Miguel Pocinho Arieiro

SUPORTE PARA REDES IOT COM ELEVADA DENSIDADE NAS TECNOLOGIAS 802.11AX E 802.11N

UNIVERSIDADE D COIMBRA
1 2 9 0

UNIVERSIDADE D
COIMBRA

Miguel Pocinho Arieiro

SUPORTE PARA REDES IOT COM ELEVADA DENSIDADE NAS TECNOLOGIAS 802.11AX E 802.11N

VOLUME 1

Relatório de Estágio no âmbito do Mestrado em Engenharia Informática, especialização em Sistemas Inteligentes orientada pelo Professor Doutor Bruno Miguel de Oliveira Sousa e à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática.

Setembro de 2021

Faculdade de Ciências e Tecnologia
Departamento de Engenharia Informática

SUPORTE PARA REDES IOT COM ELEVADA DENSIDADE NAS TECNOLOGIAS 802.11AX E 802.11N

Miguel Pocinho Arieiro

Relatório de Estágio no âmbito do Mestrado em Engenharia Informática, especialização em Sistemas Inteligentes orientada pelo Professor Doutor Bruno Miguel de Oliveira Sousa e apresentada à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática.

Setembro de 2021

UNIVERSIDADE D
COIMBRA

Resumo

Ao longo dos últimos anos, houve imensa inovação na área das tecnologias IoT (IoT 4.0, IoT Industrial, IoT para operações críticas). Estas inovações deram origem ao desenvolvimento de novos standards, os quais suportam grande número de dispositivos, throughputs mais elevados e apresentam mecanismos de poupança de energia otimizados.

Standards como o IEEE 802.11ax (conhecido como Wi-Fi version 6), NB-IoT e LTE-M são os mais conhecidos. As novidades nestes standards incluem mecanismos para suporte de Elevado número de conexões, ou seja, uma elevada densidade de conexões.

Este relatório pretende documentar a inovação suportada por estas tecnologias, particularmente para cenários críticos (por exemplo, combate a incêndios).

O primeiro semestre teve como foco a familiarização com a plataforma de simulação de redes ns-3. No segundo semestre recorreu-se a um algoritmo evolucionário para encontrar configurações otimizadas para cenários realistas de elevada densidade, como por exemplo as necessidades de comunicações no incêndio florestal de Pedrógão Grande, em 2017.

No final deste relatório, apresento uma análise comparativa entre duas versões do algoritmo evolucionário. A diferença entre estas versões reside na heurística composta utilizada por elas. O algoritmo recorre a estas heurísticas para analisar a performance dos diferentes indivíduos, obtendo-se, por isso, duas configurações optimizadas diferentes. Uma destas configurações favorece baixo consumo energético por bit, enquanto a outra favorece igualmente baixo consumo energético por bit e baixa taxa de perda de pacotes.

Palavras-Chave

IoT,

Computação Evolucionária

IEEE 802.11ax,

IEEE 802.11n,

Redes de elevada densidade

Abstract

In the past few years, IoT has been subject to innovation (IoT 4.0, Industrial IoT, IoT for critical operations). Such innovation leads to the development of new standards, that support a high number of devices, higher throughputs and that enable optimized energy saving mechanisms.

Standards like the IEEE 802.11ax (known as Wi-Fi version 6), NB-IoT and LTE-M are the most well-known. The innovation in such standards includes mechanisms to support a high number of connections, that is a high density of connections.

This report aims to document the innovation supported by these technologies, in particular for critical scenarios (e.g., firefighting).

The work in the first semester focused on the familiarization with the ns-3 simulation platform. During the second semester I implemented an evolutionary algorithm in order to search for optimized configurations in realistic high-density scenarios, such as the communication needs of Pedrógão Grande forest fire in 2017.

At the end of this report, I present a comparative analysis of two versions of the evolutionary algorithm. The difference between these versions is rooted on the composite heuristic used by them. The algorithm makes use of these heuristics in order to analyse the performance of the different individuals, therefore obtaining two different optimized configurations. One of these configurations favors low energy consumption per bit, while the other one equally favours low energy consumption per bit and low packet loss ratio.

Keywords

IoT,

Evolutionary Computation,

IEEE 802.11ax,

IEEE 802.11n,

High density networks

Agradecimentos

Este trabalho foi realizado sob a orientação do Professor Doutor Bruno Miguel de Oliveira Sousa, a quem gostaria de agradecer imenso por todo o apoio ao longo deste semestre.

Gostaria também de agradecer ao Professor Nuno Lourenço e ao Professor João Correia, por a sua disponibilidade e colaboração no algoritmo evolucionário.

Gostaria ainda de agradecer, aos meus amigos e à minha namorada, pois não só me apoiaram, ao longo dos últimos anos, como me "carregaram" em imensos jogos online.

Por fim, gostaria de agradecer imenso à minha família, particularmente ao meu irmão, por me animar constantemente, mesmo quando ando em baixo, e tornar os meus dias infinitamente melhores.

Índice

Acrónimos

ABSC	Ambulância de Socorro
ADAI	Associação para o Desenvolvimento da Aerodinâmica Industrial
AES	Advanced Encryption Protocol
AP	Access Point
BER	Bit Error Rate
BSS	Basic Service Set
CIoT	Cellular Internet of Things
CoAP	Constrained Application Protocol
CDF	Cumulative Distribution Function
DL	Downlink
DTLS	Datagram Transport Layer Security
DWN	Dynamic Wireless Network
ECC	Elliptic-curve cryptography
eDRX	Extended Discontinuous Reception
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IP	Internet Protocol
IPv6	Internet Protocol version 6
HEW	High Efficiency Wireless
HTTP	Hypertext Transfer Protocol
LPWA	Low-power Wide-area
LPWAN	Low-power Wide-area Network
LR-WPAN	Low-rate Wireless Personal Area Network
M2M	Many to Many
MA	Meio Aéreo
MAC	Medium Access Control
MCS	Modulation and Coding Scheme
MIMO	Multiple Input and Multiple Output
MQTT	Message Queuing Telemetry Transport
MTC	Machine-type communications
NAV	Network Allocation Vector
NB-IoT	Narrowband Internet of Things

O2O	One to One
OBSS-PD	Overlapping BSS/Preamble-Detection
OFDMA	Orthogonal Frequency-Division Multiple Access
PCO	Posto de Comando Operacional
PDR	Packet Delivery Ratio
PHY	Physical
PSM	Power Saving Mode
QoS	Quality of Service
RSA	Rivest-Shamir-Adleman
RSL	Received Signal Level
RSSI	Received Signal Strength Indicator
RU	Resource Unit
SINR	Signal to Interference plus Noise Ratio
SNR	Signal to Noise Ratio
SSL	Secure Sockets Layer
STA	Station
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TWT	Target Wake Time
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UL	Uplink
UL-MU	Uplink Multi User
VCOT	Veículo de Comando Tático
VFCI	Veículo Florestal de Combate a Incêndio
VLCI	Veículo Ligeiro de Combate a Incêndio
VM	Virtual Machine
VPCC	Veículo de Planeamento, Comando e Comunicações
VTGC	Veículo Tanque Grande Capacidades
VTTU	Veículo Tanque Tático Urbanos
VUCI	Veículo Urbano de Combate a Incêndio
WPAN	Wireless Personal Area Network

Lista de Figuras

Lista de Tabelas

Capítulo 1
Introdução

1.1 Contextualização

A internet das coisas (IoT) encontra-se cada vez mais presente no dia a dia das pessoas, tanto ao nível industrial e empresarial como ao nível pessoal, existindo um crescimento significativo na procura e oferta de serviços relacionados, tal como evidenciado pelo crescente número de ofertas de provedores de serviços em plataformas IoT na cloud. Apesar do crescente interesse neste tipo de tecnologia, existem diversas limitações, quer ao nível de recursos nos dispositivos, como na ausência de mecanismos eficientes de gestão de energia.

