

Synchronization for ad-hoc multi-hop networks of mobile agents

Diogo Filipe Santos Almeida

Mestrado em Engenharia de Redes e Sistemas Informáticos

Departamento de Ciência de Computadores

2023

Orientador

Pedro Miranda de Andrade de Albuquerque d'Orey, Professor Auxiliar Convidado,

Departamento de Ciência de Computadores

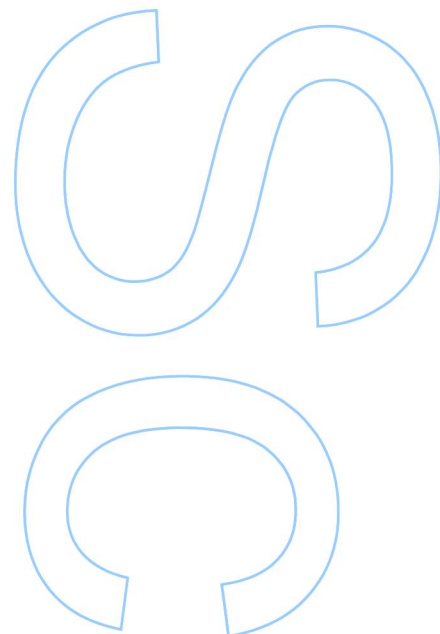
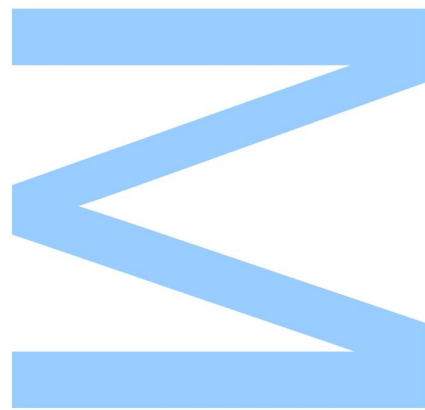
Faculdade de Ciências da Universidade do Porto

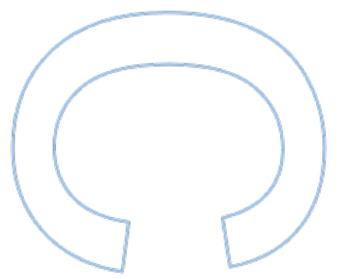
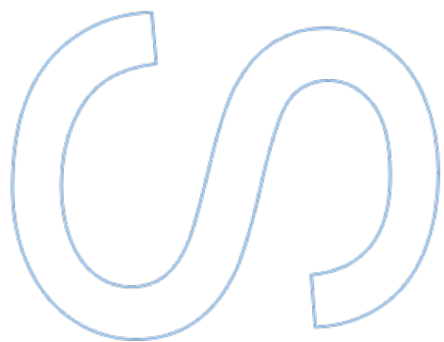
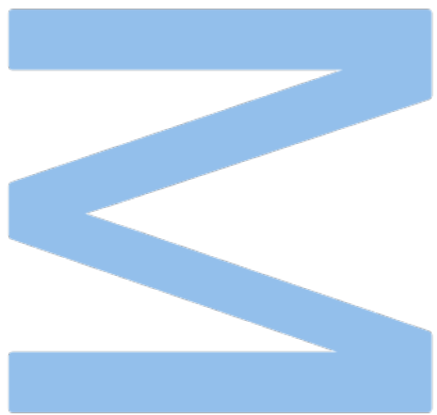
Coorientador

Luis Miguel Pinho de Almeida, Professor Associado com Agregação,

Departamento de Engenharia Eletrotécnica e de Computadores

Faculdade de Engenharia da Universidade do Porto





Abstract

Over the years, teams of Unmanned Aerial Vehicles (UAVs) have become a critical component in surveillance, monitoring, and rescue operations. However, they frequently operate in non-infrastructure areas. The use of ad hoc networks has become a viable solution to this problem. For communication, it is common to use the IEEE 802.11 protocol with Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) as the medium access control. However, its performance seriously degrades with an increasing number of medium accesses and in networks with high bandwidth requirements, leading to more collisions, packet loss, and delays. One option to improve performance is the implementation of a Time Division Multiple Access (TDMA) layer on top of CSMA/CA. This is typically implemented with a global clock, but in networks with significant topology variations, clock synchronization achieves poor precision. This dissertation covers the state of the art of overlay TDMA-based protocols with local synchronization. Subsequently, we implemented two such protocols, namely Reconfigurable and Adaptive Time Division Multiple Access (RA-TDMA) and DESYNC, in a modular TDMA architecture. Finally, we carry out an intensive comparison between them.

Resumo

Ao longo dos anos equipas de *Unmanned Aerial Vehicles* (UAVs) tornaram-se uma peça fundamental em operações de vigilância, monitorização e resgate. No entanto, operam frequentemente em áreas desprovidas de infraestrutura. O uso de redes *ad hoc* tornou-se uma solução viável para este problema. Para a realização da comunicação é comum o uso do protocolo IEEE 802.11 com o *Carrier Sense Multiple Access with Collision Avoidance* CSMA/CA como controlo de acesso ao meio. Mas a sua performance decresce seriamente com o aumento de acessos ao meio e em redes com um grande requisito de largura de banda provocando ainda mais colisões, perda de pacotes e atrasos. Uma das opções para melhorar a performance é a implementação de uma camada *Time Division Multiple Access* (TDMA) sobreposta ao protocolo MAC CSMA/CA. A camada TDMA é tipicamente implementada com um relógio global, mas em redes com grandes variações de topologia a sincronização de relógio não atinge mais do que uma precisão pobre. Assim, esta dissertação começa com um estudo do estado da arte de protocolos de sobreposição baseados em TDMA com sincronização local. Posteriormente procedeu-se à implementação dos protocolos *Reconfigurable and Adaptive Time Division Multiple Access* (RA-TDMA) e DESYNC numa arquitetura TDMA modular. Por fim realizou-se uma comparação intensiva entre eles.

Agradecimentos

Em primeiro lugar gostaria de agradecer a toda a equipa DaRTES pelo o apoio e conhecimento transmitido ao longo destes últimos anos, especialmente aos Professores Pedro d'Orey, Luis Almeida e Luis Pinto. Em seguida gostaria de agradecer à minha família, especialmente aos meus pais e ao meu irmão, por me terem dado a oportunidade de estudar e de me terem incentivado e ajudado ao longo de toda a minha jornada académica. Por fim gostaria também de agradecer aos meus amigos, Inês, Patrícia e Énio, pelo vosso constante apoio, paciência e por terem estado sempre presentes nos momentos mais difíceis.

Dedico este trabalho aos meus pais e ao meu irmão

Conteúdo

Abstract	i
Resumo	iii
Agradecimentos	v
Conteúdo	ix
Lista de Figuras	xii
Lista de Blocos de Código	xiii
Acrónimos	xv
1 Introdução	1
1.1 Contribuições	2
1.2 Organização	2
2 Conceitos Fundamentais	3
2.1 Pilha Protocolar	3
2.2 IEEE 802.11 (WiFi)	4
2.2.1 Camada Física (PHY)	4
2.2.2 Camada de Acesso ao Meio (MAC)	5
2.3 TDMA	6
3 Sincronização sem relógio	9

3.1	RA-TDMA	9
3.1.1	RA-TDMA	9
3.1.2	RA-TDMA+	11
3.1.3	RA-TDMAs	14
3.1.4	RA-TDMAs+	16
3.2	DESYNC	17
3.2.1	Algoritmo	17
3.2.2	Multi-hop	18
3.3	Comparação qualitativa	20
4	Desenvolvimento	21
4.1	Drone-RK	21
4.1.1	Módulo TDMA	22
4.1.2	Encapsulamento dos pacotes	23
4.2	Nova Arquitetura TDMA	23
4.2.1	Comunicação entre módulos	24
4.2.2	<i>TDMA_Core</i>	26
4.2.3	<i>Tracking</i> da topologia	27
4.2.4	Sincronização	29
5	Experiências e Resultados	33
5.1	Métricas	33
5.1.1	Erro de sincronização	33
5.1.2	Período Efetivo	34
5.1.3	Perda de Pacotes	34
5.2	Setup	34
5.3	Experiências	36
5.4	Resultados	38

6	Conclusões	43
	Referências	45

Lista de Figuras

2.1	Diferentes arquiteturas para redes WLAN. Na figura da esquerda temos a arquitetura com infraestrutura (BSS) e na direita uma arquitetura sem infraestrutura (IBSS ou <i>ad hoc</i>)	5
2.2	TDMA	7
3.1	Processo de escolha do <i>delay</i> a usar e respectivo atraso da próxima transmissão [1]	10
3.2	Atualização da matriz após entrada do nó 4 na rede	12
3.3	Métrica A_{Φ}	13
3.4	Topologia usada na experiência	16
3.5	Algoritmo DESYNC - <i>Desynchronization</i> do ponto de vista do nó B [3]	18
3.6	Evolução da fase ao longo do tempo numa topologia cíclica com 4 nós. Cada figura representa uma configuração inicial diferente [2]	18
3.7	Exemplo adaptado do algoritmo Extended DESYNC numa topologia com três nós [5]	19
4.1	Módulos e dependências Drone-RK [8]	22
4.2	Envio de um pacote	22
4.3	Receção de um pacote	23
4.4	Pacote enviado na rede	23
4.5	Arquitetura da <i>framework</i> TDMA	24
4.6	<i>Threads</i> presentes no <i>TDMA_Core</i>	27
4.7	Estrutura do pacote Matrix enviado	28
4.8	Etapas presentes no módulo RA-TDMAs+	30

4.9	Estrutura do pacote DESYNC enviado	30
4.10	<i>Threads</i> presentes no módulo DESYNC	32
5.1	Setup usado nas experiências	35
5.2	Configuração número 1	37
5.3	Configuração número 2	37
5.4	Configuração número 3	38
5.5	Resultados Erro de Sincronização para a configuração número 1, figura 5.2 . . .	39
5.6	Resultados Erro de Sincronização para a configuração número 2, figura 5.3 . . .	39
5.7	Resultados Erro de Sincronização para a configuração número 3, figura 5.4 . . .	39
5.8	Diferentes organizações de <i>slots</i> para a configuração número 3	40
5.9	Resultados Período Efetivo para a configuração número 1, figura 5.2	40
5.10	Resultados Período Efetivo para a configuração número 2, figura 5.3	41
5.11	Resultados Período Efetivo para a configuração número 3, figura 5.4	41
5.12	Resultados Perda de Pacotes para a configuração número 1, figura 5.2	42
5.13	Resultados Perda de Pacotes para a configuração número 2, figura 5.3	42
5.14	Resultados Perda de Pacotes para a configuração número 3, figura 5.4	42

Lista de Blocos de Código

4.1	Variáveis com os <i>delays</i> guardadas na <i>Shared Memory</i>	25
4.2	Variáveis com a informação TDMA na <i>Shared Memory</i>	25
4.3	Variáveis guardadas na <i>Shared Memory</i>	25
4.4	Estrutura do evento <i>event_slot_limits</i> enviado na <i>Socket AF_UNIX</i>	26
4.5	Estrutura do evento <i>event_slot_id</i> enviado na <i>Socket AF_UNIX</i>	26
4.6	Variáveis usadas para guardar a informação da matriz	27
4.7	Variáveis guardadas na <i>Shared Memory</i> relativa a <i>Spanning Tree</i>	29
5.1	<i>Script</i> de inicialização do nó 1	35
5.2	<i>Script</i> usado para correr as experiências	36

Acrónimos

CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance	BS	Base Station
TCP	Transmission Control Protocol	AP	Access Point
TDMA	Time Division Multiple Access	BSS	Basic Service Set
UAV	Unmanned Aerial Vehicle	IBSS	Independent Basic Service Set
UDP	User Datagram Protocol	PCF	Point Coordination Function
OSI	Open Systems Interconnection	DCF	Distributed Coordination Function
P2P	Peer-to-Peer	DIFS	Distributed Inter-Frame Space
HTTP	Hypertext Transfer Protocol	CW	Contention Window
FTP	File Transfer Protocol	ARQ	Automatic Repeat Request
DNS	Domain Network System	CRC	Cyclic Redundancy Check
RIP	Routing Information Protocol	SIFS	Short Inter-Frame Spacing
OSPF	Open Shortest Path First	RTS/CTS	Request to Send / Clear to Send
NIC	Network Interface Controller	GNSS	Global Navigation Satellite Systems
LLC	Logical Link Control	NTP	Network Time Protocol
MAC	Medium Access Control	PDR	Packet Delivery Ratio
WLAN	Wireless Local Area Network	ICMP	Internet Control Message Protocol

Capítulo 1

Introdução

Ao longo dos anos os *Unmanned Aerial Vehicles* (UAVs) tornaram-se uma peça fundamental em operações de vigilância, monitorização e resgate. Operações delicadas e de acesso complicado como por exemplo fogos florestais ou áreas contaminadas podem ser realizadas com o uso de UAVs facilitando o trabalho a, e protegendo, operadores humanos. Geralmente os UAVs são usados em equipas, comunicando entre si para atingir um determinado objetivo global. Contudo, os locais de operação são frequentemente desprovidos de infraestruturas de comunicação. As redes *ad hoc* tornaram-se uma solução fiável e de baixo custo para este problema, no entanto o seu uso apresenta diversos desafios ainda por resolver.

Um dos protocolos de comunicação mais usado nas comunicações entre os UAVs é o protocolo IEEE 802.11 (ou WiFi). Este permite a comunicação em redes sem fios (WLANs) através de sinais de radiofrequência. O protocolo de controlo de acesso ao meio (MAC) predefinido no padrão IEEE 802.11 é o *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA). Mas a sua performance decresce seriamente com o aumento de acessos ao meio e em redes com um grande requisito de largura de banda, por exemplo *streaming* de vídeo, provocando ainda mais colisões, perda de pacotes e atrasos [14]. Uma das opções para melhorar a eficiência do protocolo referido é a implementação de uma camada *Time Division Multiple Access* (TDMA) por cima do MAC CSMA/CA, juntando assim as vantagens oferecidas por ambos. O TDMA atribui a cada elemento da rede um intervalo de tempo específico para a transmissão de dados em modo exclusivo, eliminando assim as colisões no meio. Diversos algoritmos, tais como o TRAMA [10] ou o Z-MAC [12], surgiram com melhorias à tradicional implementação do protocolo TDMA. No entanto, todas estas versões fazem uso de um relógio global para a realização da sincronização o que se pode tornar bastante desafiador em redes *ad hoc* com elevada mobilidade dos elementos presentes na rede. Os protocolos *Reconfigurable and Adaptive Time Division Multiple Access* (RA-TDMA) [7] e DESYNC [3] apresentam-se como versões descentralizadas de TDMA que fazem apenas uso dos tempos de receção de pacotes para realizar a sincronização. O principal objetivo desta tese é a implementação das versões mais atualizadas dos protocolos RA-TDMA e DESYNC e a realização de uma exaustiva comparação entre eles.

1.1 Contribuições

Esta dissertação apresenta as seguintes contribuições de desenvolvimento e análise. Por um lado, a implementação dos protocolos RA-TDMA e DESYNC de forma consistente e na mesma rede *ad hoc* é uma contribuição significativa que oferece uma plataforma para comparações entre esses protocolos. A comparação efetuada é também uma contribuição para o entendimento das respectivas características e operação. Por outro lado, também se realizaram algumas melhorias ao simulador Drone-RK utilizado como base para os desenvolvimentos feitos no âmbito desta dissertação. Neste simulador implementámos uma arquitetura TDMA de sobreposição e modular que permite o uso de diferentes módulos e a sua reconfiguração em *runtime*. Neste momento foram apenas desenvolvidos os módulos de sincronização, nomeadamente RA-TDMA e DESYNC, e um módulo de seguimento de topologia. Por fim, foram implementados alguns mecanismos, não observados nem discutidos na literatura, para estabilizar a sincronização efetuada pelo o protocolo DESYNC em caso de perda de pacotes.

1.2 Organização

Este documento está organizado em 6 capítulos para além deste capítulo introdutório. No capítulo 2 introduzimos sucintamente o protocolo de redes sem fios IEEE 802.11 assim como os protocolos de acesso ao meio CSMA/CA e TDMA. No capítulo 3 descrevemos em detalhe os protocolos RA-TDMA e DESYNC e a sua evolução ao longo dos anos. Posteriormente descrevemos a arquitetura desenvolvida e a sua implementação no capítulo 4. Por fim, no capítulo 5 explicamos em detalhe as experiências realizadas, as métricas usadas na comparação e os resultados obtidos terminando com a conclusão no capítulo 6.

Capítulo 2

Conceitos Fundamentais

Este capítulo apresenta uma introdução à pilha protocolar e ao protocolo IEEE 802.11, também conhecido como WiFi (ou Wi-Fi), abordando as suas principais características e diferentes protocolos de acesso ao meio relacionados.

