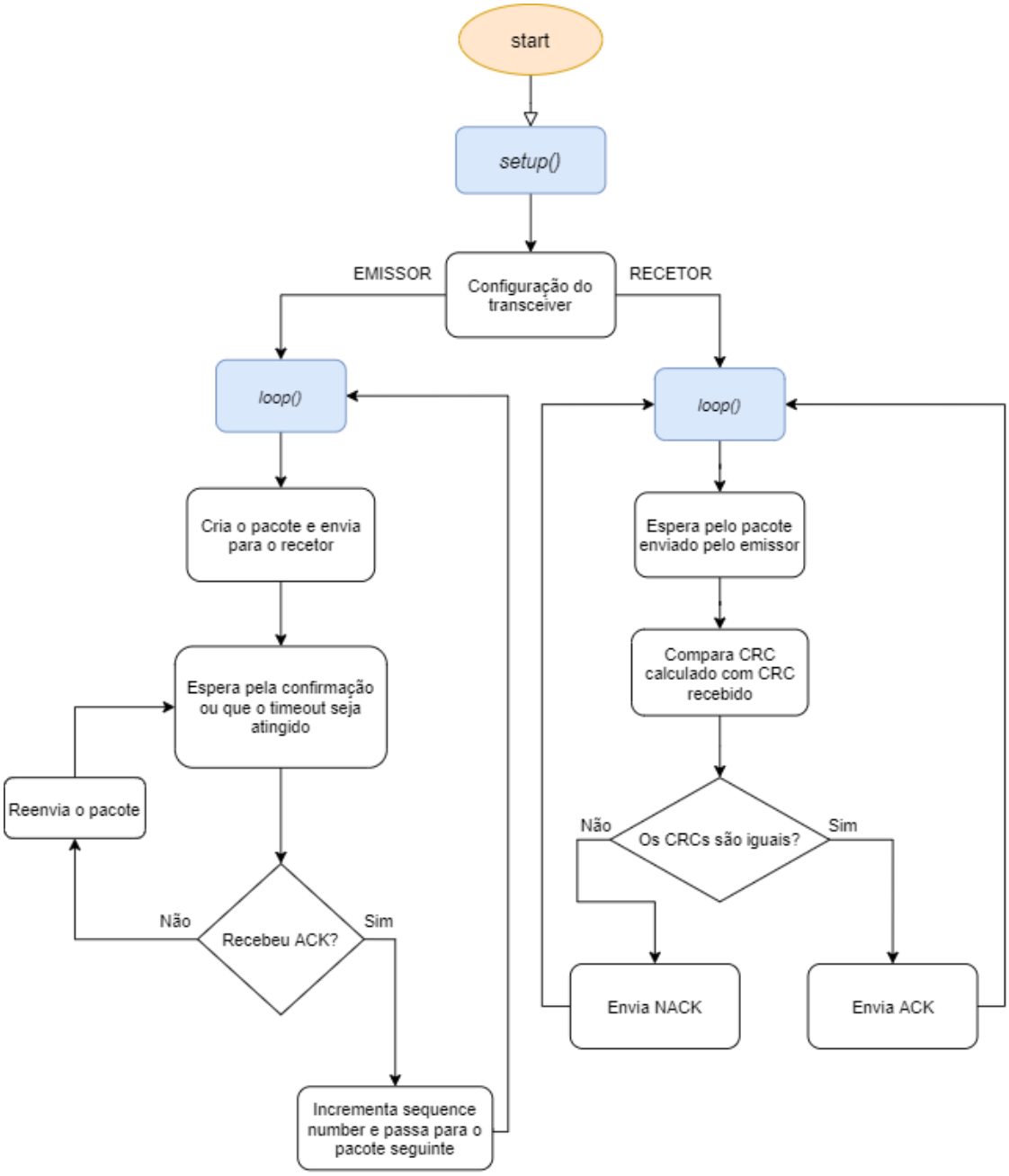

Figura 12 - Fluxograma.

2.2. *Formato da trama*

Na figura 13 está representada o formato da trama utilizada pelo nosso grupo para a transmissão de dados. A trama é constituída por 32 bytes, divididos em 3 divisões, a primeira **[A]** para o *header* (cabeçalho) com 1 byte, a segunda **[B]** para o *payload* com 30 bytes e a última **[C]** para o CRC com 1 byte.

Dentro do cabeçalho decidimos utilizar os primeiros 4 bits para o endereçamento, com os 2 primeiros para o endereço do emissor **[D]** e os outros 2 para o endereço do recetor **[E]**. Os 3 bits seguintes correspondem ao tipo de trama enviada **[F]**, esta que pode tomar 2 valores distintos, dependendo se a trama é normal ou final. Em relação ao último bit este representa o número de sequência **[G]** que dado estarmos a utilizar o mecanismo *Stop-and-wait* é suficiente o uso de um só bit. O tipo da trama podia ser implementado com apenas 2 bits e não 3, mas inicialmente pensámos em colocar um tipo para o início da trama, ideia que mais tarde se tornou inválida, mas que poderá vir a ser necessária numa fase posterior.