A IoT tem como base diversas tecnologias. Há standards estabelecidos, como é o caso do IEEE 802.15.4 [1], e standards modernos, de entre os quais se destacam o IEEE 802.11ax e o 5G, quer por serem tecnologias bastantes recentes como pelos mecanismos de suporte que oferecem para a criação de redes de elevada densidade e redução do consumo de energia. De entre estes mecanismos destacam-se particularmente a adoção de uma abordagem de acesso múltiplo por divisão de frequências ortogonais (OFDMA), *Uplink Multi User* (UL-UM), *Basic Service Set* (BSS) *colouring* e a implementação de *quiet time periods*, no 802.11ax [2][3], e ao nível do 5G, melhorias gerais na IoT celular (CIoT), especialmente ao nível da inclusão de LTE-M e *Narrowband IoT* (NB-IoT) [4][5].

No mundo atual existem diversos cenários de missão crítica que podem ser melhoradas com recurso à implementação de dispositivos IoT. Um destes cenários é o de combate a incêndios florestais e urbanos. Este cenário pode envolver a presença de imensos dispositivos sem fios numa determinada área (redes sem fios de elevada densidade), com elevada mobilidade. As tecnologias tradicionais possuem diversos problemas neste tipo de situações, má performance em localizações geográficas com elevado número de dispositivos [2], taxas de transferência reduzidas, e elevado consumo de energia [1] [2], o que impõe limitações neste tipo de cenários, em que as operações podem durar diversos dias. É nestes cenários críticos que as tecnologias IEEE 802.11ax e 5G se apresentam como possíveis soluções com os seus mecanismos de suporte a redes de elevada densidade, com baixo consumo de energia [1].

Tendo isso em conta ao longo deste projeto irei avaliar o desempenho dos standards IEEE 802.11ax e 5G como possíveis tecnologias de suporte para redes IoT de elevada densidade. Devido ao foco na avaliação de tecnologias Wi-Fi, com recurso a algoritmos evolucionários para otimização de configurações, não foi efetuado trabalho com cenários críticos com recurso a outras tecnologias como o 5G. Foi também elaborado um artigo com foco no algoritmo desenvolvido.

1.2 Objetivos

Parti para este estágio com os seguintes objetivos:

1. Avaliar de forma objetiva o suporte para redes IoT de elevada densidade na tecnologia 802.11ax;
2. Implementar via simulação cenários de redes IoT de elevada densidade com dispositivos IoT heterogéneos;
3. Avaliar o suporte real para redes IoT de elevada densidade;
4. Obter configurações otimizadas com recurso a algoritmo evolucionário;
5. Documentar dos resultados para efeitos de dissertação e publicação científica.

Para atingir os objetivos propostos é necessário conhecimento das tecnologias atuais, principalmente IEEE 802.11ax, bem como da plataforma de simulação.

Tendo presente as redes IoT, importa consolidar o conhecimento dos mecanismos de gestão de energia e de suporte a redes de elevada densidade presentes nesta tecnologia, de forma a aplicá-los ao decorrer deste estudo.

A implementação do cenário de simulação é efetuada no ns-3, pois aquando do início deste estágio, sendo um simulador *open-source*, tal apresentava o melhor suporte para a simulação destas tecnologias, não só nativamente como com diversas bibliotecas da comunidade. Já o algoritmo evolucionário encontra-se implementado em Python 3. Tal escolha prende-se com o facto desta linguagem ser amplamente utilizada na área de sistemas inteligentes e à sua fácil utilização.

Para finalizar, os dados provenientes da execução do algoritmo evolucionário e das simulações são recolhidos, analisados e documentados. Após a realização deste projeto espero obter boas configurações para redes de dispositivos IoT em cenários críticos de elevada densidade. Estas configurações deverão levar a uma melhoria significativa na performance dos dispositivos, tanto ao nível de transmissão de dados como do consumo energético.

1.3 Planeamento

De forma a facilitar o planeamento e organização, o estágio encontra-se subdivido em diversas tarefas, estando as mesmas distribuídas por dois semestres:

1º Semestre

1) Análise do estado da arte relativamente às tecnologias IEEE 802.11ax e 5G.
2) Análise do estado da arte relativamente a protocolos de comunicação em redes IoT (MQTT e CoAP).
3) Realização de estudos de simulação com cenários simples e com a tecnologia IEEE802.11n e IEEE 802.11ax. O objetivo desta tarefa prende-se com o conhecimento da plataforma de simulação a usar nas tarefas seguintes.
4) Elaboração do relatório intermédio.

2º Semestre

1) Elaboração de cenário de simulação de redes de elevada densidade com recurso a ns-3, utilizando IEEE 802.11ax e IEEE 802.11n.
2) Adoção e implementação de mecanismo inteligente adaptado à otimização da performance de dispositivos IoT em cenários críticos de elevada densidade.

3) Otimização de cenários realistas de redes IoT de elevada densidade com recurso a algoritmo evolucionário e simulações em ns-3 com as tecnologias IEEE 802.11ax e IEEE 802.11n.
4) Elaboração do relatório final.
5) Recolha de dados e elaboração de documentação para efeitos de dissertação e de publicação científica.

O seguinte digrama de Gantt (Figura 1) apresenta as diferentes áreas do trabalho realizado ao longo do estágio, organizadas por cores relativas às tarefas em que se enquadram, bem a duração das mesmas.

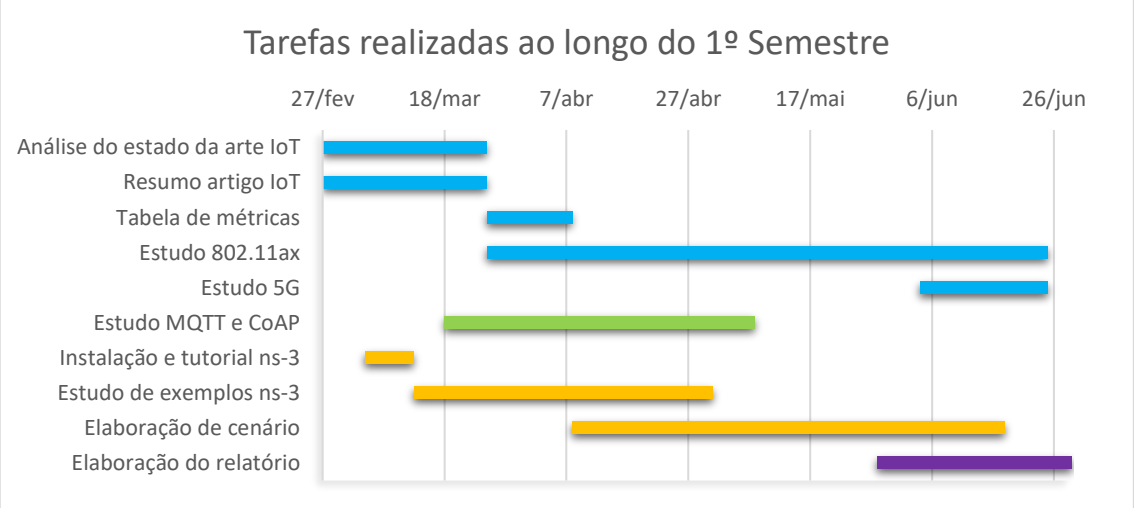


Figura 1 – Tarefas realizadas ao longo do 1º Semestre

O primeiro semestre foca-se principalmente na análise do estado de arte relativamente às tecnologias IEEE 802.11ax e 5G, e no estudo dos protocolos de comunicação MQTT e CoAP, terminando com a familiarização com o simulador de redes ns-3 através da criação de um cenário simples baseado na primeira tecnologia.