2.1 Pilha Protocolar

A pilha protocolar é uma organização em camadas de todas as etapas necessárias para a realização de comunicações em rede. No ano de 1983 surgiu o modelo **Open Systems Interconnection** (OSI), sendo posteriormente adotado como *standard* em 1984. Este divide a pilha protocolar em sete camadas:

- **7 Aplicação** - Esta camada é responsável pelas aplicações que comunicam sobre a rede, tais como, o *e-mail*, a partilha de ficheiros *Peer-to-Peer* (P2P), a voz sobre IP e o *streaming* de vídeo. Alguns dos protocolos usados nesta camada são *Hypertext Transfer Protocol* (HTTP), *File Transfer Protocol* (FTP) e *Domain Network System* (DNS).
- **6 Apresentação** - Esta camada tem como função tornar a informação legível à camada de Aplicação realizando o *encode / decode*, compressão de informação e encriptação / desencriptação.
- **5 Sessão** - É responsável pela criação, autenticação e manutenção das sessões de comunicação entre dispositivos. Inclui ainda mecanismos de sincronização, com o uso de *checkpoints*, na transferência de informação.
- **4 Transporte** - Existem dois protocolos, *Transmission Control Protocol* (TCP) e *User Datagram Protocol* (UDP) presentes na camada de transporte. Estes, são responsáveis pela transmissão de informação entre aplicações a correr nos terminais. A informação antes de ser enviada é dividida em segmentos, no caso do UDP, são denominados *datagrams*. O TCP é um protocolo baseado em conexões com garantias de entrega ordenada de segmentos.

Apresenta ainda controlo de congestionamento e controlo de fluxo. Por outro lado, o UDP é um protocolo simples sem conexões nem garantias de entrega de *datagrams*.

- **3 Rede** - A camada de rede é responsável pelo endereçamento lógico e o roteamento de pacotes. Nela é adicionado um cabeçalho IP aos pacotes oriundos da camada de transporte com diversas informações tais como o endereço de origem, o endereço de destino e a versão (ipv4 ou ipv6). Nesta camada os pacotes são denominados *datagrams*. Para além de adicionar o cabeçalho IP também identifica o nó para onde o pacote deve ser enviado de acordo com a tabela de roteamento criada manualmente ou por protocolos tais como o *Routing Information Protocol* (RIP) ou o *Open Shortest Path First* (OSPF).
- **2 Ligação de dados** - A camada de ligação de dados é responsável pela transmissão de *frames* entre nós adjacentes. Esta geralmente é implementada em *hardware* no *network interface controller* (NIC) e divide-se em duas sub camadas *Logical Link Control* (LLC) e *Medium Access Control* (MAC). A LLC contém mecanismos de controlo do link tais como deteção e correção de erros e controlo de fluxo. O MAC especifica os protocolos a serem usados para minimizar as colisões em transmissões simultâneas de diferentes nós. Esses protocolos podem ser divididos em três categorias, *channel partitioning protocols*, *random access protocols* e *token-passing protocols*.
- **1 Física** - No fim da pilha protocolar temos a camada física responsável pela transmissão física dos dados. Recebe os *frames* da camada superior e converte-os em sequências de 0s e 1s. Após a conversão os dados podem ser transmitidos por diferentes meios, tais como a luz (fibra ótica) ou ondas de rádio (redes *wireless*).

Atualmente é comum dividir a pilha protocolar em apenas 5 camadas seguindo a pilha protocolar TCP/IP: Aplicação, Transporte, Rede, Ligação de dados e Física. As camadas Apresentação e Sessão, se necessárias, são implementadas na camada de Aplicação.

2.2 IEEE 802.11 (WiFi)

IEEE 802.11 (WiFi) é uma das normas de comunicação sem fios mais popular, sendo usada globalmente em diversos ambientes. Esta norma especifica a camada física (PHY) e a camada de controlo de acesso ao meio (MAC) em redes locais (WLAN). Ao longo dos anos foram emergindo várias normas tais como o 802.11n (WiFi-4), 802.11ac (WiFi-5) e 802.11ax (WiFi-6) com diferentes capacidades e especificações maioritariamente na camada física.

2.2.1 Camada Física (PHY)

A camada física é a responsável pela transmissão física dos dados. O protocolo IEEE 802.11 opera essencialmente em dois intervalos de frequência, 2.4 - 2.485 GHz (2.4 GHz) e 5.1 - 5.8 GHz (5 GHz). Com o surgimento do WiFi-6E, uma extensão do WiFi-6, começou-se também a

utilizar os 6 GHz (5.9 - 7.1 GHz nos Estados Unidos da América e 5.9 - 6.4 GHz na Europa). Quanto maior for a frequência usada maior é a largura de banda e menor é o alcance do sinal.

Os componentes principais da arquitetura do protocolo IEEE 802.11 são as estações *wireless* denominadas por *Base Station* (BS) e os *Access Point* (AP). Estas podem ser organizadas em duas topologias diferentes, como representado na figura 2.1. Temos a topologia com infraestrutura denominada **Basic Service Set (BSS)** onde diversas BS estão conectados a um AP que fornece a ligação entre as BS e outras redes, quer *wireless*, i.e., outros BSS, quer cabladas como IEEE 802.3 (Ethernet) ou outras. Os AP estruturam a rede e na sua presença dizemos que a rede é infraestruturada. Quando as BS comunicam entre si sem recurso a um AP, i.e., sem uma infraestrutura pré-instalada, temos um **Independent BSS (IBSS)** também conhecido por rede *ad hoc*.

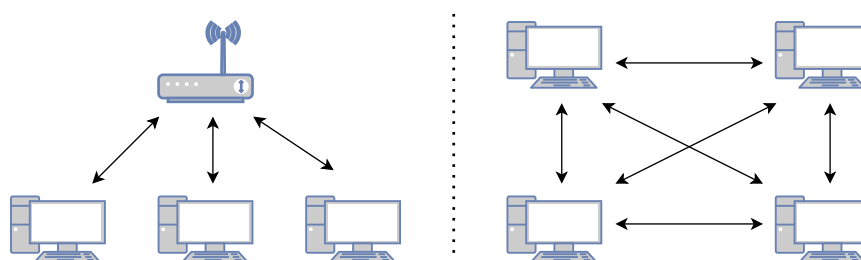


Figura 2.1: Diferentes arquiteturas para redes WLAN. Na figura da esquerda temos a arquitetura com infraestrutura (BSS) e na direita uma arquitetura sem infraestrutura (IBSS ou *ad hoc*)

Outra distinção bastante relevante em redes *wireless* é *Single-Hop* e *Multi-Hop*. Esta distinção é feita em topologias com e sem infraestrutura. A diferença baseia-se no número de dispositivos que um pacote necessita de cruzar em redes *wireless* até chegar ao destino. *Single-Hop* é quando o pacote vai direto ao destino e *Multi-Hop* quando necessita de atravessar vários dispositivos até ao destino final, ou seja, quando o pacote é reencaminhado.

2.2.2 Camada de Acesso ao Meio (MAC)

Por o meio *wireless* ser um canal único de difusão partilhado (*half-duplex*) é necessário um mecanismo de controlo de acesso ao meio (MAC). Ao contrário de redes com fios, os transmissores WiFi são incapazes de detetar colisões no meio durante o envio de um *frame* e, por isso, tentam evitá-las em vez de as detetar. Para evitar transmissões simultâneas o IEEE 802.11 implementa o protocolo *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA). Este inclui diversos métodos de coordenação que podem ser usados para evitar colisões. Dois desses métodos são o *Point Coordination Function* (PCF), apenas para redes com infraestrutura, e o *Distributed Coordination Function* (DCF), para redes com e sem infraestrutura definida. O DCF é o método *standard* definido no protocolo IEEE 802.11.

Quando o transmissor necessita de enviar um *frame*, primeiro verifica se o meio está livre. Caso esteja transmite-o após um período de tempo denominado *Distributed Inter-frame Space*

(DIFS), que é o intervalo de tempo de inatividade necessário para considerar o meio livre. Caso o meio esteja ocupado, o transmissor necessita de esperar até ao final da transmissão atual mais o intervalo de tempo DIFS. Posteriormente, e antes de transmitir o *frame*, é escolhido um número aleatório de *slots* de *backoff* através do método *binary exponential backoff*. Cada *slot* de *backoff* têm um tamanho fixo, determinado pela norma IEEE 802.11, e o número de *slots* escolhidas é limitado pela *Contention Window* (CW). Após DIFS é iniciada a contagem decrescente de *slots*, sendo que o valor é decrementado quando se verifica que o meio está livre e permanece inalterado quando o meio está ocupado. Quando o valor chega a zero o *frame* é transmitido. A cada transmissão efetuada sem sucesso a CW é duplicada em relação ao anterior valor e o processo de envio é repetido.

Existem diversos métodos de deteção do sucesso ou não de uma transmissão. O protocolo IEEE 802.11 implementa um mecanismo de *feedback* e controlo de erros, denominado *Automatic Repeat Request* (ARQ). Quando um *frame* (pacote) é recebido é sujeito a um teste de deteção de erros denominado *Cyclic Redundancy Check* (CRC). Após passar o CRC com sucesso o recetor envia um *acknowledgment* (ACK) após um período de tempo de espera, *Short Inter-frame Spacing* (SIFS). Do outro lado o transmissor espera um período de tempo SIFS antes de ficar a espera do ACK. Caso não o receba durante um intervalo de tempo denominado *ACK Timeout*, assume que o *frame* foi perdido ou corrompido e retransmite-o. Se o ACK não for recebido após um número fixo de retransmissões o transmissor descarta o *frame*. Outro mecanismo adicional, que pode ou não ser usado, para averiguar o sucesso de uma transmissão é o *Request to Send and Clear to Send* (RTS/CTS). Neste método, os *frames* RTS e CTS são usados para reservar o meio durante a transmissão. O transmissor envia um *frame* RTS e o recetor, após SIFS, responde com o CTS. Após esta troca de *frames* o transmissor já pode realizar a transmissão do *frame* de dados. No final da transmissão o recetor responde com um ACK. Este método é particularmente eficaz contra o problema do terminal escondido.

Em [14] os autores observaram que, quando sujeitos a uma grande quantidade de tráfego, existe um maior *throughput*, atrasos inferiores e uma maior robustez relativamente a alterações no tamanho da CW no método RTS/CTS em relação ao ACK. No entanto, concluiu-se que em redes com grande requisito de largura de banda, como por exemplo *streaming* de vídeo, e com um elevado número de acesso ao meio a performance do protocolo CSMA/CA decresce seriamente provocando ainda mais colisões, perda de pacotes e atrasos. Uma possível solução é a implementação de uma camada *Time Division Multiple Access* (TDMA) em cima da camada CSMA/CA.

2.3 TDMA

Time Division Multiple Access (TDMA) é outro protocolo de controlo de acesso múltiplo a um meio de transmissão partilhado. O TDMA permite que vários dispositivos (nós) partilhem o mesmo canal de frequência, atribuindo a cada um um intervalo de tempo exclusivo para envio de dados. Esse intervalo de tempo é designado por *slot* e à sequência de *slots* atribuídas ao conjunto

dos dispositivos dá-se o nome de *round*, sendo esta repetida infinitamente com um dado período (*round period*). A Fig. 2.2 representa uma ronda com N dispositivos pertencentes à rede.

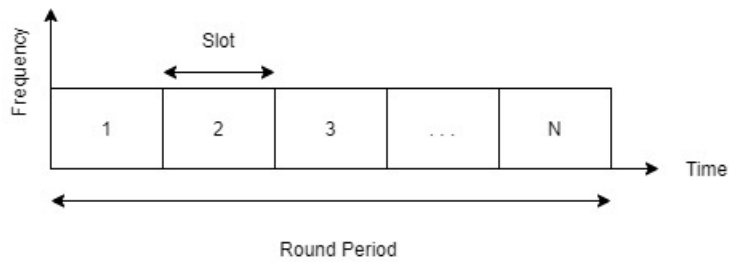


Figura 2.2: TDMA

De notar que o TDMA pode ser implementado por algumas redes diretamente sobre a camada física ou, de forma mais comum, como uma camada de sobreposição que implementa a estrutura das *slots* sobre um protocolo MAC de outro tipo, por exemplo, o CSMA/CA. O TDMA faz uso de um relógio global para a realização da sincronização. Global Navigation Satellite Systems (GNSS) ou Network Time Protocol (NTP) são algumas das referências temporais externas que podem ser usadas. No entanto, em redes *ad hoc* dinâmicas, devido a constante mobilidade dos elementos presentes, o uso de um relógio global atinge uma precisão baixa de sincronização [11]. Uma possível solução para este problema é o uso de protocolos baseados em sincronização local, ou seja, sem o uso de um relógio global.

Capítulo 3

Sincronização sem relógio

Os protocolos MAC do tipo TDMA requerem que todos os nós da rede tenham uma percepção comum do tempo, para que comecem as suas *slots* nos instantes corretos obtendo assim uma boa sequência no *round*. Contudo, fazer sincronização de relógio sobre redes *wireless ad hoc* é normalmente pouco eficiente levando a uma precisão pobre. Uma alternativa é os nós sincronizarem-se escutando as transmissões uns dos outros sem necessitarem de um relógio global. Este capítulo apresenta dois protocolos TDMA deste tipo, desenvolvidos para funcionar em sobreposição sobre uma camada MAC (CSMA/CA) de IEEE 802.11, nomeadamente o *Reconfigurable and Adaptive TDMA* (RA-TDMA) e o DESYNC-TDMA. Começamos por abordar os seus algoritmos e a sua evolução ao longo dos anos realizando posteriormente uma pequena comparação qualitativa dos mesmos.

3.1 RA-TDMA

Reconfigurable and Adaptive TDMA (RA-TDMA) é um protocolo que permite adaptar as *slots* e reconfigurar o *round* de acordo com as condições e número atual de utilizadores no meio de comunicação. Ao longo dos anos foram surgindo diversas versões deste protocolo. A primeira foi designada simplesmente como RA-TDMA [1] com o objetivo de ser aplicada em redes estruturadas. Posteriormente desenvolveu-se o RA-TDMA+ [7] para redes *mesh ad hoc* e depois o RA-TDMAs [9] para *streaming* de vídeo sobre redes de *relays*. Todas estas versões tem como objetivo a partilha de estado entre nós com a transmissão periódica de um pacote e a realização da sincronização baseando-se apenas no atraso de receção de pacotes. Isto permite configurar o *round* com as *slots* que podem depois ser usadas para transmitir dados.

3.1.1 RA-TDMA

Em [1] surgiu o RA-TDMA para redes estruturadas. Numa rede com N nós, com topologia em estrela, é escolhido um nó como referência de sincronização situado na primeira *slot* da ronda.

Os restantes $N-1$ nós sincronizam em relação ao nó referência. Cada nó transmite um pacote em *broadcast* no início da sua *slot* no tempo $t_{i,now}$. O próximo tempo de transmissão é definido como $t_{i,next} = t_{i,now} + T$, sendo T a duração da ronda TDMA. Ao longo da ronda o nó referência escuta todos os pacotes enviados pelos outros e calcula o *delay* δ dos mesmos, diferença entre o tempo efetivo de chegada e o tempo esperado de chegada. Antes da transmissão em $t_{0,next}$ o nó referência escolhe o maior valor de *delay* observado e atrasa a sua transmissão como representado na equação 3.1. Esta adaptação é limitada a um valor máximo de atraso, $\Delta_N = \frac{T}{N} \times \varepsilon$, onde $0 < \varepsilon < 1$.

$$t_{0,next} = t_{0,now} + T + \max(\delta_j)_{j=1..N-1, 0 \leq \delta_j \leq \Delta_N} \quad (3.1)$$

Na figura 3.1 é possível observar este processo onde o nó 0, nó de referência, após escutar todos os pacotes e calcular os respetivos *delays*, atrasa a sua transmissão de acordo com o maior *delay* observado ao longo da ronda, o *delay* δ_2 .