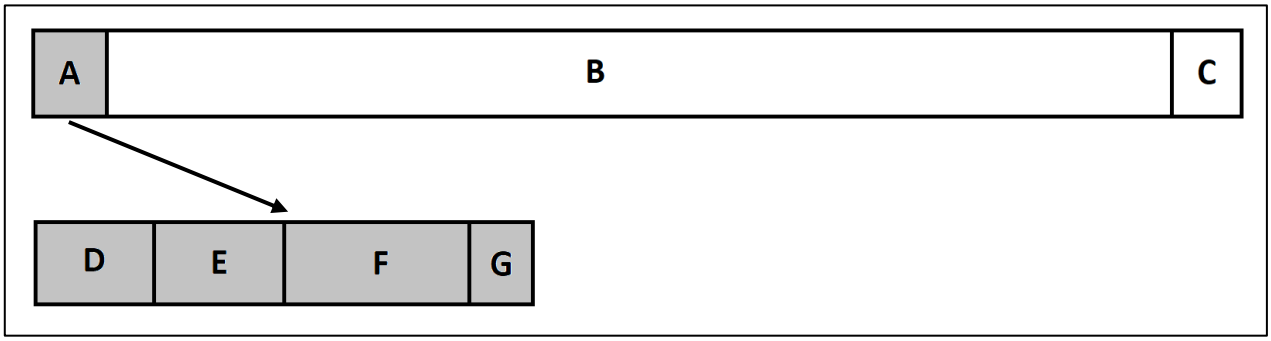

Figura 13 - Formato da trama de dados.

A -> *Header* (1 byte);

B -> *Payload* (30 bytes);

C -> CRC (1 byte);

D -> Endereço Emissor (2 bits);

E -> Endereço Destino (2 bits);

F -> Tipo (3 bits);

G -> Número de sequência (1 bit).

Para além da trama de dados também tivemos de formatar outros 2 tipos de tramas, a trama ACK e a NACK, em que ambas contêm somente um byte parecido ao *header* **[A]** da figura 13. A única diferença é que nestas tramas o tipo em vez de ser 001, será 010 para o ACK e 011 para o NACK.

3. *Testes*

3.1. *Round-trip time*

Para o cálculo do valor teórico do RTT utilizamos as seguintes expressões (1.1) onde é mostrada a forma de cálculo do tempo de transmissão, (1.2) para o cálculo do tempo de propagação e (1.3) onde calculamos o *round-trip time*. Na primeira expressão utilizamos *L* (tamanho do pacote em bits) com o valor 256 e *R* (taxa de transmissão) que varia entre 250 kbps, 1 Mbps e 2 Mbps. Na segunda utilizamos *d* que corresponde à distância dos módulos RF e *v* que corresponde à velocidade de transmissão. Visto que este valor é muito reduzido não foi levado em conta nos cálculos teóricos.

$$T_{trans} = \frac{L}{R} \quad (1.1)$$

$$T_{prop} = \frac{d}{v} \quad (1.2)$$

$$RTT = T_{trans} + T_{transACK} + 2 * T_{prop} \quad (1.3)$$

Na tabela 1 apresentamos os valores calculados assim como os valores teóricos com o objetivo de realizar uma melhor apreciação dos resultados.

Tabela 1 - Comparação valores RTT.

RTT	Teórico (microssegundos)	Prático (microssegundos)
250 kbps	1056	2295 - 2394
1 Mbps	264	1116 - 1211
2 Mbps	132	913 - 916

3.2. *Alcance*

Para o teste de alcance nesta fase definimos um número de pacotes a enviar, 250, e fizeram-se vários testes para 3 distâncias (10 centímetros, 1 metro e 10 metros) de modo a determinar máximo e mínimo de pacotes que eram retransmitidos. Na tabela seguinte encontram-se o número de pacotes que foram necessários retransmitir pelo emissor tanto por perdas do pacote como por pacotes corrompidos. Para uma melhor apreciação dos resultados realizamos vários testes em que apontamos o mínimo e máximo de pacotes retransmitidos para 3 taxas de transmissão diferentes.

Tabela 2 - Valores alcance (pacotes retransmitidos).

Alcance	10 centímetros		1 metro		10 metros	
	Mínimo	Máximo	Mínimo	Máximo	Mínimo	Máximo
250 Kbps	0	0	0	0	4	8
1 Mbps	0	0	0	0	7	11
2 Mbps	0	0	0	1	12	19

3.3. *PER (Packet Error Rate)*

No cálculo do *Packet Error Rate* utilizamos a fórmula (1.4) em que *pRetransmitidos* corresponde ao número de pacotes retransmitidos e *pTotal* ao número total de pacotes recebidos. Isto é multiplicado por 100 de modo a obter o valor em percentagem. Na tabela 3 apresentamos esses valores.

$$PER = \frac{pRetransmitidos}{pTotal} * 100 \qquad (1.4)$$

Tabela 3 - Valores do PER (em %).