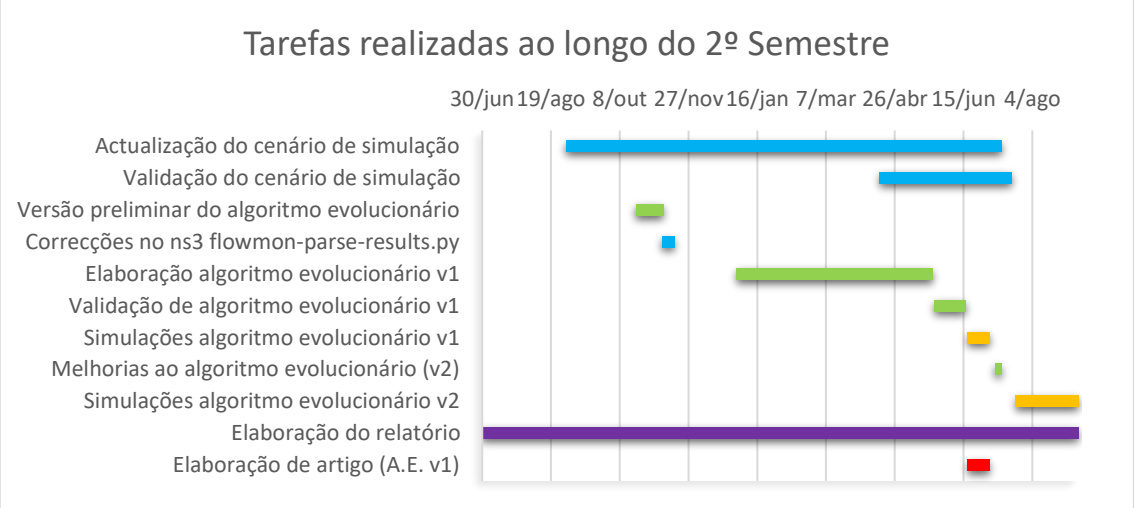


Figura 2 – Tarefas realizadas ao longo do 2º Semestre

Já o segundo semestre tem como objetivo a implementação de um algoritmo evolucionário. Com recurso ao mesmo, obtêm-se configurações otimizadas para um cenário crítico de elevada densidade (combate a incêndios florestais). Os resultados da simulação são utilizados na elaboração de documentação para efeitos de dissertação e de publicação científica.

1.4 Análise de risco do estágio

De forma a analisar e combater os possíveis riscos, elaborei uma matriz de risco. Na matriz apresentada na Tabela 1, classifico os diferentes riscos de acordo com a probabilidade de ocorrência e o possível impacto que teriam no desenvolvimento deste projeto.

Tabela 1 – Matriz de risco

		Impacto		
		Baixo	**Moderado**	**Elevado**
Probabilidade	**Provável**		R2	R6
	Possível		R3 R5	R1
	Improvável			R4

R1 – Problemas de saúde

R2 – Complexidade temporal de simulação extremamente elevada

R3 – Complexidade espacial de simulação elevada

R4 – Ocorrência de defeitos na versão final

R5 – Problemas de implementação de recolha de métricas do cenário de simulação

R6 – Limitações de hardware

De forma a mitigar o impacto e a diminuir a probabilidade de ocorrência dos diversos riscos foram tomadas diversas medidas. As principais medidas encontram-se definidas na Tabela 2.

Tabela 2 – Medidas de mitigação de riscos

Risco	**Medidas**
R1	Seguir medidas de prevenção de COVID19; Manutenção de cuidados habituais devido a problemas de saúde crónicos.
R2	Otimização de cenário de simulação;

	Implementação modular possibilitando a remoção fácil de diversos componentes opcionais; Implementação de métodos de configuração dos diversos componentes.
R3	Otimização de cenário de simulação; Implementação modular possibilitando a remoção fácil de diversos componentes opcionais; Implementação de métodos de configuração dos diversos componentes.
R4	Testagem e validação do cenário de simulação e do algoritmo evolucionário, ao longo do processo de desenvolvimento.
R5	Consulta da documentação do simulador de redes ns-3 e de diversos exemplos no Google Groups.
R6	Utilização de máquinas virtuais do Departamento de Engenharia Informática.

1.5 Contribuições

Deste projeto resultaram as seguintes contribuições:

- Algoritmo evolucionário, que determina configurações otimizadas para cenários de elevada densidade, com reduzido consumo de energia e baixa perda de pacotes.
- Cenário de simulação para redes 802.11ax e 802.11n, com suporte para diversas configurações
- Artigo científico para publicação na MSWIM que foi rejeitado. Foi acordado com o orientador proceder a melhorias para submissão na EvoApplications 2022.
- Repositório público GitHub com código e resultados de experimentação: https://github.com/MiguelArieiro/critical_IoT

Capítulo 2
Enquadramento

O combate a incêndios florestais possui diversos requisitos técnicos e operacionais, os quais culminam com a necessidade de estabelecer canais comunicação. Estes canais de comunicação são críticos para a missão, nomeadamente: combate a incêndios, segurança das pessoas, resgate, emergências, entre outros. Estes cenários críticos encontram-se agora suportados por veículos aéreos não tripulados (UAVs) e dispositivos IoT, de forma a assistir as equipas de emergência na sua missão [29, 30, 31, 32]

A disponibilidade do serviço é outro requisito fortemente associado com as infraestruturas de comunicação. Estas podem depender de diversas tecnologias (por exemplo, LoRaWAN, 5G, LTE M etc). Tal infraestrutura requere elevados níveis de resiliência, particularmente em situações de desastre que possam impactar a disponibilidade do serviço. Devido a isto, a implementação de redes dinâmicas wireless (DWNs) recorrendo a diferentes tecnologias (por exemplo IEEE 802.11ax e 5G) possibilita a comunicação entre dispositivos (D2D), sem o suporte de uma infraestrutura, ou em condições de elevada densidade, nas quais os dipositivos móveis no local, excedam as capacidades da torre celular [33].

Neste contexto, sugere-se a disponibilização de diversas DWN, para comunicações de segurança pública. Existem diversas abordagens. Uma delas consiste na implementação de uma BS que cobre todo o cenário de missão. No entanto, esta abordagem não providencia QoS adequada para comunicação em tempo real, em situações de elevada densidade, ou seja, situações com elevado número de dispositivos conectados por km^2. Outra abordagem considera a disponibilização de um BS móvel e UAVs ou dispositivos em veículos, de forma a estabelecer uma rede cooperativa. Esta arquitetura melhora tanto a capacidade do sistema, como a cobertura. Por esta razão, as soluções atuais seguem um modelo híbrido, lidando com os problemas de densidade com recurso a arquiteturas *small-cell*, as quais dependem de múltiplas tecnologias, como LTE e Wi-Fi [35]

2.1 IoT

Internet das coisas, ou IoT, é um conceito resultante da convergência de diferentes tecnologias aos longo da história, tendo sofrido um crescimento tremendo na última década. A sua definição moderna consiste na interligação de vários dispositivos de forma a existir comunicação entre eles, geralmente sem envolvimento humano direto. Espera-se que até ao final de 2020, existam mais de 50 mil milhões de dispositivos conectados globalmente [44].

Este tipo de tecnologia possui diversas aplicações, tanto ao nível do consumidor privado como empresarial. Ao nível do consumidor, o IoT encontra-se fortemente associada ao conceito de *smart home*, e a dispositivos controlados via smartphone, já a um nível empresarial as aplicações são extremamente diversas, como, por exemplo, aplicações industriais e aplicações médicas (*smart healthcare*) [4].

Entre as diferentes áreas da IoT, uma área tem vindo a ganhar extrema relevância: IoT crítica. Esta área da IoT consiste em casos de utilização críticos, onde se requer não só elevada fiabilidade e completa disponibilidade dos serviços, como, em determinados casos, reduzida

latência [26]. Alguns destes cenários são aplicações médicas remotas, condução autónoma [26], ou monitorização de sensores em tempo real.

2.2 Tecnologias

Existem diversos tipos de dispositivos IoT, tendo por isso diferentes limitações. No tipo de cenários de elevada densidade que pretendo estudar, os equipamentos não só seriam afetados pelas limitações energéticas características dos mesmos [1] como, também, pelas limitações das taxas de transferência das tecnologias de comunicação em si, e interferência por parte de outros dispositivos devido ao elevado número de dispositivos presentes.

Felizmente, existem diversos standards modernos. Estes standards não só implementam mecanismos de redução do consumo de energia, como suportam redes de elevada densidade. Entre estes standards destacam-se IEEE 802.11ax e o 5G.

2.2.1 IEEE 802.15.4

Apesar do foco deste projeto ser o 802.11ax e o 5G, não posso deixar de analisar uma das bases das tecnologias IoT industriais: o standard para low-rate wireless personal area networks (LR-WPAN) IEEE 802.15.4. Este standard tem elevada gestão energética, sendo tal feito alcançado através da especificação da camada física (PHY) e da camada de *medium access control* (MAC) [1] com este objetivo. Deste design resulta um standard com baixo consumo de energia, mas taxas de transferência reduzidas.

Esta tecnologia foi implementada em diferentes standards industriais com ampla adoção mundial, destacando-se em particular ZigBee, WIA-PA, WirelessHART e ISA100.11a. Destes apenas os primeiros dois implementam na totalidade a camada MAC, tomando, os restantes, decisões de forma a otimizar a camada MAC para os seus cenários de utilização específicos [1].