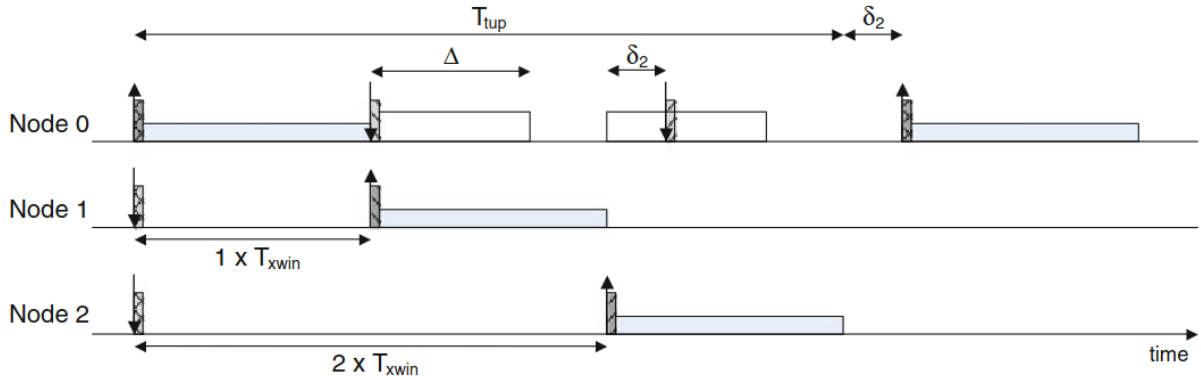


Figura 3.1: Processo de escolha do *delay* a usar e respetivo atraso da próxima transmissão [1]

Os restantes nós na rede, após receberem o pacote do nó de referencia calculam o respetivo *delay* e ajustam a sua próxima transmissão com base no *delay* obtido, como representado na equação 3.2.

$$t_{i,next} = t_0 + i \times \frac{T}{N}, \quad i = 1..N - 1 \quad (3.2)$$

Numa rede dinâmica há entradas e saídas de nós da rede, provocando constantes reestruturações na ronda TDMA. Estas provocam alterações no número e tamanho das *slots*, o que impossibilita pré-definir a atribuição de nós às *slots*. Como esta atribuição têm de ser feita em *runtime* um dos desafios é manter a consistência entre todos os elementos presentes na rede. Uma das diversas abordagens de o conseguir é colocar todos os IDs conhecidos por ordem crescente antes de atribuir às *slots*. Numa topologia com três nós com IDs iguais a $\{5, 10, 1\}$ a atribuição de *slots* seria igual à *slot* número 0 igual ao ID 1, *slot* número 1 igual ao ID 5 e por fim *slot* número 2 igual ao ID 10. Para todos os nós obterem a mesma perceção da rede, o número ativo de nós e os seus IDs, é necessário enviar no pacote um vetor de *membership* com a informação local. Após todos os

nós obterem a mesma visão da rede é possível dividir a ronda TDMA em *slots* iguais mantendo assim a consistência entre todos os nós.

3.1.2 RA-TDMA+

Com o intuito de suportar colaboração em tempo real entre robôs de uma equipa sem uma infraestrutura surgiu o protocolo RA-TDMA+. Este protocolo permite a realização da sincronização em redes *mesh ad hoc* sem a necessidade de um relógio global. Cada nó, pertencente a rede, é identificado por um ID único. A cada ID é associada uma *slot* para uso exclusivo, t_{slot} com duração igual a $\frac{T}{N}$, onde T representa o tempo de duração da ronda TDMA e N o número total de nós pertencente à rede num dado instante. Por ser uma rede *mesh* dinâmica poderão existir nós sem ligação direta a todos os outros nós pertencentes à rede. Para contornar este problema e todos os nós obterem a mesma visão global da rede é necessário realizar a partilha de estado. O estado corresponde à informação que um nó contém sobre a rede num dado instante, como por exemplo a topologia da rede e os seus vizinhos diretos. Para a partilha de estado é usado *flooding with aggregation*. No início de cada *slot*, cada nó partilha a sua visão da rede com os membros da equipa em *multicast*. Na receção desses pacotes o recetor usa a informação do transmissor para atualizar a sua visão da rede. Após um número finito de rondas todos os nós pertencentes à equipa irão conter a mesma visão global da rede.

Tracking da topologia. A topologia de rede é representada por uma matriz binária M onde $M(j, k) = 1$ representa a existência de link direto e $M(j, k) = 0$ a ausência de link entre dois nós. O nó j após receber um pacote com o estado de outro nó verifica linha a linha se a informação é mais recente ou não. Esta verificação é realizada com o uso do valor *age* associado a cada linha. Caso seja, a informação dessa linha é substituída, mantendo assim só a informação mais atualizada. Para além de verificar as linhas na matriz relativas aos outros nós, o nó j atualiza a sua própria linha com a existência ou não de link, sendo o único que o pode fazer.

Tracking membership. Cada vez que um nó recebe o estado de outro nó, do qual ele não conhecia, é realizado um processo de integração, como representado na figura 3.2. Inicia-se por atribuir uma posição de *slot* ao nó descoberto, uma das diversas maneiras de o fazer, mantendo a consistência entre todos os nós, é colocar todos os IDs conhecidos por ordem crescente antes de atribuir às *slots*, como descrito na secção 3.1.1. Após descoberto a posição de *slot* do nó, adiciona-se uma linha e coluna na respetiva posição na matriz de adjacências. Posteriormente à atualização o nó recetor necessita de atualizar a sua própria linha adicionando o link ao nó descoberto. Na ronda seguinte esta atualização é disseminada aos outros nós.

relativo à ronda, é representado por um ponto na circunferência com um ângulo θ , sendo que o mesmo ponto representa em rondas diferentes o mesmo tempo relativo.

$$\theta(t) = \frac{2\pi}{T}t \quad (3.5)$$

Sendo ϕ a fase inicial de uma ronda já com o *offset* t_ϕ incluído, podemos representar a fase do início da *slot* i como σ_i :

$$\theta(t_{s_{k,i}}) = \frac{2\pi}{T} \times t_{s_{k,i}} = \frac{2\pi}{T} \times (t_{rk} + i \times \frac{T}{N}) = \phi + i \times \frac{2\pi}{N} = \sigma_i \quad (3.6)$$

O nó i , de acordo com a sua visão local da rede, começa a ronda na fase ϕ_i e a *slot* em s_i . É considerado o estado ideal de sincronização quando $\forall_{i,j} \phi_i = \phi_j$, ou seja, quando todos os nós tem a mesma noção da fase respetiva ao início de ronda. Para quantificar o estado de sincronização usa-se a métrica $Arc(\Phi)$, A_Φ , que representa o intervalo mínimo que contém todas as fases de todos os nós. Quanto menor o intervalo melhor é o estado atual de sincronização. Na figura 3.3a podemos observar dois estados de sincronização. Por ter um valor inferior de A_Φ , o estado representado no lado esquerdo está mais próximo do estado ideal de sincronização do que o do lado direito. Diferentes nós podem apresentar diferentes visões de sincronização da rede, como representado na figura 3.3b.

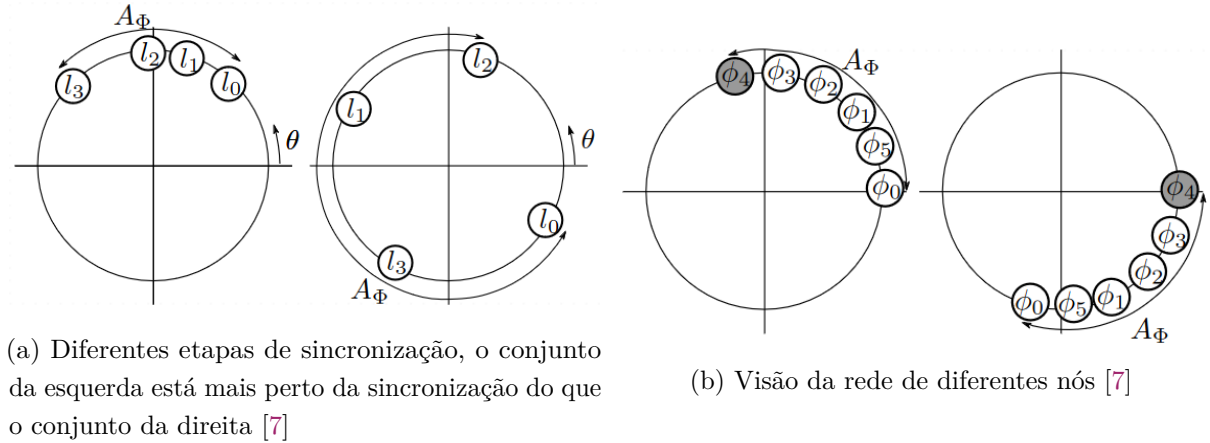


Figura 3.3: Métrica A_Φ

O nó i após receber o pacote enviado pelo nó j , no tempo t_{rx}^j , consegue calcular o início da *slot* de j e respetiva fase do nó com as seguintes equações:

$$t_{s_{k,j}} = t_{rx}^j - t_{len}^j \quad (3.7)$$

$$\sigma_j = \theta(t_{s_{k,j}}) = \phi_j + j \times \frac{2\pi}{N} \quad (3.8)$$

$$\phi_j = \sigma_j - j \times \frac{2\pi}{N} \quad (3.9)$$

onde t_{len}^j é o tempo de transmissão do pacote. Quando chega a vez do nó i transmitir, este já conhece todas as fases dos seus vizinhos, N_i , enviadas na ronda imediatamente anterior. Com esta informação já consegue calcular o *delay* necessário, δ_i , a adicionar à sua próxima transmissão:

$$\delta_i = \max_{k \in N_i \cup i} (\phi_k - \phi_i) \times \frac{T}{2\pi} \quad (3.10)$$

Como usamos a variável Δ para limitar o tempo máximo de atraso da transmissão, a próxima fase de transmissão é calculada com a seguinte equação:

$$\phi_i^{next} = \phi_i^{now} + \min(\Delta, \delta_i) \times \frac{2\pi}{T} \quad (3.11)$$

Problemas: Topologias de rede com *loops* tornam-se desafiadores para a realização com sucesso deste algoritmo, sendo que, em alguns casos, não foi possível obter a convergência na sincronização. Para mitigar este problema usou-se *spanning trees* construídas a partir da matriz de topologia, que permitem limitar o número de nós com os quais é feita a sincronização. No entanto, esta abordagem atrasa o processo de disseminação da informação e consequentemente o tempo até atingir o estado ideal de sincronização.

3.1.3 RA-TDMAs

O protocolo RA-TDMAs foi desenvolvido para suportar *streaming* de vídeo entre *Unmanned Aerial Vehicles* (UAVs) e uma estação base de monitorização. Este protocolo assume uma rede com topologia em linha disposta a transmitir uma grande quantidade de dados que são reencaminhados por n nós (UAV's) antes de chegarem à estação base. Por existir um incremento de *delay* em cada retransmissão, o valor n necessita de ser um valor baixo, segundo os autores, de até 5 nós para evitar *delays* de segundos. A cada nó é previamente associado um ID e respetiva *slot*, durante a qual o nó tem permissão para transmitir na ronda TDMA.

Sincronização Um nó k para realizar a sincronização precisa de calcular o *delay*, δ , de todos os pacotes recebidos. Para o fazer são necessárias duas variáveis enviadas no *header* do pacote, o *slot* ID e o *offset* de transmissão, diferença de tempo entre o envio da mensagem e o início da *slot*. Com estas variáveis o *delay* é calculado com a seguinte formula:

$$\delta_k = (rx_k - r\hat{x}_k + T/2)_T - T/2 \quad (3.12)$$

onde rx_k representa o tempo real de chegada do pacote e $r\hat{x}_k$ o tempo esperado de chegada. Estes tempos são relativos à ronda TDMA. Para conhecer o valor de $r\hat{x}_k$ precisamos de calcular primeiro o tempo esperado de início de *slot* do transmissor do pacote. Supondo que dois nós com IDs de *slots* iguais a j e i estão perfeitamente sincronizados, então o nó j estima que o início da *slot* de i , $\hat{B}^{(i)}$, é dado por:

$$\hat{B}^{(i)} = (B^{(j)} - (j - i)s + T)_T \quad \text{onde } j, i \in [1, n] \quad e \quad s = T/n \quad (3.13)$$

Após descoberto o valor de $\hat{B}^{(i)}$ o nó j já consegue calcular o tempo esperado de chegada do pacote, $r\hat{x}_k$:

$$r\hat{x}_k = (\hat{B}^{(i)} + p_k)_T \quad (3.14)$$

Tendo em conta as equações 3.12 e 3.14, o cálculo do *delay* fica:

$$\delta_k = (rx_k - r\hat{x}_k + T/2)_T - T/2 = (rx_k - (\hat{B}^{(i)} + p_k) + T/2)_T - T/2 \quad (3.15)$$

Antes de um nó começar a transmitir, no início da sua *slot*, este verifica todos os *delays* observados durante a ronda anterior. Caso o nó ache que os pacotes chegaram atrasados, atrasa os limites da sua *slot* (início e fim) começando assim a sua transmissão mais tarde. Em contrapartida, caso o recetor ache que os pacotes chegaram adiantados, mantém a sua *slot* e espera que o transmissor futuramente atualize a sua. Como são analisados os *delays* de diversos pacotes é necessário agregar os mesmos de modo a obter o valor final de desvio de fase. Diversas heurísticas, f , foram testadas:

- **Mínimo** : usa-se o pacote com o menor atraso;
- **Máximo** : usa-se o pacote com o maior atraso;
- **Mediana** : realiza a mediana dos atrasos de todos os pacotes de modo a evitar pacotes isolados com muito ou pouco atraso.

Por fim, o desvio final de transmissão é calculado com a seguinte equação:

$$\Delta(r) = \min(\max(f(\delta_1, \delta_2, \dots), 0), \Delta_{max}) \quad (3.16)$$

onde Δ_{max} é o valor máximo de desvio que se pode ajustar. E o novo início de transmissão do nó j é :

$$B^{(j)}(r) = B^{(j)}(r - 1) + \Delta(r) \quad (3.17)$$

Por ser uma topologia em linha cada nó realiza a sincronização com apenas um nó, sendo este correspondente à *slot* imediatamente anterior à sua. Todos os nós irão adaptar-se até que nenhum atraso seja detetado, atingindo-se assim a convergência.

Testes e resultados Foram realizados testes ao protocolo em laboratório com o uso de AR Drone 2.0 UAV. Criou-se uma topologia em linha, representada na figura 3.4, com três nós mais uma *Base Station* (BS). Todos os nós estavam separados a uma distância de três metros. O *drone* 1, R1, envia a *stream* de vídeo que é retransmitida pelos restantes nós, R2 e R3, até ser recebida pela BS.

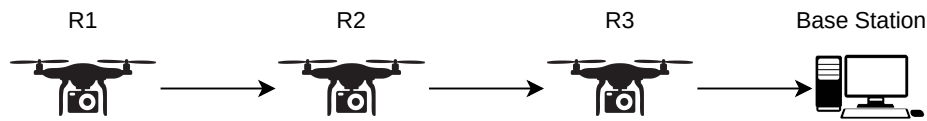


Figura 3.4: Topologia usada na experiência

É enviado um *frame* a cada 130 milissegundos, (7.5 FPS), sendo que cada *frame*, com tamanho igual a 11 kB, é dividido em 73 pacotes UDP, cada um com um *payload* de 154 Bytes. O período da ronda TDMA é de 96 milissegundos e as *slots* de 32 milissegundos de tamanho. O Δ_{max} foi fixado em 8 milissegundos, que corresponde a 25% do tamanho da *slot*. As variáveis necessárias à realização da sincronização são transmitidas no pacote com a seguinte estrutura:

```
uint8_t      slot_id ; // 0 - 254 (one slot per host)
slot_limits_t slot ; // [init end] ms - 16bits
uint8_t      timestamp_ms ; // when we call sendto()
uint8_t      timestamp_part_ms ; // number of parts in 256 - fraction of 1 ms
uint32_t      seq_num ; // unique packet id
```

Foram testadas as diferentes heurísticas (Mínimo, Máximo e Mediana) durante 5 minutos, o correspondente a cerca de 300 rondas TDMA. Observou-se um melhor *throughput* e uma maior estabilidade da ronda TDMA usando o mínimo enquanto que o máximo transmite uma maior confiabilidade (*Packet Delivery Ratio* - PDR). No caso da mediana obteve-se uma performance intermédia mas sempre perto dos piores casos das outras heurísticas.

3.1.4 RA-TDMA+

O mais recente protocolo da família RA-TDMA, o RA-TDMA+ é uma combinação dos protocolos RA-TDMA+ e RA-TDMAs. Do RA-TDMA+, implementado para redes *ad hoc* mas com baixa largura de banda, adotou-se o *tracking* da topologia descrito na secção 3.1.2 e do RA-TDMAs, para topologias em linha com elevada largura de banda, o algoritmo de sincronização, secção 3.1.3. Esta junção permitiu combinar o melhor que ambos os protocolos oferecem, criando assim um

algoritmo robusto em redes *ad hoc* para elevada largura de banda.

3.2 DESYNC

Com base em eventos biológicos, o algoritmo DESYNC apresenta uma nova primitiva de sincronização: a *Desynchronization* [3]. A ideia subjacente a esta primitiva é permitir que os nós da rede realizem as suas tarefas o mais distante possível dos outros nós. Esta pode ser usada para a implementação do protocolo TDMA eliminando a necessidade de usar um relógio global na sincronização e permitindo o ajuste automático dos nós que pertencem à rede, sem comprometer a largura de banda.

3.2.1 Algoritmo

O algoritmo DESYNC baseia-se na *framework* de *pulse-coupled oscillator* apresentada por Mirollo e Strogatz [4]. Nesta *framework* cada nó é representado como um oscilador com frequência $\omega = 1/T$, sendo T o período. A fase é a posição relativa de uma oscilação dentro de um ciclo, sendo representada por $\phi_i(t) \in [0, 1]$ para um dado nó i no tempo t . Quando este atinge $\phi_i(t) = 1$ o nó i "dispara" indicando aos outros nós o fim do seu ciclo. Com esta informação os nós podem adiantar ou atrasar os seus valores de fase conforme o necessário. Para obter um estado de *Desynchronization* é necessário que a diferença de fases entre quaisquer dois nós seja igual a $\frac{1}{n}$, com n a indicar o número total de nós na rede.