Alcance	10 centímetros		1 metro		10 metros	
	Mínimo	Máximo	Mínimo	Máximo	Mínimo	Máximo
250 Kbps	0.0	0.0	0.0	0.0	1.6	3.2
1 Mbps	0.0	0.0	0.0	0.0	2.8	4.4
2 Mbps	0.0	0.0	0.0	0.4	4.8	7.6

4. *Discussão de resultados*

Em relação ao *Round-Trip Time* realizamos testes para 3 taxas de transmissão distintas. No código realizado no Arduíno IDE, a parte do emissor calcula por métodos complementares à biblioteca do RF o tempo que demora a enviar uma mensagem e a receber a confirmação por parte do recetor. Começamos por calcular os tempos teóricos, sem haver interferências de modo a comparar com os valores práticos calculados posteriormente. Quando fomos a testar o nosso código os valores deram-nos um pouco mais elevados, observamos que estas diferenças de tempo também acontecem para as outras taxas de transmissão e ocorreram devido às mais variadas interferências, por exemplo ruído, outros computadores e outros aparelhos tecnológicos presentes na sala onde realizamos os testes, mas também existem atrasos de software e hardware introduzidos pelos computadores e pela própria placa ESP. Para além disto nos cálculos tóricos não levamos em conta o tempo de processamento das placas ESP.

O objetivo por detrás da realização destes testes nesta fase foi possibilitar a comparação com os valores retirados na fase 2 deste projeto. Através da observação de ambas as tabelas retiramos que o tempo prático aumentou, isto deve-se ao facto de nesta fase termos adicionado o controlo de erros e de fluxo que faz com que o tempo de processamento do código aumente.

Em relação ao alcance, nesta fase nós observámos o número total de pacotes que eram retransmitidos, não podemos comparar com a fase anterior visto que nessa não havia controlo de erros pelo que não eram retransmitidos pacotes. Apesar disso recorrendo às tabelas do alcance desta fase conseguimos analisar a quantidade de pacotes que eram retransmitidos variando a distância entre 10 centímetros, 1 metro e 10 metros. Para os 10 centímetros e 1 metro reparamos que o número de pacotes que necessitavam de retransmissão eram quase sempre 0 já com 10 metros de distância estes números aproximavam-se dos 10 pacotes retransmitidos em cada 250. De referir que estes números também podem ter a ver com o ambiente em que os testes foram feitos visto ter sido num espaço com passagem de pessoas assim como a presença dos mais variados aparelhos tecnológicos.

Conclusão

Nesta fase chegamos à conclusão de que os conhecimentos de redes de computadores 1 são cada vez mais necessários.

Desta forma, para a deteção e correção de erros tivemos de realizar uma pesquisa de métodos fiáveis e eficientes tal como aplicar os métodos dados em redes de computadores 1, para tal foi necessário encontrar e utilizar bibliotecas de forma a aplicar os métodos de deteção de erros e de correção das mensagens enviadas. Foi necessária uma contribuição e organização geral do grupo para que fossem utilizados os métodos mais eficientes e mais corretos para o projeto.

Para desenvolvimento de código foi necessária contribuição de todo o grupo e junção de conhecimentos e entreajuda. Assim conseguimos alcançar os objetivos necessários para esta fase.

Concluindo, foi realizado um bom trabalho de grupo e com boa cooperação, continuamos motivados e com bons resultados devido à nossa organização. Cada vez mais conseguimos lidar melhor com as advertências atuais resultando a um melhor desempenho.

Referências

1. Slides da aula teórica de Redes de Computadores I; Nicolau Pinto, Maria João.
2. https://sites.google.com/site/redesdecomunicacaonivel3/home/redes-de-comunicacao-1/tecnicas-de-deteccao-e-correcao-de-erros-em-transmissoes-digitais/crc---cyclic-redundancy-check ; *Cyclic Redundancy Check,* Ana Paula Santos.
3. https://github.com/hideakitai/CRCx , Hideaki Tai.

Autoavaliação

Inês Barreira Marques: Na fase 3 do projeto ajudei na pesquisa teórica e na escrita do relatório, bem como na discussão de várias questões que surgiram na realização desta fase.

José Pedro Fernandes Peleja: Nesta fase fiquei encarregue de desenvolver o relatório e realizar a pesquisa dos protocolos necessários. Ajudei na divisão e desenvolvimento de código.

Rui Filipe Ribeiro Freitas: Para esta fase contribui na elaboração do código assim como na realização dos testes. Ajudei também na pesquisa e escrita do relatório bem como na sua revisão.

Tiago João Pereira Ferreira: Contribui no desenvolvimento dos vários códigos e estive presente na realização dos testes. Para além disso ajudei na elaboração do relatório.