2.2.2 IEEE 802.11ax

IEEE 802.11ax também conhecido como *High Efficiency Wireless* (HEW) ou comercialmente como Wi-Fi 6 [6] é um standard wireless que opera nas frequências entre 1GHz e 6GHz. Os standards 802.11 anteriores geralmente focavam-se principalmente no aumento de taxas de transferência, mas nesta emenda o foco em taxas de transferência foi algo secundário. Ao desenvolver esta nova versão do standard 802.11 houve um foco no desempenho das redes de elevada densidade. Redes de elevada densidade, consistem em redes com elevado número de estações (STA) e múltiplos pontos de acesso (AP) concentrados numa pequena área geográfica, existindo por isso sobreposição de canais [2]. Apesar do foco neste tipo de redes, existiu também uma preocupação ao nível do consumo energético, resultando na implementação de mecanismos de poupança de energia [2][4]. Estes mecanismos são extremamente úteis, não só para dispositivos portáteis genéricos, como smartphones, mas particularmente em pequenos dispositivos IoT, a utilização dos quais está frequentemente associada a limitações energéticas [1].

Tabela 3 – Standards Wi-Fi da família IEEE 802.11

IEEE Standard	Taxa de transferência máxima	Banda RF	Técnicas	Largura de canal	Alcance
802.11	1 a 2 Mbps [8][14][15]	2.4 GHz [8][14]	DSSS [8]	20 MHz [8] [14]	100m [8]
802.11b	11 Mbps [8][14][15]	2.4 GHz [8][14]	DSSS [8]	20 MHz [8] [14]	100m [8]
802.11a	54 Mbps [8][14][15]	5 GHz [8][14]	OFDM [8]	20 MHz [8] [14]	80m [8]
802.11g	54 Mbps [8][14][15]	2.4 GHz [8][14]	DSSS, OFDM [8]	20 MHz, 40 MHz [8][14]	100m [8]
802.11n (Wi-Fi 4)	600 Mbps [8][15]	2.4 GHz & 5 GHz [8][14]	OFDM [8], SU- MIMO [14]	20 MHz, 40 MHz [8][14]	140m [8]
802.11p	27 Mbps [8]	5.9 GHz [8]	MIMO, Frame aggregation [8]	10 MHz [8]	1km [8]
802.11ac (Wi-Fi 5)	6.9 Gbps [2][15]	5 GHz [14]	OFDM [8], MU-MIMO (DL) [14][15]	20 MHz, 40 MHz, 80 MHz, 160/80+80 MHz[14][15]	160m [8]
802.11ah (Wi-Fi HaLow)	347 Mbps	900 MHz	OFDM [25]	1 MHz, 2MHz, 4MHz, 8MHz, 16 MHz	1km [25]
802.11ax (Wi-Fi 6)	9.6 Gbps [2]	1 a 6 GHz [2]	OFDMA, MU-MIMO [2]	20 MHz, 40 MHz, 80 MHz, 160/80+80 MHz [2][14]	>90m [3]
802.11be (Wi-Fi 7)	30 Gbps [23][24]	1 GHz a 7.25 GHz [23]	OFDMA, MU-MIMO [23][24]	20 MHz, 40 MHz, 80 MHz, 160/80+80 MHz, 240/160+80 MHz, 320/160+160 MHz [23][24]	-

Analisando a Tabela 3, é possível constatar que, apesar de não ser um dos objetivos principais no desenvolvimento do standard 802.11ax, houve um aumento considerável da taxa de transferência máxima suportada para 9.6 Gbps. Este valor traduz um aumento de ~37% relativamente ao standard anterior (IEEE 802.11ac) [2][15]. Esta melhoria, bem como as melhorias ao nível da criação de redes de elevada densidade, é uma consequência da adoção de uma abordagem OFDMA e da implementação de Uplink Multi User (UL-MU) [2] [3].

2.2.2.1 OFDMA

Tanto OFDM como OFDMA consistem na divisão do sinal em frequências ortogonais de forma a aproveitar melhor a rede diminuindo a interferência entre *subcarriers*, ou tons, adjacentes, podendo o OFDMA ser visto como uma versão de OFDM para vários utilizadores.

Como muitos dos seus predecessores, o standard anterior (802.11ac) utilizava OFDM para diminuir os níveis de interferência nas suas comunicações. A utilização desta tecnologia em conjunção com a adoção de canais extremamente largos (80MHz e 160MHz) [3], afetava negativamente as taxas de transferência médias. Ao utilizar OFDMA é efetuada uma divisão em *subcarriers* mais estreitos do que com OFDM, sendo os tons adjacentes agrupados em *resource units* (RU) [2] [3].

Uma RU pode conter 26, 52, 106, 242, 484, 996 ou 2x996 tons, sendo a banda de 20MHz correspondente a uma RU de 242 tons, a de 40MHz a 484 tons, a de 80MHz a 996 tons e a de 160MHz (80MHz + 80MHz) a duas RUs de 996 tons, podendo quando necessário subdividir as mesmas [2][3]. Nos casos em que uma estação (STA) apenas necessita de transmitir uma pequena quantidade de dados, seria ineficiente a atribuição de RUs típicas de taxas de transferência elevadas, por esse motivo é-lhe antes atribuída uma RU estreita.

Estas RUs são atribuídas aos diversos utilizadores consoante as suas necessidades, podendo desta forma realizar-se a transmissão simultânea por parte de diversos utilizadores. Tal deve-se à redução de interferência associada à divisão do sinal em frequências ortogonais, ou seja, melhor nível de SINR, o que leva a maior t*hroughput* e maior eficiência da rede.

2.2.2.2 Outras funcionalidades

Outras adições ao novo standard wireless são: melhorias no BSS colouring e no Network Allocation Vector (NAV), operação de *microsleep*, redesenho de *Target Wake Time* (TWT) e gestão de energia (*oppurtunistic power save*).

BSS colouring no 802.11ax trata-se de uma expansão da mesma funcionalidade presente no 802.11ac, e permite a distinção de frames inter e intra-BSS, mesmo quando o seu conteúdo se encontra corrompido. Para tal, atribui uma "cor" a cada BSS, a qual será utilizada para distinguir pacotes de diferentes BSS, diminuindo o número de colisões e interferência.

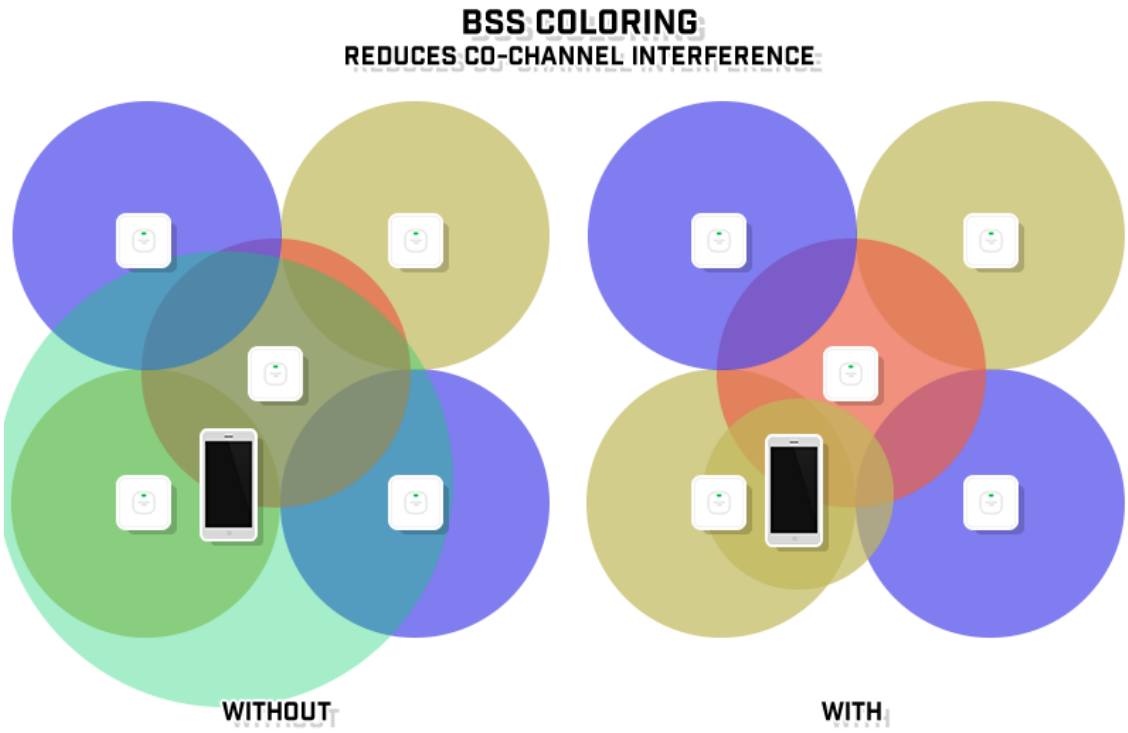


Figura 3 – Efeitos de BSS Colouring [7]