No algoritmo DESYNC os "disparos" a indicar o fim do ciclo correspondem ao envio de uma mensagem em *broadcast* para todos os nós. Cada nó guarda o tempo de envio dos "disparos" recebidos. O nó i após obter os valores temporais do "disparo" do nó $i-1$ e do nó $i+1$, ajusta o seu valor de fase para ficar no ponto intermédio entre os nós $i-1$ e $i+1$. Na figura 3.5 é possível observar este processo com o nó B a movimentar o seu valor de fase assim que ambos os seus nós vizinhos, A e C, "disparam". O ponto intermédio do nó i é calculado com a seguinte fórmula:

$$\tilde{\phi}_{mid}(t) = \frac{1}{2} [\tilde{\phi}_{i+1}(t) + \tilde{\phi}_{i-1}(t)] \quad (mod \ 1) \quad (3.18)$$

e o novo valor de fase do nó i :

$$\phi'_i(t) = (1 - \alpha)\phi_i(t) + \alpha\tilde{\phi}_{mid}(t) \quad (3.19)$$

onde $\alpha \in [0, 1]$ é um parâmetro que limita o salto máximo que o nó i pode dar desde a sua posição atual até à nova posição. A constante movimentação dos nós para o ponto intermédio entre os seus vizinhos leva ao estado de *Desynchronization*.

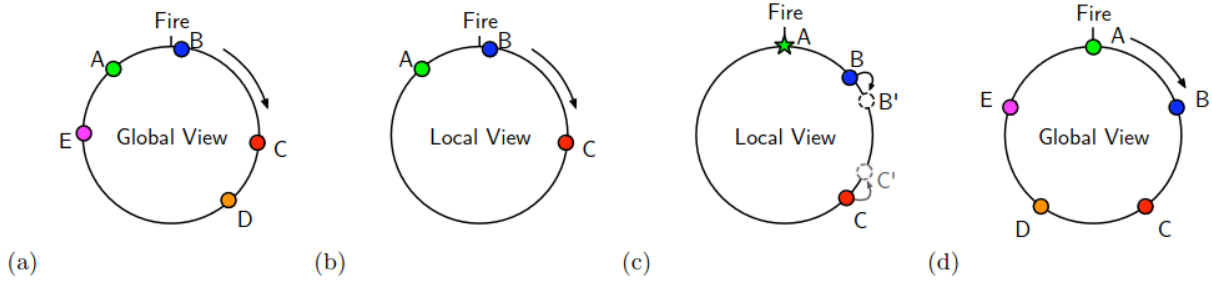


Figura 3.5: Algoritmo DESYNC - *Desynchronization* do ponto de vista do nó B [3]

Este algoritmo é a base do protocolo DESYNC-TDMA que usa os pontos intermédios entre vizinhos como limites de *slot* TDMA. O início da *slot* do nó i é representado como o ponto intermédio entre ϕ_i e ϕ_{i-1} e o fim da *slot* como o ponto intermédio entre ϕ_i e ϕ_{i+1} .

3.2.2 Multi-hop

Em [2] os autores demonstraram que existe sempre pelo menos uma configuração de *Desynchronization* em redes *multi-hop*. Para isso utilizaram coloração de grafos que consiste em atribuir cores aos nós de modo a que nós vizinhos não contenham a mesma cor. Cada cor permite ao nó aceder a um recurso, neste caso a uma *slot* TDMA. Nós com cores iguais acedem ao mesmo tempo à mesma *slot*. Foram realizados testes em diferentes tipos de grafos: *Fully-connected Graph*, *Path Graph*, *Cycle Graph* e *Star Graph*. Em todos os testes foi provada a convergência do algoritmo DESYNC, no entanto, também se verificou que diferentes configurações iniciais podem levar a diferentes estados de *Desynchronization*, como observado na figura 3.6.

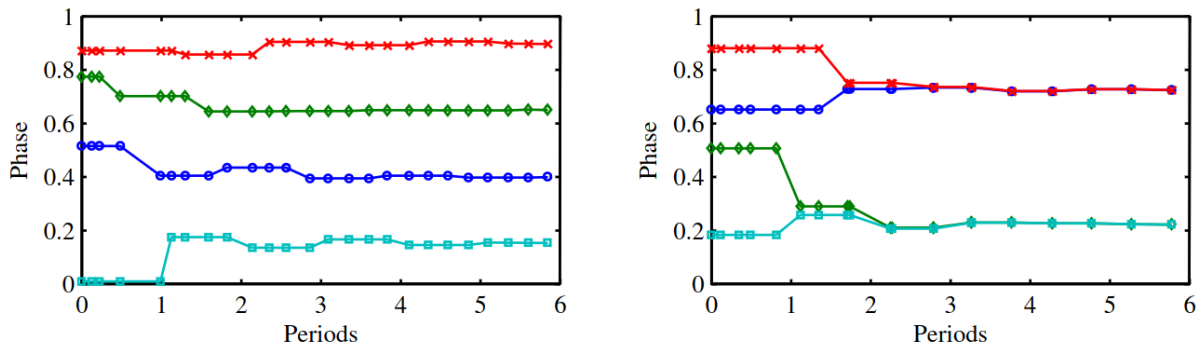


Figura 3.6: Evolução da fase ao longo do tempo numa topologia cíclica com 4 nós. Cada figura representa uma configuração inicial diferente [2]

No entanto, nem todos os estados de *Desynchronization* são ideais para a comunicação entre nós. Um dos problemas existentes em redes sem fios é o *Hidden Terminal* que ocorre quando dois ou mais nós que não conseguem comunicar entre si transmitem simultaneamente dados para o mesmo nó. Esta transmissão simultânea provoca colisões e consequente perda de pacotes. Para contornar este problema e criar o estado de *Desynchronization* ideal à comunicação é necessário

ter em conta os *links* entre os *Hidden Terminals*. Ou seja, todos os nós que estão a uma distância de dois *hops* devem ter um *link* direto no grafo. A este grafo é dado o nome de *Constraint Graph*.

3.2.2.1 Extended DESYNC

Diversos algoritmos, tais como o Extended DESYNC [6] ou o MD-MAC [13], surgiram com implementações funcionais em *multi-hop*. No entanto, decidiu-se aprofundar e implementar o algoritmo Extended DESYNC por ser o que faz uso do *Constraint Graph* mantendo-se assim o mais fiel possível ao algoritmo original. A construção do *Constraint Graph* é realizada através de um processo denominado por *Reciprocal Phase Shift Propagation* (PSP). Este algoritmo consiste na partilha de informação local entre nós. Cada nó necessita de saber os valores relativos da fase de todos os seus vizinhos até dois *hops*. Para o conseguir, o pacote enviado a indicar o fim de ciclo deve conter uma lista com os identificadores e desvios de fases de todos os vizinhos com ligação direta, ou seja a uma distância de um *hop*.

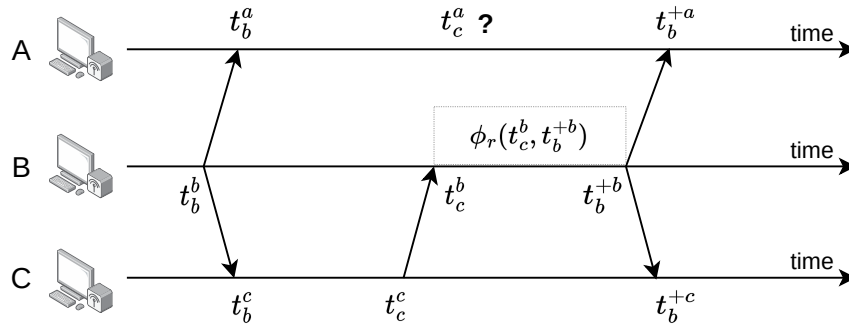


Figura 3.7: Exemplo adaptado do algoritmo Extended DESYNC numa topologia com três nós [5]

Na figura 3.7 podemos observar este processo numa topologia em linha com três nós A-B-C onde os nós A e C são *hidden nodes* e por isso não comunicam diretamente entre si. O nó B envia o seu pacote no tempo t_b^b sendo recebido pelos nós A e C nos tempos t_b^a e t_b^c respetivamente. De seguida o nó C envia o seu pacote t_c^c , mas só o nó B o recebe diretamente em t_c^b . Em t_b^{+b} o nó B volta a enviar o pacote, mas desta vez o pacote já contém o desvio de fase do nó C. Este desvio é calculado com a seguinte fórmula:

$$\phi_r(t_c^b, t_b^{+b}) = t_b^{+b} - t_c^b \quad (3.20)$$

O nó A, que até então desconhecia a existência do nó C, após receber o pacote enviado por B em t_b^{+a} já consegue calcular o tempo de envio do nó C em t_c^a ganhando assim conhecimento dos seus vizinhos a dois *hops*. Este cálculo é realizado com a seguinte fórmula:

$$t_c^a = t_b^{+a} - \phi_r(t_c^b, t_b^{+b}) \quad (3.21)$$

É de realçar que em [5], os autores têm em conta os diversos *delays* de transmissão na realização dos cálculos. Após todos os nós terem conhecimento dos seus vizinhos até dois *hops* já é possível definir com segurança o nó anterior e posterior em fase de cada nó pertencente à rede.

3.3 Comparação qualitativa

Apesar de ambos os algoritmos cumprirem com sucesso a sincronização do protocolo TDMA sem o uso de um relógio global, estes diferem em alguns aspetos. Ambos os protocolos transmitem um pacote com o intuito de realizar a sincronização, no entanto o RA-TDMA, mesmo que o pacote de partilha de estado se perca na rede, realiza a sincronização com todos os restantes pacotes recebidos. Esta característica é vantajosa em relação ao DESYNC pois caso o pacote se perca a sincronização é realizada com informação desatualizada criando alguma instabilidade no algoritmo. O RA-TDMA usa o *flooding with aggregation* para disseminar a visão global de um nó. Em redes de maior dimensão enviar a visão global pode ser prejudicial. Quanto maior o número de nós maior é o tamanho do pacote a enviar e mais recursos computacionais são necessários para armazenar a informação, limitando assim a escalabilidade do protocolo. No entanto a visão global permite complementar o algoritmo de sincronização com outras funcionalidades, por exemplo *routing*. O DESYNC permite uma maior escalabilidade pois usa apenas os nós a uma distância de dois *hops* e só guarda o tempo de chegada dos pacotes. Por outro lado, a implementação de *routing* requer um mecanismo de seguimento de topologia ou de descoberta de rotas. Outro ponto diferenciador é os diferentes estados que se podem obter no DESYNC para a mesma rede. No RA-TDMA obtém-se sempre N *slots* para N nós enquanto que no DESYNC o número de *slots* pode ser inferior ao número de nós sem comprometer a sincronização. Um ponto comum aos dois algoritmos é o uso de Δ_{max} para limitar o desvio que a *slot* pode realizar na adaptação às condições do meio (i.e., aos atrasos da rede).

Capítulo 4

Desenvolvimento

Este capítulo descreve em detalhe a arquitetura desenvolvida. Inicialmente aborda-se o simulador usado, Drone-RK, dando uma pequena introdução aos módulos que este contém. Por o módulo TDMA ser fundamental na nova arquitetura irá ser abordado minuciosamente, explicando em detalhe o seu funcionamento e como é realizado o encapsulamento necessário de pacotes. Posteriormente é descrita a nova arquitetura modular e baseada em eventos incluindo os módulos usados e como eles se interligam entre si. Por fim são descritas as implementações dos diversos módulos criados.

4.1 Drone-RK

Para a realização deste projeto usou-se um simulador *open source* designado por Drone-RK. Este foi desenvolvido de modo a funcionar em ambiente simulado, onde é possível simular uma rede constituída por diversos UAVs e as suas respetivas comunicações ou em ambiente real onde cada computador é um UAV presente na rede. Durante o desenvolvimento deste trabalho usou-se sempre o modo em ambiente real. Para a realização das comunicações são usadas diversas *frameworks* tais como TDMA e *routing multi-hop*. Para além de vários códigos de exemplo, temos ainda acesso ao código dos diferentes módulos representadas na figura 4.1. Para o desenvolvimento desta solução usou-se, já incluídos no simulador, os módulos TDMA para incluir o protocolo TDMA e o módulo Utils com várias ferramentas gerais.

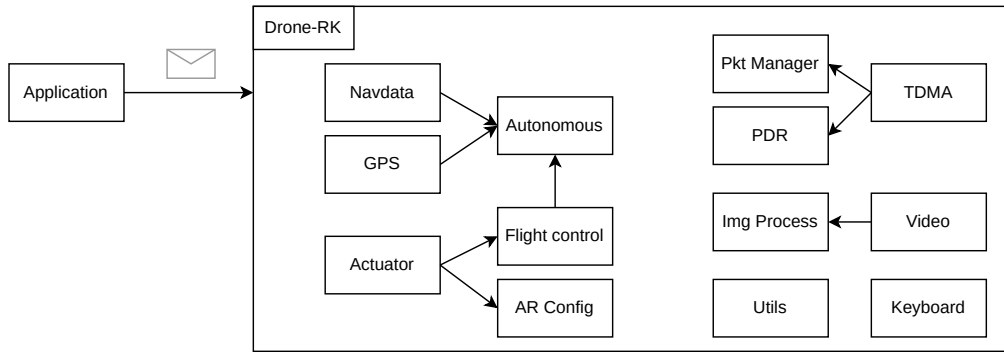


Figura 4.1: Módulos e dependências Drone-RK [8]

4.1.1 Módulo TDMA

O módulo TDMA é o responsável pelo controlo de acesso ao meio. Este é dividido em duas *threads*, a *thread* de envio de pacotes (TX) e a de receção de pacotes (RX). É de realçar que a *thread* TX só está ativa durante os limites da *slot*, no resto do tempo está em modo *sleep*. Quando uma aplicação necessita de enviar um pacote, figura 4.2, chama a função *TDMA_Send()* e este é adicionado a uma fila de pacotes em espera (TX Queue) até ser possível transmitir. Pouco antes do início da *slot* é realizada a sincronização, ou seja, a alteração dos limites da *slot* caso seja necessário. Na implementação padrão do Drone-RK só é possível atrasar a *slot* no tempo. Após a sincronização ser feita é a altura de transmitir. Os pacotes saem, um a um, da fila de espera (TX Queue) e vão para a etapa de encapsulamento. Por fim, o pacote é enviado com o uso de *sockets* UDP.

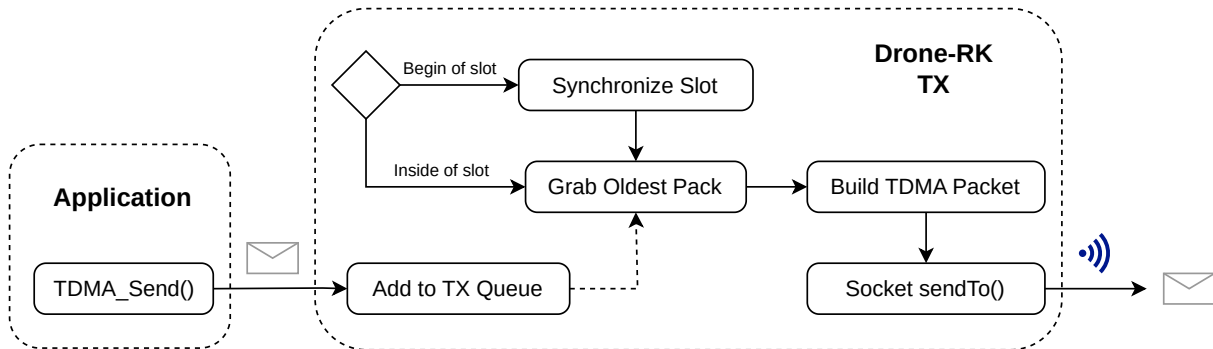


Figura 4.2: Envio de um pacote

Na *thread* RX, após os pacotes chegarem à *socket*, é realizado o desencapsulamento dos cabeçalhos e o cálculo do *delay* do pacote. Por fim, o pacote é adicionado a uma fila de espera (RX Queue) até ser recebido pela aplicação com a chamada da função *TDMA_Receive()*.