A operação de *microsleep*, permite a uma STA desligar temporariamente o seu rádio durante a duração de uma transmissão de um BSS sobreposto [1]. O TWT consiste na negociação prévia entre STA e AP de períodos de tempo para transmissão e receção de dados, permitindo que as estações estejam ativas apenas durante esses períodos e permaneçam "adormecidas" durante o restante tempo, poupando imensa energia.

Todas estas funcionalidades são de extrema importância para a criação de redes IoT de elevada densidade, pois não só diminuem consideravelmente o consumo de energia na utilização de Wi-Fi, como melhoram significativamente taxas de transferência em redes com elevado número de dispositivos.

2.2.3 5G

O 5G é uma evolução das tecnologias atuais baseadas em redes 4G. Este standard é desenvolvido pelo 3rd Generation Partnership Project (3GPP) [4][5] e introduz otimizações em diversos aspetos. Estas vão desde uma melhor utilização do espetro de frequências, a um conjunto de serviços e tecnologias que facilitam o desenvolvimento de novos serviços e um número ligações elevado.

No contexto de IoT, o 5G inclui otimizações relevantes ao nível de IoT celular (CIoT) [4][5], tais como incorporação das tecnologias de redes sem fios de baixa energia Narrowband IoT (NB-IoT) [4][5] e de LTE-M [4], as quais descrevo mais pormenorizadamente nas subsecções seguintes, e um aumento da eficiência energética [4], entre outras otimizações.

O desenvolvimento destes standards para redes celulares de dispositivos IoT por parte do 3GPP começou ainda em 2011 com o 3GPP Release 10, sendo o primeiro perfil definido, em

Março de 2015, no 3GPP Release 12 (LTE Cat-0) [4]. Desde então foram desenvolvidos novos standards – LTE-M e NB-IoT – e efetuadas diversas alterações e otimizações dos mesmos.

Tanto NB-IoT como LTE-M são tipos de *low-power wide-area network* (LPWAN) [4][5], ou seja, redes sem fios, de baixa energia utilizadas em comunicações com taxas de transferência reduzidas. Este tipo de redes é frequentemente utilizado com dispositivos com necessidade de elevada eficiência energética, como é o caso de imensos dispositivos IoT [1][4][5].

Ambas as tecnologias apresentam funcionalidades como *Extended Discontinuous Reception* (eDRX) e *Power Saving Mode* (PSM) [22], as quais lhes permitem hibernar durante extensos períodos de tempo e reduzir significativamente o consumo de energia. Além disso, tanto LTE-M como NB-IoT operam em bandas LTE licenciadas, o que garante a existência de pouca interferência e oferece elevada qualidade de serviço (QoS) [4].

2.2.3.1 NB-IoT

Narrowband Internet of Things, ou NB-IoT, é uma tecnologia LPWAN baseada em redes celulares que oferece cobertura melhorada para um grande número de dispositivos com *throughput* reduzido com baixo consumo de energia, baixo custo, bem como baixa sensibilidade a latência [16], como é o caso de sensores.

Inicialmente definido no 3GPP Release 13 como LTE Cat-NB1[4][16], este standard faz uso de uma largura de banda de 180 kHz (200 kHz *carrier*)[4][5] e oferece *taxas de transferência* até 26-27.3 kbps *downlink* e 32.25-62 kbps *uplink* [5][22] com latência superior a 1s [21].

Em 2017, foi publicado o 3GPP Release 14, onde se encontra definida uma versão melhorada desta tecnologia, a LTE Cat-NB2, tendo NB-IoT sofrido ligeiras melhorias nos últimos anos. Uma das principais novidades desta versão é um aumento considerável da taxa de transferência para 80-127 kbps *downlink* e 105-159 kbps *uplink* [22].

2.2.3.2 LTE-M

LTE-M é uma tecnologia LPWAN baseada em redes celulares destinada a *Machine-type Communication* (MTC) [5].

Inicialmente definida no 3GPP Release 13 como LTE Cat-M1 [4][5], esta especificação recorre a uma largura de banda de 1.08 MHz (1.4MHz carrier) [17], oferecendo até 1Mbps *downlink* e *uplink* [4][17] e latência de inferior a 150ms em modo de operação normal [21].

Em 2017, o 3GPP Release 14 apresentou uma versão melhorada sob o nome LTE Cat-M2 [17]. Esta versão suporta taxas de transferência de até 4Mbps *downlink* e 7Mbps *uplink* [17]. Posteriormente foram acrescentadas diversas funcionalidades que contribuíram para uma maior diminuição da latência e do consumo energético [5].

2.2.4 Comparação

Após este capítulo é possível constar que existem diferenças significativas entre os diversos standards analisados, desde as frequências utilizadas às diversas funcionalidades disponibilizadas, entre outros atributos destacados na Tabela 4. Acredito que tanto o IEEE 802.11ax como as tecnologias 5G têm o seu lugar no mundo das tecnologias IoT, sendo, no entanto, indicadas para diferentes situações.

Tabela 4 – Tecnologias 5G vs IEEE 802.11ax

Standard	Taxa de transferência máxima	Banda RF	Técnicas	Largura de canal	Alcance
IEEE 802.11ax	9.6 Gbps [2]	1 a 6 GHz [2]	OFDMA, MU-MIMO [2]	20 MHz, 40 MHz, 80 MHz, 80+80 MHz [2][14]	>90m [3]
LTE Cat-NB1	26 kbps (DL)[5][21] 32-66 kbps (UL)[5][21]	Frequências LTE licenciadas* [4][5]	SISO [4]	180 kHz (200 kHz carrier) [4][5][16]	40km [16]
LTE Cat-NB2	80-127 kbps (DL) [22] 105-159 kbps (UL) [22]	Frequências LTE licenciadas* [4][5]	SISO [4]	180 kHz (200 kHz carrier) [16]	40-120km [16]
LTE Cat-M1	1 Mbps (DL e UL) [4][17]	Frequências LTE licenciadas* [4][5]	SISO [4]	1.4 MHz [4][5][17]	100 km [17]
LTE Cat-M2	4 Mbps (DL) 7 Mbps (UL) [17]	Frequências LTE licenciadas* [4][5]	SISO [4]	5 MHz [17]	100 km [17]

*Frequências utilizadas variam consoante região

LTE-M e NB-IoT são tecnologias 5G indicadas para dispositivos que não necessitem de transmitir muita informação, e/ou necessitem de consumo energético extremamente reduzido. Ambas as tecnologias apresentam funcionalidades de poupança de energia do 5G. No entanto, o LTE-M foca-se em latência reduzida e taxas de transferência superiores, já o NB-IoT leva o conceito de baixo consumo de energia ao limite. Estas escolhas de design por parte do NB-IoT culminam num standard com consumo de energia muito reduzido, mas que apenas suporta taxas de transferência igualmente baixas, fazendo uso de canais extremamente pequenos (180-200 kHz) [21].

O IEEE 802.11ax disponibiliza taxas de transferência extremamente elevadas, em frequências abertas, com baixa latência, mas está associado a menor alcance e custo mais elevado do que tecnologias 5G.

Concluindo, NB-IoT parece ser mais indicado para transmissão de informação por parte de pequenos sensores, enquanto que LTE-M será melhor para dispositivos com mais recursos disponíveis, que necessitem de latência reduzida e/ou taxas de transferência um pouco superiores, já o IEEE 802.11ax aparenta ser melhor para transmissão de grandes quantidades de dados com alcances relativamente pequenos (até 90m sem modo de alcance extenso [3]) , embora apresente excelente suporte para redes de elevada densidade.

2.3 Protocolos de comunicação

No mundo de atual dos dispositivos IoT, de forma a haver transmissão de dados, são necessárias soluções diferentes do típico modelo cliente servidor [1]. Estas soluções devem ser seguras, fiáveis e eficientes. É aí que entram os protocolos de comunicação. Os protocolos MQTT e CoAP são amplamente utilizados na comunicação entre dispositivos IoT. Por esse motivo, irei cingir-me aos mesmos.

2.3.1 MQTT

O Message Queuing Telemetry Transport (MQTT) é um protocolo de mensagens publish/subscribe para comunicação *many-to-many*. Este protocolo segue um modelo cliente/servidor (broker), representado na Figura 4, sendo a comunicação efetuada através de uma ligação TCP, a qual poderá ser encriptada com recurso a SSL/TLS [10].

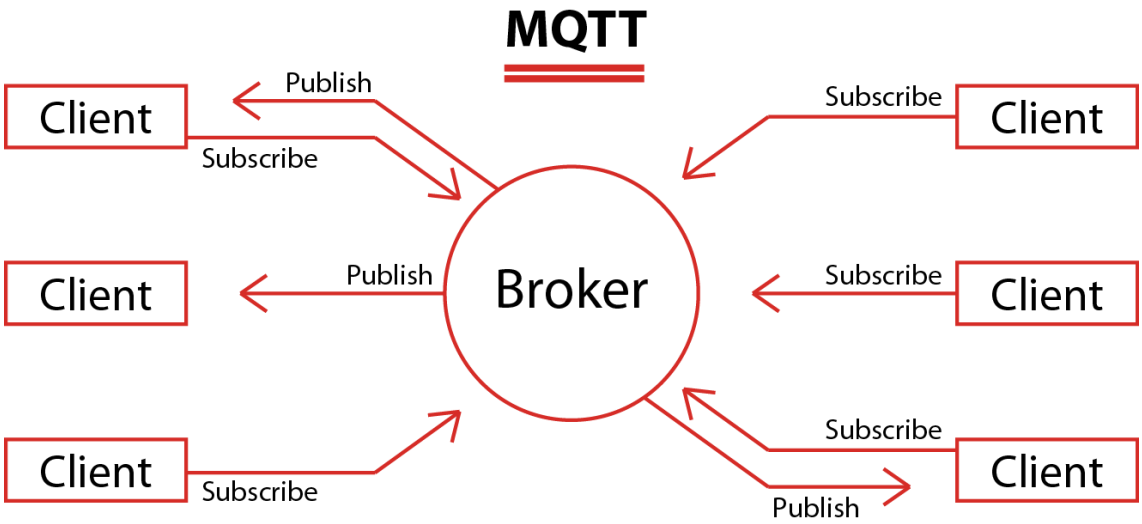


Figura 4 – Modelo MQTT [20]

Neste protocolo os clientes subscrevem tópicos, nos quais as mensagens são publicadas. Estes tópicos funcionam como um sistema de ficheiros hierárquico, sendo as mensagens publicadas num nível acessíveis a partir de níveis superiores.

Este protocolo suporta mensagens persistentes, armazenando a mensagem mais recente, e o envio automático de mensagens quando um cliente se desconecta do broker, e apresenta três níveis de qualidade de serviço [12][13]:

- QoS0: *At most once*
- QoS1: *At least once*
- QoS2: *Exactly once*

QoS0 é o modo de transferência mais rápido dos três. A mensagem é enviada uma única vez, não existindo qualquer confirmação do seu envio através da rede. Esta mensagem poderá perder-se caso o servidor falhe ou o cliente se encontre desconectado, pois o MQTT não reenvia a mensagem a qualquer cliente que se encontrasse desconectado aquando do envio inicial. Este nível de qualidade de serviço também é conhecido como "*fire and forget*".

Mensagens em modo QoS1 são garantidamente entregues, pelo menos uma vez. Este tipo de mensagens é armazenado localmente pelo remetente, até este receber a confirmação de que a mesma foi publicada pelo destinatário. Esta poderá ser entregue múltiplas vezes caso exista uma falha que impeça a receção do *acknowledgement* por parte do remetente.

O nível mais elevado de qualidade de serviço (QoS2) é, também, o mais lento dos três. Este modo de envio de mensagens garante que estas são entregues exatamente uma vez. No QoS2, tal como no Q0S1 as mensagens são armazenadas localmente pelo remetente, até o mesmo receber a confirmação da publicação desta por parte do destinatário.

Ao nível da segurança, o MQTT não só suporta a encriptação da ligação com SSL/TLS, como autenticação com recurso a nome de utilizador e password [10], ou certificados X.509 [11].

O MQTT apresenta também algumas desvantagens como, por exemplo, o requisito de uma conexão TCP constante ao broker e a utilização de grandes *strings* como nome de tópicos. Felizmente, ambos os problemas são resolvidos no protocolo MQTT-SN, o qual conta com indexação dos nomes dos tópicos e utiliza UDP [10].

2.3.2 CoAP

O Constrained Application Protocol (CoAP) é um protocolo de transferência de documentos, com foco em comunicações *one-to-one*. Este protocolo é baseado em UDP com ordenamento de datagramas implementado na pilha protocolar, e suporta tanto UDP broadcast como multicast para endereçamento. A ligação poderá ser encriptada com recurso a IPSec ou DTLS, tanto com RSA+AES como com ECC+AES [10].

Observando a Figura 5, é possível constatar que tal como o MQTT o CoAP segue um modelo cliente/servidor, mas não possui brokers nem tópicos. No CoAP os clientes requisitam mensagens aos servidores, os quais lhes enviam respostas. Os clientes podem utilizar GET, PUT, POST e DELETE.

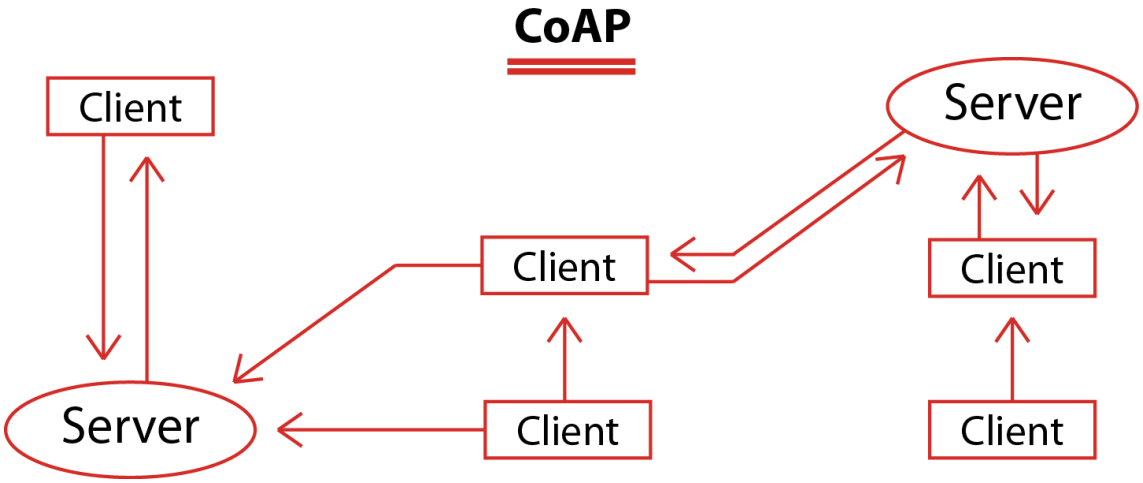


Figura 5 – Modelo CoAP [20]

De forma a manter ligações entre cliente e servidor, este protocolo apresenta uma *flag* observe, a qual permite que o servidor continue a responder após a resposta inicial.

O CoAP suporta negociação e descoberta de conteúdo, e apresenta dois níveis de qualidade de serviço:

- *Non-confirmable messages*
- *Confirmable messages*

O primeiro nível consiste no envio das mensagens sem qualquer confirmação da sua receção (*fire and forget*), já as "*confirmable messages*" consistem no envio de mensagens e na confirmação da receção por parte do destinatário, através de envio de um "*acknowledgement*" ao remetente.