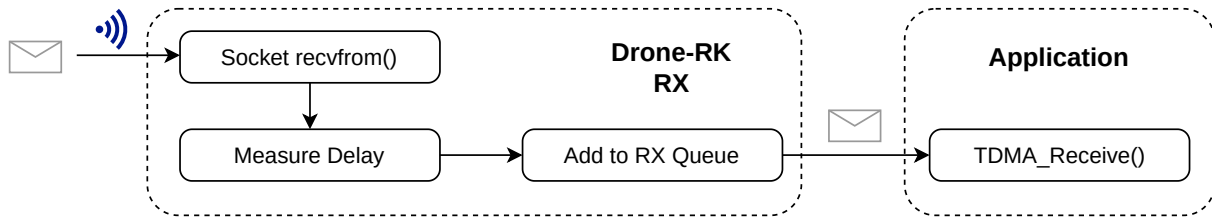


Figura 4.3: Receção de um pacote

No entanto, para conseguir implementar os diferentes algoritmos necessários, foi preciso reformular toda a arquitetura da *framework* TDMA no simulador. A *framework* original estava implementada para realizar a sincronização um pouco antes do início da *slot* mas, diferentes algoritmos podem apresentar diferentes momentos de sincronização, tais como o RA-TDMAs+ e o DESYNC-TDMA. Para além disso esta arquitetura não permite antecipar a *slot*, só atrasar. Decidiu-se então criar uma nova arquitetura TDMA modular e baseada em eventos que é descrita na secção seguinte.

4.1.2 Encapsulamento dos pacotes

Como referido anteriormente na secção 4.1.1, os pacotes antes de serem enviados são sujeitos a uma etapa de encapsulamento. Nesta etapa o pacote é revestido com um cabeçalho TDMA onde estão presentes dados como o tipo de pacote, o número de sequência geral e entre links, o *timestamp* relativo de envio (tempo atual da ronda TDMA), o ID e os limites da *slot* de transmissão. No total são 18 Bytes de cabeçalho TDMA, sendo este essencial ao funcionamento do protocolo RA-TDMAs+, como explicado na secção 3.1.3.

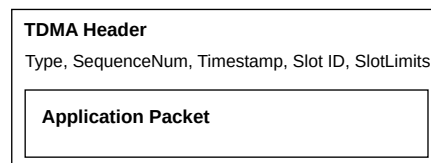


Figura 4.4: Pacote enviado na rede

4.2 Nova Arquitetura TDMA

O desenvolvimento desta arquitetura modular e baseada em eventos, em *User Space*, tornou-se crucial para a implementação dos diversos algoritmos de sincronização. Esta é constituída essencialmente por duas partes, o *TDMA_Core* e os diversos módulos extra que podem ou não ser utilizados. *Topology Tracking* e *Synchronizer* são alguns dos módulos existentes. As principais vantagens desta arquitetura em relação à tradicional usada no simulador Drone-RK são:

- **Modularidade** - É possível, durante a execução do *software*, adicionar ou remover módulos

consoante as necessidades e sem interferir com o bom funcionamento do *TDMA_Core*.

- **Eventos** - Todas as alterações necessárias a aplicar no protocolo TDMA, tais como a alteração dos limites e ordem das *slots* na ronda, são enviadas pelos diversos módulos e processadas pelo *TDMA_Core* a qualquer momento. Assim, é possível ultrapassar a limitação da sincronização ser realizada apenas no início da *slot*, pois a qualquer momento é possível gerar um evento para alterar os respetivos limites.

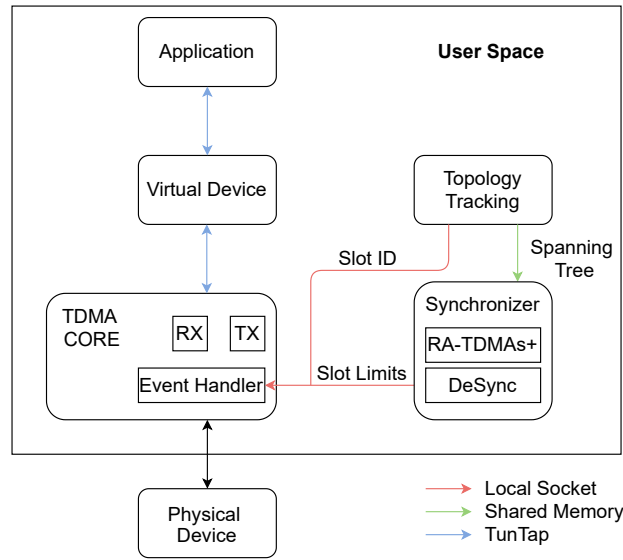


Figura 4.5: Arquitetura da *framework* TDMA

4.2.1 Comunicação entre módulos

Numa arquitetura modular é essencial existir comunicação entre módulos. Como cada módulo é um processo usou-se *Inter Process Communication* (IPC). Existem diversas abordagens de IPC, tais como *Pipes*, *Queues*, *Shared Memory*, *Signals*, *Sockets*, *Semaphores* e *Remote Procedure Call* (RPC). Para a comunicação entre os diferentes módulos e o *TDMA_Core* usou-se *Local Sockets* do tipo *AF_UNIX*. Utilizou-se *Shared Memory* para partilhar a informação necessária ao bom funcionamento dos módulos, por exemplo a partilha da *Spanning Tree* entre o módulo de *Topology Tracking* (criador e com permissões para alterar) e o módulo *Synchronizer* (só com permissões de leitura), para além de todas as informações gerais da ronda TDMA.

Desenvolveu-se um ficheiro em C, *tdma_ipc.c*, para lidar com a comunicação entre o módulo *TDMA_Core* e os restantes. Como referido anteriormente usaram-se essencialmente dois métodos de comunicação, *Shared Memory* para partilha de informação e *AF_UNIX Sockets* para enviar ao *TDMA_Core* alterações de limites ou IDs de *slots*. A *Shared Memory* com a informação global do protocolo TDMA é criada pelo módulo *TDMA_Core*, sendo ele o único com permissões de escrita. Caso outros módulos queiram modificar alguma variável lá presente necessitam de enviar um evento, com o uso de *AF_UNIX Sockets*, ao *TDMA_Core* a requisitar a mudança.

```
typedef struct {
    int64_t      counter; //number of delays already in arrays
    float        ms[PKTDELAY_ARRAY_SIZE]; //delay of this packet
    unsigned int sender_id[PKTDELAY_ARRAY_SIZE]; //who sent this pkt
} pkt_delay_t;
```

Bloco de Código 4.1: Variáveis com os *delays* guardadas na *Shared Memory*

```
typedef struct {
    uint16_t     standard_width_ms; // roundperiod / slot_total
    uint8_t      total;             // number of slots in the network - max 254
    uint8_t      id;                // my slot id
} slot_info_t;

typedef struct {
    uint16_t     round_period_ms ; // TDMA round period, in ms
    slot_info_t  slot_info;
    uint8_t      node_ip; // myy ip
    /* ... */
} param_tdma_t ;
```

Bloco de Código 4.2: Variáveis com a informação TDMA na *Shared Memory*

```
typedef struct {
    //first part
    pthread_mutex_t mut_stats;
    volatile uint8_t numberOfActiveNodes;
    param_tdma_t    param_tdma;
    volatile pkt_delay_t delays;

    //second part
    pthread_mutex_t mut_slot;
    slot_limits_t   myslot;
} tdmaSHM_t;
```

Bloco de Código 4.3: Variáveis guardadas na *Shared Memory*

Nos blocos de código 4.1, 4.2 e 4.3 estão representadas todas as variáveis guardadas na *Shared Memory*. Com base no número de vezes que cada variável precisa de ser acedida decidiu-se dividir a *struct tdmaSHM_t* em duas secções. Na primeira secção temos o número total de nós na rede num dado momento, *numberOfActiveNodes*, os parâmetros globais da ronda TDMA, *param_tdma*, e os *delays* dos pacotes recebidos, *delays*. Os *delays* dos pacotes são guardados na *struct pkt_delay_t* onde está o número total de pacotes recebidos, o respetivo ID do nó que o enviou e o *delay* de cada um representado em milissegundos. Na segunda secção apenas temos os limites, início e fim, da *slot* de transmissão. Cada secção é protegida por um *mutex* para evitar acessos simultâneos pelos diferentes módulos.

```
typedef struct{
    uint8_t type;
    slot_limits_t limits;
} __attribute__((packed)) event_slot_limits;
```

Bloco de Código 4.4: Estrutura do evento *event_slot_limits* enviado na *Socket AF_UNIX*

```
typedef struct{
    uint8_t type;
    slot_info_t slot_info;
} __attribute__((packed)) event_slot_id;
```

Bloco de Código 4.5: Estrutura do evento *event_slot_id* enviado na *Socket AF_UNIX*

Quando um módulo necessita de comunicar com o *TDMA_Core* fá-lo, criando um evento, através de *Sockets AF_UNIX*. Existem dois eventos, evento do tipo 0 que serve para realizar a alteração dos limites da *slot* e evento do tipo 1 para alterar o respetivo ID. Nos blocos de código 4.4 e 4.5 estão representadas as estruturas dos dois eventos. Na *struct event_slot_limits* temos a variável *limits* que guarda o tempo de início e fim da *slot* e no *event_slot_id* a variável *slot_info* com a informação do número total de *slots*, o tamanho único de cada uma e o novo ID a usar. Ambas as *structs* tem a variável *type* que representa o tipo de evento enviado.

Para poder usar aplicações reais na nossa arquitetura, como o VLC, usou-se o TUN/TAP. O TUN/TAP é uma ferramenta que permite criar uma interface de rede virtual com acesso e control sobre os pacotes enviados e recebidos nessa interface. Este divide-se em dois modos TUN e TAP. O modo TUN permite criar um interface para pacotes IP da camada 3 (rede) da pilha protocolar, já o modo TAP cria um interface para *frames Ethernet* na camada 2 (Ligação de dados) da pilha protocolar. Para criar um túnel entre as aplicações e o *TDMA_Core* usou-se o modo TUN. As aplicações enviam os pacotes para o IP correspondente à interface virtual *tun0*, sendo depois capturados por um programa em C, *tuntap.c*, representado na figura 4.5 como *Virtual Device*. Os pacotes são posteriormente encapsulados com o cabeçalho TDMA, secção 4.1.2, o IP de destino é alterado e são enviados pelo *TDMA_Core* durante a *slot* de transmissão. Na receção é realizado o caminho inverso. Esta abordagem tem a desvantagem de criar um *overhead* ao pacote enviado pela aplicação. O pacote original depois de ser capturado pelo *tuntap.c* é enviado ao *TDMA_Core* onde é encapsulado como demonstrado na secção 4.1.2. No final é enviado um novo pacote UDP que contém como *payload* o pacote UDP enviado pela aplicação mais os *headers* TDMA adicionados.

4.2.2 *TDMA_Core*

O *TDMA_Core* é o principal processo nesta arquitetura, pois é a partir dele que é implementado e gerido todo o protocolo TDMA. Este é dividido em quatro *threads*, a *RX*, a *TX*, a *Config* e o

Event Handler. O funcionamento das *threads RX* e *TX* é semelhante ao já implementado pelo simulador, figuras 4.3 e 4.2. A *thread Config* surgiu para resolver o problema de sincronização quando a *slot* de transmissão se sobrepõe ao final da ronda, começando na ronda n e acabando em $n+1$. É na mudança de ronda que todas as configurações TDMA, incluindo os novos limites de *slot*, são atualizados, tornando assim este momento fulcral para o funcionamento do protocolo de sincronização. A *thread Config* fica então responsável por toda a configuração da ronda TDMA e pelo início da *thread TX* com o uso de *Semaphores*. Para receber e responder com as alterações necessárias aos eventos produzidos criou-se a *thread Event Handler*.

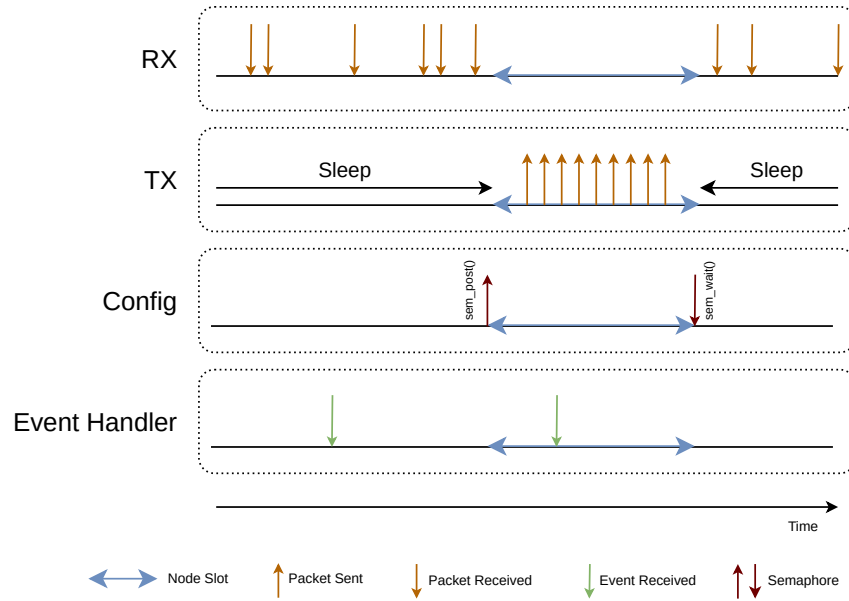


Figura 4.6: *Threads* presentes no *TDMA_Core*

4.2.3 *Tracking* da topologia

O *tracking* da topologia é essencial para o bom funcionamento do protocolo RA-TDMAs+, tendo sido por isso necessário a criação de um módulo apenas responsável pelo *tracking* de topologia. Este módulo foi implementado no ficheiro *matrix.c*. Para manter uma visão global da rede é necessário saber o número total de nós ativos na rede, os respetivos IDs e uma matriz com a informação dos *links* entre nós. Para manter a matriz sempre atualizada com a informação mais recente também foi necessário guardar a idade, *age*, de cada linha presente na matriz. Esta informação foi guardada numa *struct*, *tdma_matrix_t*, como representado no bloco de código 4.6.

```
typedef struct {
    uint8_t  myId;
    uint8_t  numberOfActiveNodes;
    uint8_t  matrix[MNUM_DRONES][MNUM_DRONES];
    uint8_t  idOfActiveNodes[MNUM_DRONES];
    double   creationTime[MNUM_DRONES];
    double   age[MNUM_DRONES];
}
```

```
} __attribute__((packed)) tdma_matrix_t ;
```

Bloco de Código 4.6: Variáveis usadas para guardar a informação da matriz

Este módulo apresenta três funções principais: *MATRIX_init()* responsável por carregar o módulo e iniciar todas as variáveis necessárias, *MATRIX_parsePkt()* para fazer o *parse* dos pacotes recebidos e atualização da matriz e o *MATRIX_share()* para enviar a matriz local em *broadcast* aos restantes nós.

Um nó após receber um pacote com destino à porta 2000 chama a função *MATRIX_parsePkt()*. Nesta é realizada a desserialização do pacote numa variável do tipo *tdma_matrix_t* e posteriormente a informação recebida é usada para realizar o *update* da matriz local. No processo de *update* a matriz recebida é comparada linha a linha com a matriz local. É de realçar que a mesma linha nas duas matrizes pode representar nós com IDs diferentes, sendo por isso necessário procurar a linha respetiva ao ID, caso esta exista. Isto acontece quando os nós ainda não têm a informação de todos os nós presentes na rede, ou seja, durante o processo de descoberta. A linha da matriz local é atualizada caso a respetiva linha na matriz recebida contenha informação mais recente, isto é, valor de idade inferior. As linhas dos nós que não estão presentes na matriz local são copiadas diretamente para a respetiva posição, atualizando também o número total de nós na rede, os respetivos IDs e a ordem das *slots*. Para todos os nós manterem a mesma visão da configuração atual da ronda, os IDs são colocados por ordem crescente e atribuídos por ordem às *slots*. Numa rede com IDs iguais a 3,5,1 a *slot* 1 ficaria com o ID 1, a *slot* 2 com o ID 3 e, por fim, a *slot* 3 com o ID 5. Para terminar o nó atualiza a sua própria linha adicionando, caso ainda não tenha, *link* ao nó que enviou a matriz. Após a realização do *update* caso o ID da *slot* do nó altere é enviado um evento do tipo 1 (*Slot ID*) ao *TDMA_Core*.

No início da *slot* de transmissão o nó envia a sua informação local aos restantes nós através da função *MATRIX_share()*. Antes de proceder ao envio removem-se as linhas e respetivos IDs de nós com idade superior a 3 rondas TDMA. Caso alguma linha atinja este *timeout* de 3 rondas é necessário enviar um evento do tipo 1 (*Slot ID*) ao *TDMA_Core*. Por fim, calcula-se a idade de cada linha, realiza-se a serialização da matriz, como representado na figura 4.7, e envia-se em *broadcast* para a rede.

Number Of Nodes (n)	ID of the nodes [n]	Matrix [n][n]	Age [n]
---------------------	---------------------	---------------	---------

Figura 4.7: Estrutura do pacote Matrix enviado

A variável idade, *age*, tem um papel fundamental neste módulo, pois é ela que dita se uma dada informação é mais recente ou não do que outra. Quando um nó envia a sua matriz local ele envia a linha que corresponde ao seu ID com idade igual a 0. O nó que recebe verifica a idade e calcula uma estimativa do tempo de criação, *creationTime*, com a seguinte formula:

$$creationTime = epoch - age \quad (4.1)$$

Posteriormente quando essa informação é reenviada pelo nó que a recebeu vai ter idade igual a:

$$age = epoch - creationTime \quad (4.2)$$

Sendo que a variável *age* é um valor que está sempre a aumentar.