Este protocolo de comunicação apresenta como principal desvantagem a existência de possíveis problemas de NAT. Este problema deve-se aos sensores serem frequentemente o servidor. De forma a funcionar por detrás de NAT, os dispositivos teriam primeiro de enviar um pedido ao servidor de forma a associar ambos os dispositivos. A melhor forma de lidar com este cenário é a utilização de *tunneling* ou de IPv6.

2.3.3 Comparação

Observando a Tabela 5, podemos constatar que existem bastantes diferenças em ambas os protocolos de comunicação estudados, apresentando ambos diferentes funcionalidades.

Tabela 5 – MQTT vs CoAP [20]

Funcionalidades	**MQTT**	**CoAP**
Protocolo de transporte [10][20]	TCP	UDP
Modelo de comunicação [5][10][20]	*Publish-Subscribe*	*Request-Response*
Tipologia [10][20]	*Many-to-many*	*One-to-one*
Fiabilidade [10][12][13][20]	QoS0: *At most once* QoS1: *At least once* QoS2: *Exactly once*	*Non-confirmable messages* *Confirmable messages*
Segurança [10][11][20]	SSL/TLS Autenticação com X.509 ou username+password	IPSec, DTLS
Outras [10][19][20]	Bom para conexões remotas	RESTful Latência reduzida Problemas de NAT

MQTT é um protocolo M2M, o que o torna particularmente útil quando é necessário passar as mesmas informações para imensos dispositivos. Por sua vez o CoAP é primariamente um protocolo *one-to-one* o que o torna mais útil para transferência de informação de estado. Uma vez que o MQTT mantem uma conexão aberta, este é mais indicado do que CoAP para ambientes NAT, mas tal também acarreta um maior consumo de energia.

Uma das vantagens do CoAP é o suporte para negociação e descoberta de conteúdo devido à implementação RESTful, enquanto o MQTT requer o conhecimento prévio do formato das mensagens.

Concluindo, ambos os protocolos têm as suas vantagens e desvantagens, sendo ambos úteis em determinados cenários.

2.4 Mecanismos de otimização

De forma a otimizar as configurações utilizadas optei pela utilização de um algoritmo evolucionário. Este tipo de algoritmo é bastante utilizado para otimização de problema com elevada complexidade computacional, devido à forma como consegue encontrar boas soluções, em tempo útil. Isto deve-se ao facto de não verificar todas as soluções possíveis, mas sim "melhorar" as possíveis soluções ao longo de diversas gerações. Para tal, faz uso a sistemas de seleção e modificação de soluções ao longo de diversas iterações (gerações), com recurso a aleatoriedade controlada.

2.4.1 Algoritmos evolucionários

Um algoritmo evolucionário é um tipo de algoritmo que visa encontrar as boas soluções para problemas complexos, em tempo útil. Para tal, estes algoritmos inspiram-se na teoria da evolução de Charles Darwin, descrita no seu famoso livro "The origin of species" [43]. Segundo esta teoria, os indivíduos de uma espécie competem não apenas com outras espécies, mas também com elementos da mesma espécie, o que afeta a sua taxa de sobrevivência [42].

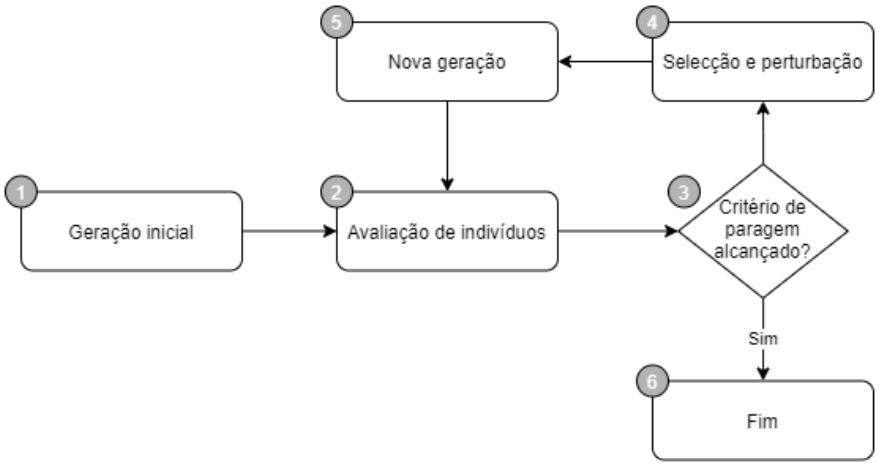


Figura 6 – Implementação genérica de algoritmo evolucionário

Em essência, um algoritmo evolucionário consiste na simulação de uma população de indivíduos, ao longo de várias gerações, conforme ilustrado na Figura 6. Cada indivíduo codifica uma possível solução para o problema. Geralmente, em cada geração, esta é modificada, gerando-se novos indivíduos a partir dos existentes, e posteriormente avaliada, com recurso a determinadas heurísticas. No final, seleciona-se o melhor indivíduo, ou seja, a melhor solução alcançada ao longo das diversas gerações.

Capítulo 3
EvoMCS

3.1 Descrição

De forma a estudar o desempenho das tecnologias IEEE 802.11ax e IEEE 802.11n em cenários de elevada densidade, irei recorrer à simulação baseada em cenários críticos reais. Ou seja, cenários que necessitam não só de elevada fiabilidade e completa disponibilidade dos serviços, como, por vezes, de reduzida latência.

3.1.1 Incêndios

Todos os anos, com a subida da temperatura, vêm os grandes incêndios florestais. O combate a estes incêndios de grandes dimensões pode ser extremamente difícil e complexo, chegando a envolver centenas de veículos e milhares de operacionais, em áreas relativamente extensas, com necessidades de comunicação e partilha de informação. É, por esse motivo, essencial a manutenção de linhas de comunicação e partilha de informação ao longo de áreas na ordem das centenas de hectare, com pontos de elevada concentração de operacionais.

Um dos maiores incêndios de que há registo, em Portugal, foi o incêndio de Pedrógão Grande, em 2017, o qual foi bastante estudado. Este incêndio é caracterizado, não só pela sua extensão, e número de bombeiros e veículos envolvidos, mas também, pela elevada perda de vidas e falhas de comunicação. Segundo um estudo realizado pela Associação para o Desenvolvimento da Aerodinâmica Industrial (ADAI) este incêndio envolveu centenas de veículos e operacionais (Tabela 6), sendo, por isso, um bom ponto de partida para a simulação de um cenário de combate a grandes incêndios. Simulação esta que pretende estudar e viabilizar standards como o IEEE 802.11ax, como tecnologias de suporte a missões críticas.

A manutenção de linhas de comunicações é algo de extrema importância no combate a grandes incêndios florestais, sendo por vezes difícil devido à ausência de infraestrutura dedicada. O IEEE 802.11ax, não só não requer infraestrutura pré-existente no terreno, como apresenta funcionalidades de suporte a cenários com elevada concentração de dispositivos, bem como mecanismos de poupança energética, sendo, por isso, um bom candidato a tecnologia de comunicações para este tipo de cenários.

Tabela 6 – Resumo dos meios utilizados no combate ao incêndio de Pedrógão Grande [27]

Incêndio	PCO	VOPE	VFCI	VTTU	VCOT	MA	Outros	Operacionais
Gois	1		±100	±40	±80	±10	±120	±1200
Moninhos Cimeiros (Figueiró dos Vinhos)			±6		±4	±2	±7	±50
Cabeças (Alvaiázere)		1	±9	±9	±7	±1	±9	±80
Pardieiros (Penela)			±15	±5	±15	±5	±20	±120
Escalos – Fundeiros (juntou-se a PG)	1		±18	±4	±8	±2	1 (ABSc)	±120
Regadas – (juntou-se a PG)			±50	±20	±50	±12	±12 (ABSc) ±10 (VUCI)	±600

MA	MEIO AÉREO
PCO	POSTO DE COMANDO OPERACIONAL
VCOT	VEÍCULO DE COMANDO TÁTICO
VFCI	VEÍCULO FLORESTAL DE COMBATE A INCÊNDIO
VLCI	VEÍCULO LIGEIRO DE COMBATE A INCÊNDIO
VPCC	VEÍCULO DE PLANEAMENTO, COMANDO E COMUNICAÇÕES
VTGC	VEÍCULO TANQUE GRANDE CAPACIDADES
VTTU	VEÍCULO TANQUE TÁTICO URBANOS
VUCI	VEÍCULO URBANO DE COMBATE A INCÊNDIO

De forma a simular um cenário de combate a incêndios irei considerar o seguinte:

- Cada veículo agrega a informação dos seus operacionais
- Cada veículo encontra-se representado por um AP
- Cada operacional e os seus sensores encontra-se representado por uma STA

Estas considerações têm em conta a hierarquia estabelecida no teatro de operações, na qual existem elementos responsáveis pela comunicação e gestão de operacionais no terreno. Com base nestas considerações, elaborei o cenário representado de forma simplificada na Figura 7.