Para além da matriz de topologia é ainda gerada uma *Spanning Tree* com o algoritmo *Prim* que é partilhada com os diversos módulos com o uso de *Shared Memory*. Esta é atualizada ao enviar, depois de remover os *links* que atingiram *timeout* e após realizar o *update* da informação local na receção de uma matriz. As variáveis guardadas na *Shared Memory* da *Spanning Tree* estão representadas no bloco de código 4.7.

```
typedef struct{
    pthread_mutex_t    mut_spanningTree;
    uint8_t            numberOfActiveNodes;
    uint8_t            spanningTree[MNUM_DRONES][MNUM_DRONES];
    uint8_t            idOfActiveNodes[MNUM_DRONES];
} spanningTreeSHM_t;
```

Bloco de Código 4.7: Variáveis guardadas na *Shared Memory* relativa a *Spanning Tree*

4.2.4 Sincronização

Para a implementação dos algoritmos de sincronização criaram-se dois módulos diferentes, RA-TDMAs+ e DESYNC, como observado na figura 4.5.

4.2.4.1 RA-TDMAs+

Este módulo tem como objetivo a implementação do algoritmo RA-TDMAs+. Por o algoritmo necessitar de realizar o *tracking* da topologia e aceder aos *delays* dos pacotes este módulo tem acesso de leitura às *Shared Memory* da *Spanning Tree* e do *TDMA_Core*. Após o módulo RA-TDMAs+ iniciar e obter acesso a todas as informações necessárias entra em modo *sleep* até três milissegundos antes do final da ronda TDMA. Estes três milissegundos de intervalo são essenciais pois permitem aceder aos *delays*, calcular o valor delta e retornar essa informação ao *TDMA_Core* antes de ele alterar os limites da *slot* no final da ronda. Quando o tempo de *sleep* acaba o algoritmo de sincronização é realizado com os *delays* fornecidos pelo *TDMA_Core* e guardados na *Shared Memory*. Os *delays* são calculados como representado na secção 3.1.3.

Com o uso da *Spanning Tree* é realizada uma filtragem nos *delays* a usar de modo a evitar ciclos de topologia na sincronização. Apenas são usados os *delays* em que existe *link* na *Spanning Tree* entre o nó transmissor e nó que está a realizar a sincronização. Após a filtragem realizada e os *delays* separados por ID de transmissor é calculado a média dos valores de *delay* para cada ID. Este processo é descrito em detalhe na secção 3.1.3. Entre todos os valores de média é escolhido o maior valor para o ajuste da *slot*, valor delta, sendo este valor limitado a vinte e cinco por cento do tamanho da *slot* atual. Por fim os limites da *slot* são ajustados com o novo valor de delta e enviados para o *TDMA_Core* com o uso de um evento do tipo 0. Este processo está demonstrado na figura 4.8.

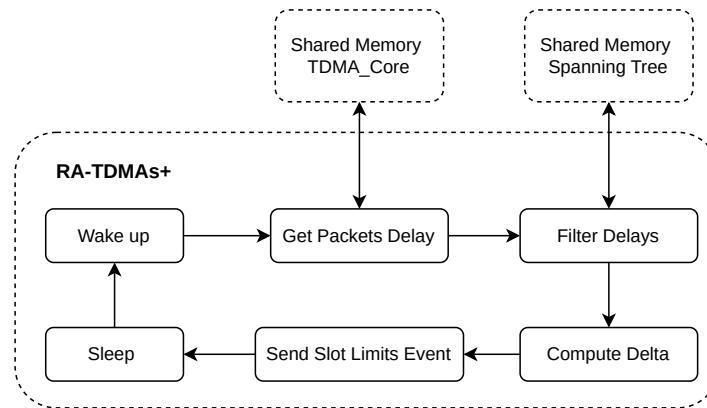


Figura 4.8: Etapas presentes no módulo RA-TDMAs+

4.2.4.2 DESYNC

Este módulo tem como objetivo implementar o algoritmo de sincronização DESYNC. Por este usar pacotes próprios para a realização da sincronização o módulo foi dividido em duas *threads*, a *thread* RX para receber os pacotes e realizar a sincronização e a *thread* TX para enviar os pacotes. Para a sincronização são necessárias três variáveis, *my_fire*, que corresponde ao *epoch* de envio do pacote do nó, o *epoch* de receção do pacote antecessor, *prev_fire*, e o *epoch* de receção do pacote posterior ao enviado pelo o nó, *next_fire*.

Como referido no capítulo 3.2.2 para adaptar o algoritmo DESYNC a redes *multi-hop* é necessário os nós terem conhecimento dos seus vizinhos até dois *hops*. Para isso, cada nó envia no seu pacote a informação dos seus vizinhos diretos, nós de quem recebeu pacotes. A estrutura do pacote, representado na figura 4.9, consiste inicialmente num byte com a indicação do número total de nós, seguido de um *array* com os seus IDs e por fim um *array* com a diferença de tempo, *epoch* – tempo de receção do pacote.

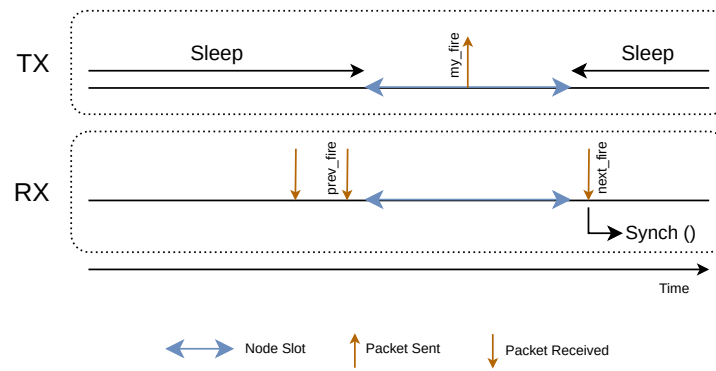
Number Of Nodes (n)	ID of the nodes [n]	Diff Time [n]
---------------------	---------------------	---------------

Figura 4.9: Estrutura do pacote DESYNC enviado

Como referido em 4.2.2 é no final da ronda TDMA que todas as configurações incluindo os limites da *slot* são atualizadas. Por isso, a *thread* TX após inicializada entra em modo *sleep* até ao final da ronda. Posteriormente ao obter o valor atualizado de limites de *slot* entra outra vez em modo *sleep* até ser necessário enviar o pacote. O tempo de envio do pacote corresponde exatamente ao meio da *slot*. Por fim o pacote é criado, enviado e o valor de *epoch* atual é atribuído à variável *my_fire*.

A *thread* RX só é ativada quando recebe um pacote, o resto do tempo está em modo *sleep*. Após a receção do pacote o nó atualiza a sua informação local com os dados presentes no pacote, ID do nó e tempo de envio de pacote. Caso o pacote tenha sido recebido posteriormente ao nó ter enviado o seu pacote é realizada a sincronização. Com o uso da informação local o nó descobre o valor das variáveis *prev_fire* e *next_fire* e usa a equação 3.19 para calcular os novos limites de *slot*. O valor máximo de deslocamento da *slot*, Δ_{max} , é vinte e cinco por cento do seu tamanho. Todo este processo é descrito em detalhe na secção 3.2.1. Um aspeto relevante é o facto dos valores dos tempos nos pacotes serem instantes de transmissão, Por outro lado, as correções são efetuadas com instantes de receção. Para se poder relacionar os tempos de receção com os tempos recebidos nos pacotes foi necessário remover aos tempos de receção o tempo de transmissão na rede, ou seja, o tempo que o pacote demora a chegar desde que foi enviado até ao destino. Escolheu-se o valor de 3 milissegundos por tentativa e erro. Por fim, os novos limites de *slot* são enviados ao módulo *TDMA_Core* com o uso de eventos do tipo 0.

Para além do algoritmo original implementaram-se defesas contra a perda de pacotes. A cada pacote perdido a instabilidade da rede aumenta o que provoca constantes reconfigurações e um aumento do tempo até chegar ao ponto ideal de sincronização. Como representado na figura 3.5 a ordem dos nós em relação aos seus vizinhos é constante até existirem alterações no número total de nós. Numa rede com quatro nós {1,2,3,4} se a ordem dos nós for 3,2,1,4 o nó 2 deve realizar sempre a sincronização com os nós 3 e 1, ou seja, o *prev_fire* deve ser sempre relativo ao nó 3 e *next_fire* relativo ao nó 1. No entanto, no algoritmo original este problema não é tido em conta e caso o pacote do nó 1 falhe, o nó 2 usa o nó 4 como *next_fire*, provocando assim instabilidade na rede. Para contornar este problema e melhorar a estabilidade da rede implementou-se um mecanismo de defesa baseado apenas na informação local do nó. A partir do *epoch* do pacote recebido calcula-se o tempo relativo na ronda TDMA e com esses tempos define-se a ordem dos nós na rede. A ordem definida é igual para todos os nós. A conversão de *epoch* para tempo relativo é extremamente importante, pois caso haja alguma perda ou o pacote ainda não tenha sido recebido é possível fazer a comparação com pacotes anteriores, mantendo assim a ordem. Após a ordem definida e a associação *prev_fire* e *next_fire* a determinados nós, é possível estimar e usar o *epoch* do pacote perdido para a realização dos cálculos. No exemplo demonstrado em cima, caso o nó 2 perdesse um pacote do nó 1 ia estima-lo com base no último pacote recebido desse mesmo nó em vez de usar o do nó 4. Após três rondas sem receber informações de um determinado nó procedemos à retirada desse mesmo nó da informação local e configuração das *slots*. Este intervalo de tempo permite-nos fazer a distinção entre simples perdas de pacotes e nós a sair da rede. No entanto, para conseguir fazer as estimativas é necessário cada nó guardar a informação dos IDs dos restantes nós conhecidos e o *epoch* do seu ultimo pacote recebido.

Figura 4.10: *Threads* presentes no módulo DESYNC

Capítulo 5

Experiências e Resultados

Este capítulo descreve as experiências realizadas, as métricas usadas na comparação e os respectivos resultados obtidos. Inicialmente aborda-se as métricas, explicando em detalhe a sua definição, a sua maneira de calcular e o significado do seu resultado. Em seguida, é descrito o *setup* usado e as diversas experiências realizadas. Por fim, são descritos os resultados obtidos das experiências realizadas, onde é explicado em detalhe o seu significado para cada métrica avaliada. Todos os resultados foram retirados dos *logs* do nó 1, ou seja, correspondem à visão que o nó 1 têm da rede.

5.1 Métricas

Para qualificar e comparar os dois algoritmos de sincronização propostos, usaram-se três métricas. As métricas são Erro de Sincronização, Período Efetivo e Perda de Pacotes, sendo que a métrica Erro de Sincronização é calculado de maneira diferente para cada algoritmo. Foi implementado um *script* Python que, a partir dos *logs* fornecidos pelo simulador, calcula as métricas e apresenta o resultado em forma de gráfico.

5.1.1 Erro de sincronização

A métrica Erro de Sincronização quantifica o quão sincronizada a rede está. Os dois algoritmos de sincronização testados, RA-TDMAs+ e DESYNC, usam definições diferentes para o cálculo desta métrica.

No algoritmo RA-TDMAs+ [9] esta métrica mede o intervalo de tempo entre o início da *slot* de transmissão local e o tempo (estimado) de fim da *slot* imediatamente anterior. O cálculo do tempo estimado de fim da *slot* é calculado com as informações recebidas no cabeçalho TDMA do pacote, secção 4.1.2, sendo que esta métrica é calculada em todos os pacotes recebidos. O erro de sincronização do nó j na ronda r é calculado com a seguinte equação:

$$\epsilon_s^{(j)}(r) = \hat{E}^{(j-1)}(r) - B^{(j)}(r) \quad (5.1)$$

onde $\hat{E}^{(j-1)}(r)$ representa o tempo estimado de fim da *slot* anterior e $B^{(j)}(r)$ o início da *slot* local. Um resultado negativo indica que a *slot* anterior (estimada) acabou mais cedo do que o previsto criando um intervalo entre as *slots*, enquanto que um valor positivo indica a sobreposição das *slots*. Idealmente o resultado desta métrica é zero.

O algoritmo DESYNC [3] define esta métrica como a média do desvio do tamanho ideal de *slot* numa dada ronda. Este tamanho é igual a T/n onde T é o período da ronda TDMA e n o número de nós. Esta métrica é calculada com a seguinte equação:

$$\epsilon_s(r) = \frac{1}{n} \left\| \vec{\Delta}^{(k)} - \vec{\Delta}^* \right\|_1 \quad (5.2)$$

onde $\vec{\Delta}^{(k)}$ representa o tamanho da *slot* k e $\vec{\Delta}^*$ o tamanho ideal da *slot*.

5.1.2 Período Efetivo

A métrica Período Efetivo ϵ_t representa a visão do nó j do período real da ronda TDMA. É calculada, como representado na equação 5.3, a partir do limite inicial da respetiva *slot* de transmissão do nó.

$$\epsilon_t^{(j)}(r) = B^{(j)}(r) - B^{(j)}(r-1) + T \quad (5.3)$$

onde B representa o tempo de início da *slot* de transmissão, r a ronda TDMA a calcular e T o período TDMA. Numa rede sincronizada o valor ideal de ϵ_t é igual ao valor da ronda TDMA, T .

5.1.3 Perda de Pacotes

Esta métrica, calculada entre *links* de nós, verifica a quantidade de pacotes enviados por um dado nó que não foram recebidos. Quanto mais pequeno for o valor de perda de pacotes melhor, no entanto uma taxa inferior a cinco por cento já é um valor aceitável. O cálculo desta métrica é realizado com o uso dos números de sequência presentes nos pacotes recebidos.

5.2 Setup

Para a realização das experiências criou-se uma pequena rede em laboratório com cinco computadores, quatro computadores (modelo Samsung N150 Plus) a correr o algoritmo de sincronização

e outro a monitorizar a rede com o uso de Wireshark e gráficos em tempo real feitos em Python. Todos os computadores tinham como sistema operativo o Linux, Xubuntu nos computadores a correr o algoritmo e Ubuntu no computador a monitorizar a rede. Cada computador está equipado com uma placa de rede 802.11 b/g/n externa configurada na mesma rede *ad hoc* e com endereço IP igual a 192.168.2.X onde X é o identificador do nó. Para uma maior facilidade a editar código e a correr as experiências configurou-se a interface eth0 como estática com IP 192.168.4.X. Ligaram-se todos os computadores por cabo a um *switch* e através de SSH realizaram-se as alterações necessárias.

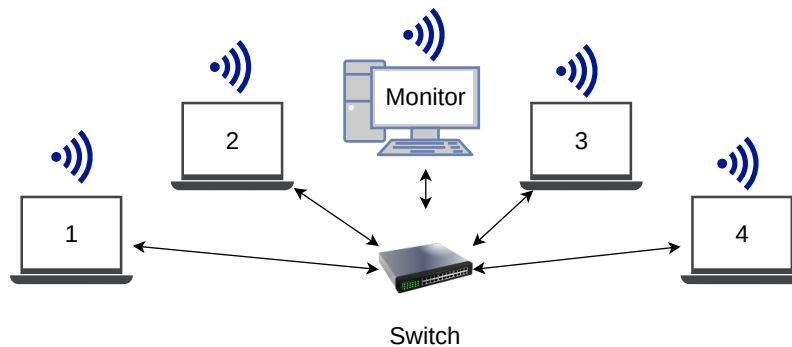


Figura 5.1: Setup usado nas experiências

Para a realização das experiências criou-se, em cada computador, um *script* em Bash com o comportamento que cada nó deve ter ao longo da experiência. No bloco de código 5.1 está representado o comportamento do nó 1. Assim que o *script* é inicializado o nó começa logo a transmitir, passado 10 segundos o processo é desligado e após 15 segundos de ter sido desligado volta a transmitir, durante 35 segundos, até ao final da experiência. Para inicializar os *scripts* ao mesmo tempo em todos os computadores usou-se a ferramenta *pdsh*. Esta ferramenta permite o envio de comandos SSH paralelos para vários *hosts*. No bloco de código 5.2 está representado o código usado para correr as experiências. Para criar diferentes topologias usou-se a *firewall* do Linux (*iptables*) para bloquear a receção de pacotes de um respetivo IP com o seguinte comando: `iptables -A INPUT -s <IP> -j REJECT`

```
#!/bin/bash

cd ~/Desktop/dronerk/src/extras/tuntap/
./tuntap_GS 1 > /dev/null 2>&1 &
pid1=$!

cd ~/Desktop/dronerk/src/synchronizers/<algorithm>
sleep 0.5 ; ./<algorithm> 1 > /dev/null 2>&1 &
pid2=$!

sleep 10

kill -2 $pid1 $pid2
```

```

sleep 15

cd ~/Desktop/dronerk/src/extras/tuntap/
./tuntap_GS 2 > /dev/null 2>&1 &
pid1=$!

cd ~/Desktop/dronerk/src/synchronizers/<algorithm>
sleep 0.5 ; ./<algorithm> 1 > /dev/null 2>&1 &
pid2=$!

sleep 35

kill -2 $pid1 $pid2

```

Bloco de Código 5.1: *Script* de inicialização do nó 1

```

#!/bin/bash

echo "Removing Logs..."
pdsh -R ssh -w <hosts> 'cd ~/Desktop/dronerk/src/extras/tuntap/ && ./rmlogs.sh'

for i in {1..50}
do
    echo "Test $i"
    pdsh -R ssh -w <hosts> 'cd ~/Desktop/<algorithm>/ && ./<experiment>.sh'
    sleep 0.2
done

```

Bloco de Código 5.2: *Script* usado para correr as experiências

Para adicionar interferências na rede usou-se o comando ping a transmitir pacotes *Internet Control Message Protocol* (ICMP) com 64 Bytes a cada 2 milissegundos. Todos os computadores na rede *ad hoc* enviam pacotes ICMP entre si.

5.3 Experiências

Para as experiências simularam-se três configurações diferentes, sendo que cada configuração foi repetida para cada algoritmo de sincronização duas vezes, com e sem interferências na rede. No total foram realizadas doze experiências, sendo que cada uma foi repetida 50 vezes. Todas elas duraram 1 minuto e cada ronda TDMA 100 milissegundos. No total decorreram 600 rondas TDMA. O valor máximo de deslocamento da *slot*, Δ_{max} , foi fixado em vinte e cinco por cento do tamanho da *slot* para ambos os algoritmos e só foram transmitidos os pacotes necessários à realização da sincronização.

A primeira configuração, representada na figura 5.2 serviu para demonstrar o comportamento dos algoritmos quando sujeitos a entradas e saídas de nós em redes *single hop*. Começou-se por

adicionar os quatro nós, com IDs 1,2,3,4, ao mesmo tempo à rede. Após 15 segundos removeram-se dois nós com IDs 2 e 3. Os nós voltaram a entrar na rede depois de 15 e 25 segundos respectivamente, mantendo-se assim o número de nós inalterado até ao final da experiência.

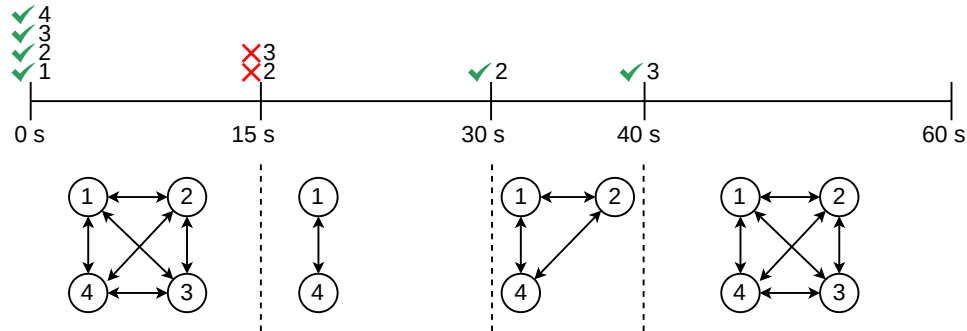


Figura 5.2: Configuração número 1

A segunda configuração, figura 5.3 teve como objetivo analisar o comportamento dos nós em redes com *two hops*. Começou-se com os 4 nós numa topologia em *ring*. Após 20 segundos retirou-se o nó 2 da rede, passando os restantes nós a estarem em linha. Depois de outros 20 segundos voltou-se a adicionar o nó 2 à rede, ficando assim até ao final da experiência.

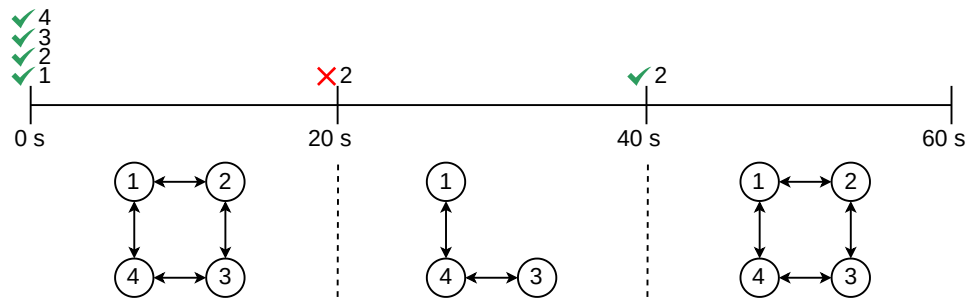


Figura 5.3: Configuração número 2

Por fim, a terceira e última configuração, figura 5.4 realizada em redes com *three hops* teve especialmente o intuito de analisar o comportamento do algoritmo DESYNC, que só usa informação até *two hops* para realizar a sincronização. Iniciou-se a experiência com os quatro nós, IDs 1,2,3,4 ao mesmo tempo em linha. Passado 30s adicionou-se *link* entre o nó 1 e o nó 4, mantendo-se assim a topologia inalterada até ao fim da experiência.

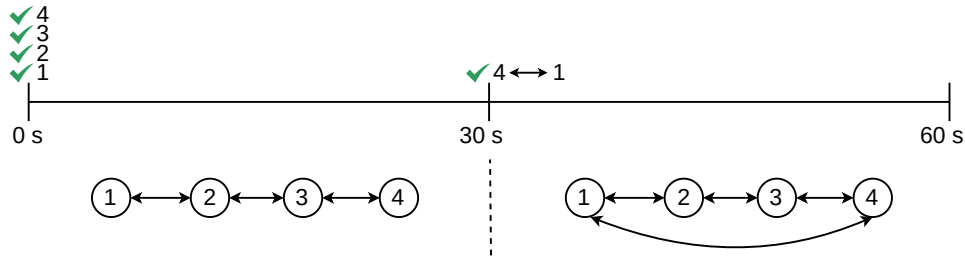


Figura 5.4: Configuração número 3

5.4 Resultados

As figuras 5.5, 5.6 e 5.7 representam a frequência de cada valor de Erro de Sincronização, com e sem interferência, em cada configuração. Esta métrica, por ter significados diferentes para cada algoritmo, não pode ser diretamente comparada entre os mesmos, no entanto há algumas conclusões que podemos retirar. O DESYNC por ser um algoritmo que após sincronizado mantém os limites das suas *slots* sempre iguais apresenta uma maior estabilidade com a média ao longo de toda a experiência a situar-se muito perto do valor 0. Ao contrário do DESYNC, o RA-TDMAs+ é um algoritmo que promove o constante movimento das *slots* para evitar interferências periódicas por isso, apresenta uma maior variação entre rondas com a média de erro a situar-se muito perto do valor 2. É possível observar que mesmo aumentando o número de *hops* na topologia os resultados, respetivos a cada algoritmo, foram semelhantes. É ainda possível observar que quando existe interferências há uma diminuição dos picos dos valores médios e o respetivo aumento da dispersão.

Na figura 5.7 no gráfico do DESYNC é de realçar a coluna situada no valor 2,5. Esta deve-se a característica do DESYNC de configurações iniciais diferentes poderem resultar em organizações de *slots* diferentes. Com os 4 nós presentes na rede em linha (1-2-3-4), o nó 1 e o nó 4 estão separados por 3 *hops*. Quando eles são vizinhos de *slot* tem tendência para se juntar numa só, deixando 3 *slots* para 4 nós. É possível observar este efeito na figura 5.8, sendo que isto só é possível porque os nós estão suficientemente afastados na rede de modo a não causarem colisões. Nos cálculos do Erro de Sincronização para o DESYNC, por não termos acesso ao número efetivo de *slots* realizamos os cálculos com número de *slots* igual ao número de nós presentes na rede. O que provoca que mesmo que a rede esteja sincronizada o erro varia entre 2 e 3.

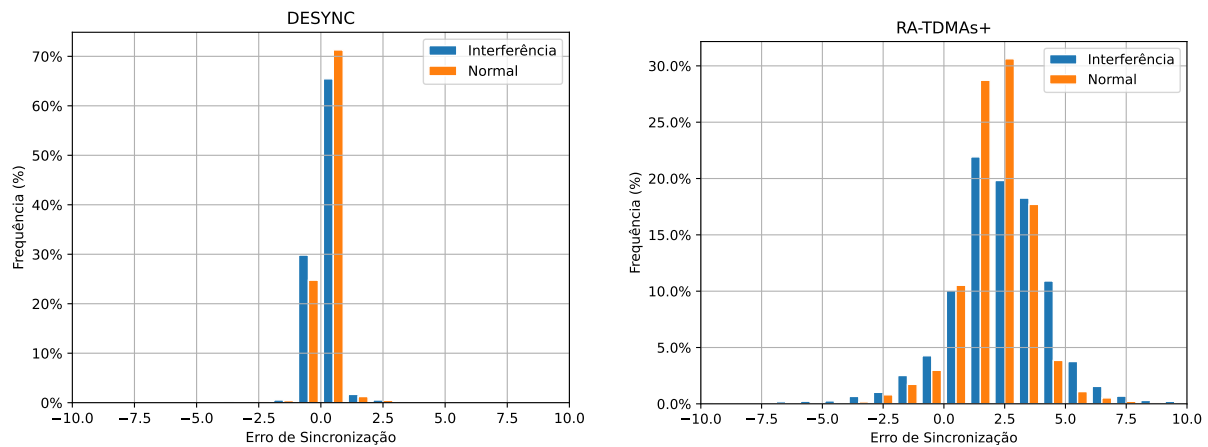


Figura 5.5: Resultados Erro de Sincronização para a configuração número 1, figura 5.2

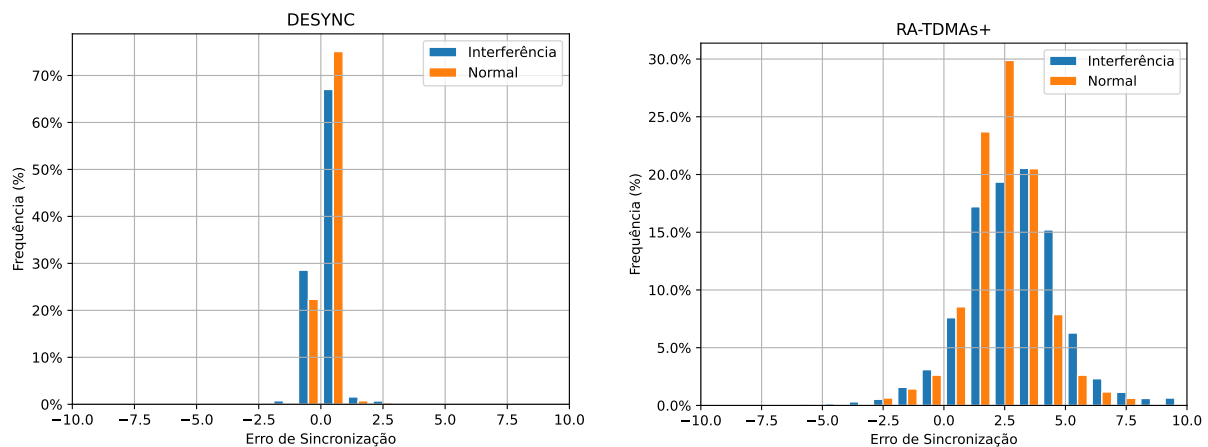


Figura 5.6: Resultados Erro de Sincronização para a configuração número 2, figura 5.3

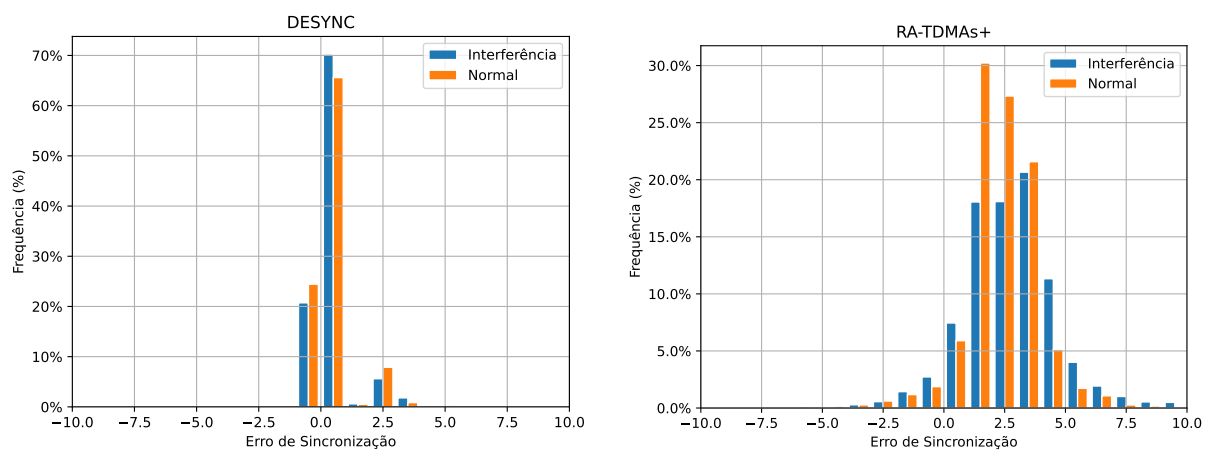


Figura 5.7: Resultados Erro de Sincronização para a configuração número 3, figura 5.4

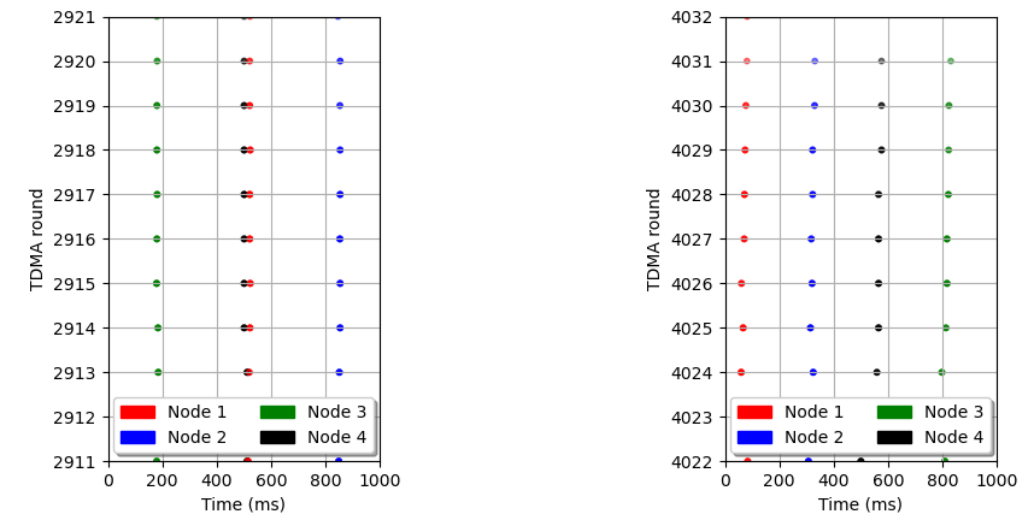


Figura 5.8: Diferentes organizações de *slots* para a configuração número 3

Em relação à métrica Período Efetivo obtiveram-se resultados distintos, no entanto esperados devido à maneira de funcionamento dos algoritmos. O DESYNC pode adiantar ou atrasar o início da *slot*, ou seja, o Período Efetivo pode ser um valor superior ou inferior à ronda TDMA. Já o RA-TDMAs+ como apenas atrasa o início da *slot* de transmissão têm sempre valores de Período Efetivo superiores ou iguais à ronda TDMA. Tal como no Erro de Sincronização observou-se um aumento na dispersão dos valores quando existiu interferências.

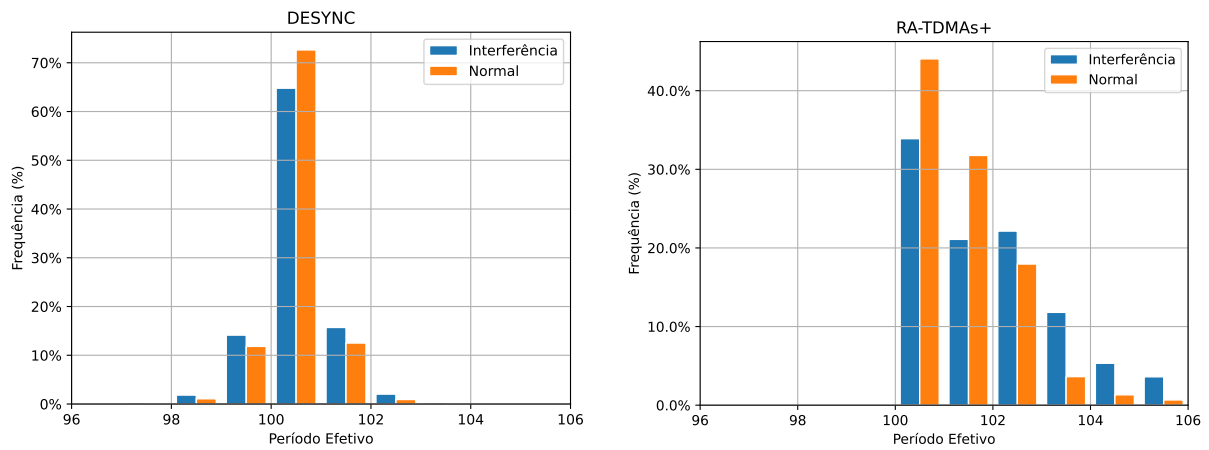


Figura 5.9: Resultados Período Efetivo para a configuração número 1, figura 5.2

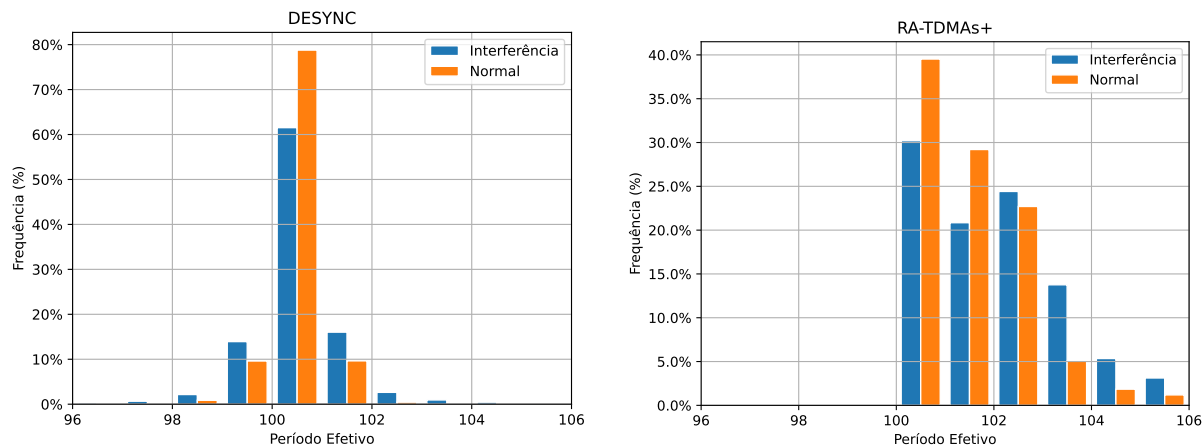


Figura 5.10: Resultados Período Efetivo para a configuração número 2, figura 5.3

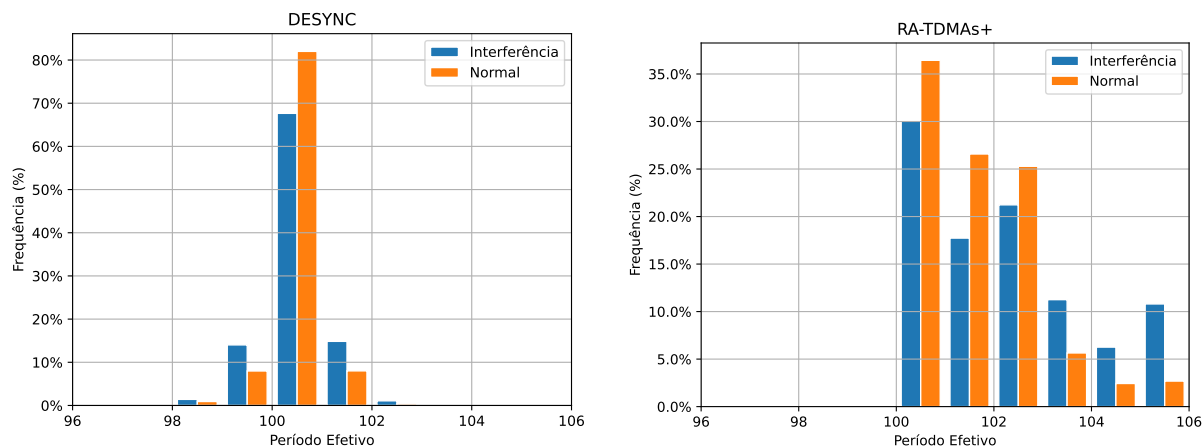


Figura 5.11: Resultados Período Efetivo para a configuração número 3, figura 5.4

Na métrica Perda de Pacotes não se observou praticamente nenhuma diferença entre os algoritmos, no entanto notou-se uma grande diferença na presença de interferências externas. Os gráficos 5.12, 5.13 e 5.14 representam a média de Perda de Pacotes por nó. Este aumento de perda deve-se à grande quantidade de interferência aplicada, 4 pacotes ICMP a cada 2 milissegundos. No entanto, com a análise dos gráficos do Erro de Sincronização, percebemos que mesmo com a grande quantidade de perda de pacotes a sincronização realizou-se com sucesso, o que demonstra que ambos os protocolos estão equipados com mecanismos de defesa à perda de pacotes.

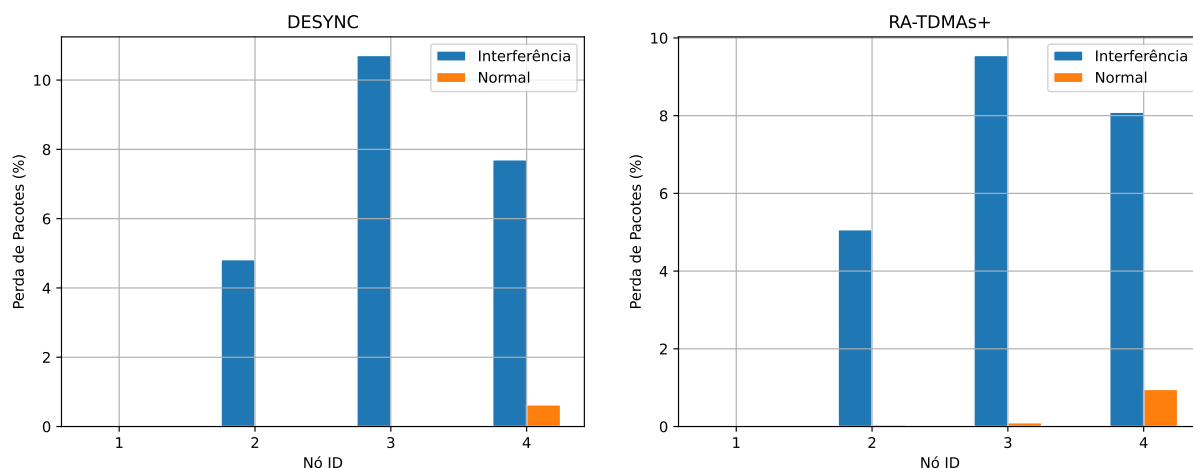


Figura 5.12: Resultados Perda de Pacotes para a configuração número 1, figura 5.2

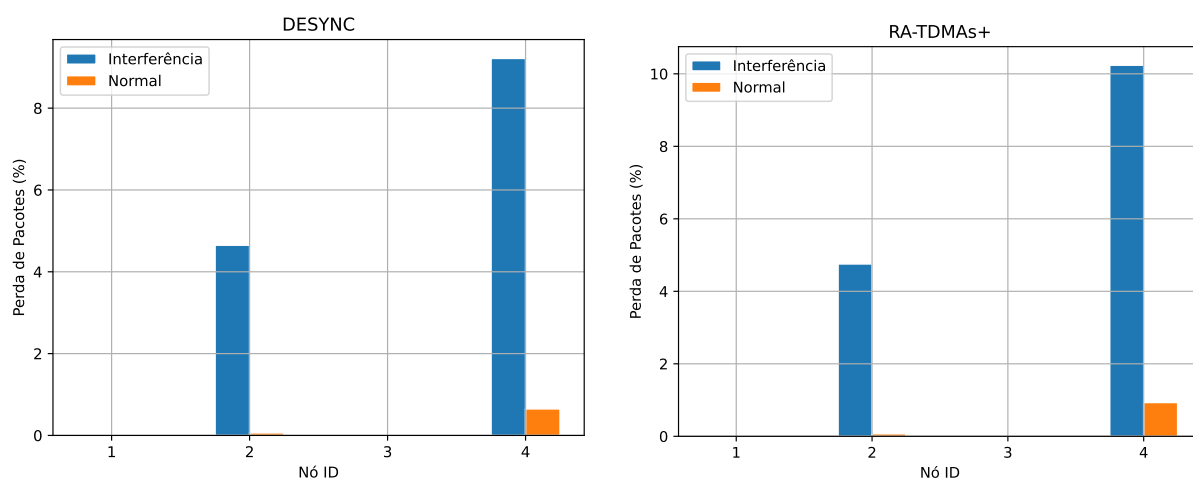


Figura 5.13: Resultados Perda de Pacotes para a configuração número 2, figura 5.3

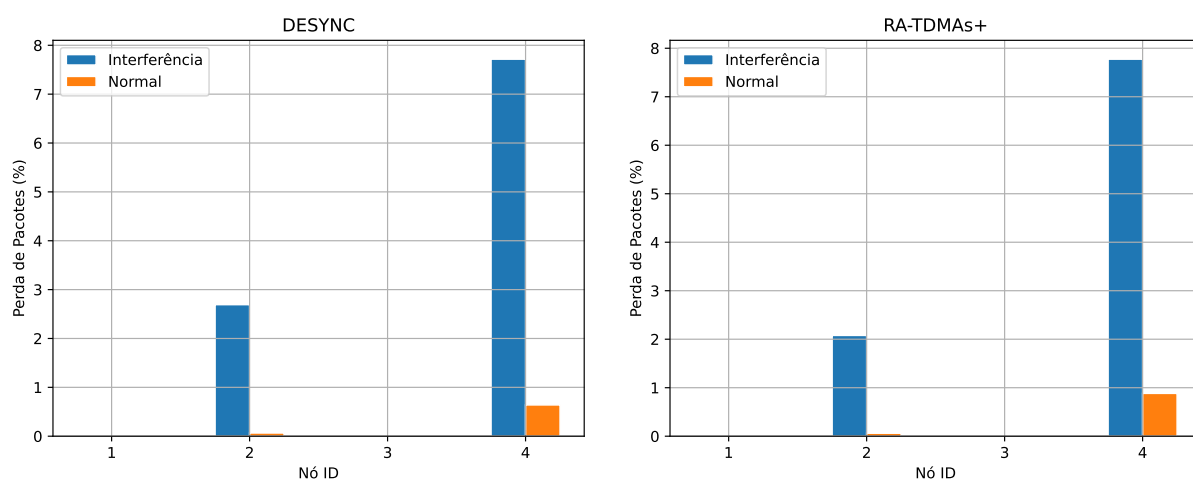


Figura 5.14: Resultados Perda de Pacotes para a configuração número 3, figura 5.4

Capítulo 6

Conclusões

Esta dissertação teve o propósito de estudar e comparar protocolos TDMA descentralizados baseados em sincronização local. O foco caiu especialmente sobre os algoritmos RA-TDMA e DESYNC. Estes realizam a sincronização apenas com o uso dos instantes de pacotes recebidos. Para além da respetiva implementação desenvolveu-se uma arquitetura modular onde é possível trocar o algoritmo sem ser necessário modificar o código, sendo assim fácil comutar entre o uso de RA-TDMA e de DESYNC. Para além disso esta arquitetura também está apta a receber outros módulos que possam ser úteis ao protocolo, por exemplo um módulo de *routing*.

Foram realizados diversos testes com diferentes topologias e variações das mesmas. Apesar de diferentes ambos os protocolos apresentaram uma *performance* semelhante nos testes realizados. Por outro lado foi possível verificar algumas características distintivas dos dois protocolos que se revelaram ao longo da sua implementação, permitindo detetar algumas vantagens e desvantagens dos mesmos. Por exemplo, a adaptação das *slots* em RA-TDMA é feita apenas "para a frente", ou seja, o protocolo nunca antecipa *slots*. Já o DESYNC faz ajustes balanceados em torno do valor desejado para a duração da *slot*. Assim, o DESYNC mantém o período médio das rondas enquanto o RA-TDMA apresenta um período médio superior devido às adaptações das *slots*. Outra diferença visível é a maior dispersão do erro de sincronização em RA-TDMA comparativamente ao DESYNC. Em termos operacionais, ambos os protocolos trocam informações sobre os nós vizinhos nos pacotes de sincronização. Mas enquanto o DESYNC apenas envia e mantém informação até *two hops* o RA-TDMA mantém uma matriz de adjacências atualizada com a visão global da rede. Para além disso, a sincronização no RA-TDMA necessita de informações do transmissor do pacote para o cálculo do *delay*, criando assim, na arquitetura desenvolvida, um *overhead* aos pacotes enviados pela aplicação. Caso seja necessário o uso de *routing* o RA-TDMA já está devidamente preparado, sendo apenas necessário usar a informação que consta na matriz de adjacências. Já no caso do DESYNC seria necessário a implementação de um algoritmo de raiz para a realização do *routing*. É de realçar que não foi possível testar e averiguar a qualidade dos algoritmos em redes com grandes quantidades de nós. Fazer uma escolha de qual o melhor protocolo a usar deverá ter em atenção o tamanho da rede, a necessidade ou não de *routing* e da quantidade de memória disponível em cada nó.

Como trabalho futuro, a partir desta arquitetura, há diversos caminhos que podem ser seguidos. Por exemplo, a criação de mais módulos para serem integrados na arquitetura. A informação da matriz de adjacência poderia também ser utilizada para adicionar *routing* ao protocolo TDMA. Por fim, a implementação de toda a arquitetura em *Kernel Space* seria um desafio interessante que traria benefícios em termos de controlo dos instantes de transmissão e do *overhead* dos mecanismos implementados.

Referências

- [1] Luís Almeida, Frederico Santos, and Luís Oliveira. *Structuring Communications for Mobile Cyber-Physical Systems*, pages 51–76. 01 2016. ISBN: 978-3-319-26867-5. doi:10.1007/978-3-319-26869-9_3.
- [2] Julius Degesys and Radhika Nagpal. *Towards desynchronization of multi-hop topologies*. pages 129–138, 10 2008. doi:10.1109/SASO.2008.70.
- [3] Julius Degesys, Ian Rose, Ankit Patel, and Radhika Nagpal. *Desync: Self-organizing desynchronization and tdma on wireless sensor networks*. In *IPSN 2007: Proceedings of the Sixth International Symposium on Information Processing in Sensor Networks*, pages 11–20, 05 2007. ISBN: 978-1-59593-638-7. doi:10.1109/IPSN.2007.4379660.
- [4] Renato Mirollo and Steven H. Strogatz. Synchronization of pulse-coupled biological oscillators. *Siam Journal on Applied Mathematics*, 50:1645–1662, 1990.
- [5] Clemens Mühlberger. *Design of a Self-Organizing MAC Protocol for Dynamic Multi-Hop Topologies*. doctoralthesis, Universität Würzburg, 2018.
- [6] Clemens Mühlberger and Reiner Kolla. Extended desynchronization for multi-hop topologies. 01 2009.
- [7] Luís Oliveira, Luís Almeida, and Daniel Mosse. *A clockless synchronisation framework for cooperating mobile robots*. pages 305–315, 04 2018. doi:10.1109/RTAS.2018.00038.
- [8] Luis Ramos Pinto. *Aerial Multi-hop Sensor Networks*. PhD thesis, Carnegie Mellon University, 2018.
- [9] Luis Ramos Pinto and Luís Almeida. *A robust approach to tdma synchronization in aerial networks*. *Sensors*, 18:4497, 12 2018. doi:10.3390/s18124497.
- [10] Venkatesh Rajendran, Katia Obraczka, and Jose Joaquin Garcia-Luna-Aceves. Energy-efficient collision-free medium access control for wireless sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 181–192, 2003.
- [11] Ill-Keun Rhee, Jaehan Lee, Jangsub Kim, Erchin Serpedin, and Yik-Chung Wu. Clock synchronization in wireless sensor networks: An overview. *Sensors*, 9(1):56–85, 2009.

-
- [12] Injong Rhee, Ajit Warrier, Mahesh Aia, and Jeongki Min. Z-mac: a hybrid mac for wireless sensor networks. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 90–101, 2005.
 - [13] Chaoyi Zheng, Shengchun Huang, Jibo Wei, and Qiangjian Dong. [Md-mac: A distributed tdma protocol based on desynchronization for multi-hop topologies](#). *Sensors*, 19:5102, 11 2019. doi:10.3390/s19235102.
 - [14] Eustathia Ziouva and Theodore Antonakopoulos. [Csma/ca performance under high traffic conditions: throughput and delay analysis](#). *Computer Communications*, 25(3):313–321, 2002. ISSN: 0140-3664. doi:[https://doi.org/10.1016/S0140-3664\(01\)00369-3](https://doi.org/10.1016/S0140-3664(01)00369-3).