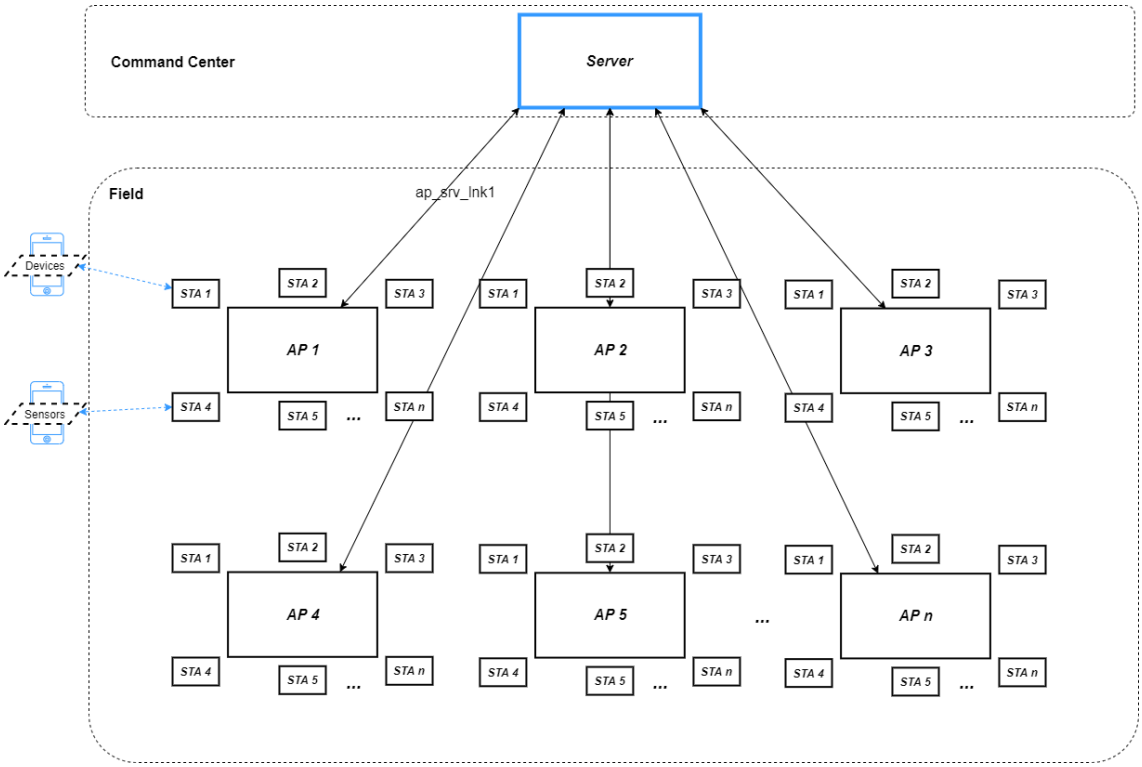


Figura 7 – Configuração do cenário de incêndio florestal

Nesta figura as STA representam o conjunto de cada operacional e os dispositivos por ele transportados, os quais transmitem a informação aos veículos (AP), sendo esta retransmitida a um servidor no centro de comando. Uma vez que o foco principal do estudo é o consumo de energia por parte dos dispositivos IoT, de forma a diminuir possíveis defeitos no cenário de simulação, omitiu-se o servidor central, pois o mesmo não teria qualquer impacto nestas medições.

3.2 Algoritmo Evolucionário

Um dos principais objetivos deste projeto consiste na avaliação do desempenho de diferentes tecnologias, nomeadamente IEEE 802.11ax e IEEE 802.11n. No entanto, existem inúmeras configurações para cada tecnologia (Tabela 7), tornando a execução de simulações, com todas estas configurações, algo impossível, em tempo útil. Por esse motivo, foi elaborado um algoritmo evolucionário baseado em operadores de mutação.

Tabela 7 – Parâmetros de indivíduos do algoritmo evolucionário

Standard	Frequência (GHz)	Channel Width (MHz)	MCS	Guard Interval (ns)	RTS	Estended Block	OBSS/ PD	UDP/TCP
IEEE 802.11ax	5, 6	20, 40, 80, 160	[0, 11]	800, 1600, 3200	0,1	0, 1	0, 1	0, 1
IEEE 802.11n	5	20, 40, 80, 160	[0, 23]	0 (800), 1 (400)	0,1			0, 1

O algoritmo utilizado encontra-se descrito na Figura 8. Nesta implementação parte-se de uma população inicial de indivíduos gerados aleatoriamente (passo 1), cujo genoma codifica uma configuração possível do sistema. Estes indivíduos são utilizados para gerar descendentes com base num operador de mutação (passos 5 e 6). Tanto a população inicial como os seus descendentes são avaliados num cenário de simulação, de acordo com uma heurística definida na subsecção seguinte. Após cada ciclo, os melhores indivíduos da população original são combinados com os descendentes e alguns indivíduos gerados aleatoriamente, formando a população da geração seguinte.

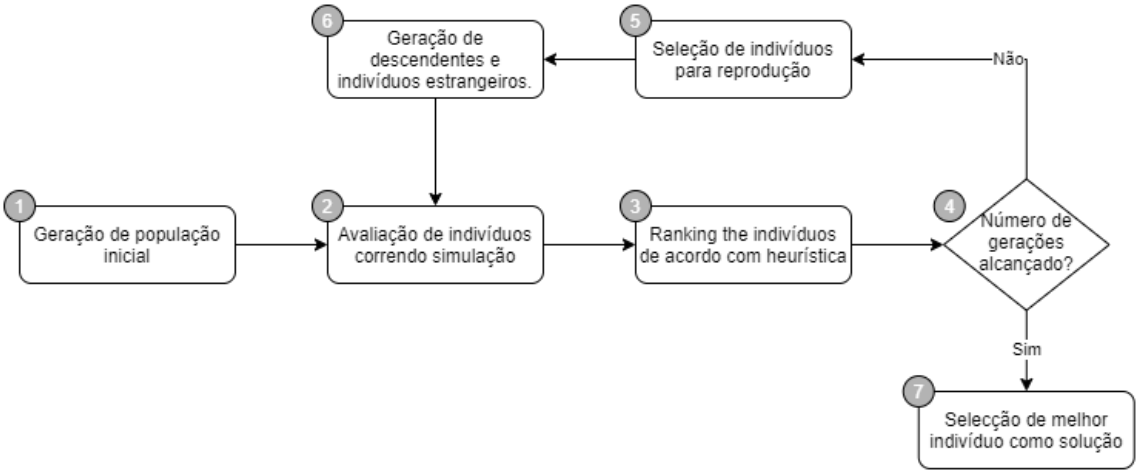


Figura 8 – Esquema do algoritmo evolucionário

No final da execução do algoritmo obtém-se as melhores soluções encontradas para o problema de otimização.

3.2.1 Heurística

De forma a avaliar o desempenho de cada indivíduo da população, é executada uma simulação com os parâmetros por ele codificados, sendo os resultados posteriormente avaliados com recurso a uma heurística.

Esta heurística composta é baseada nas diferentes métricas definidas na Tabela 8.

Tabela 8 – Métricas utilizadas na elaboração da heurística

Métricas	**Descrição**
Consumo energético	Consumo energético médio por estação. (Joules)
Packet Loss Ratio	Percentagem de perda de pacotes medida na camada de rede.
Throughput	Throughput médio por estação, medido na camada aplicacional, com base nos pacotes recebidos nos APs. (bit/s)

Aquando da avaliação do indivíduo estas métricas são combinadas numa heurística composta, a qual é utilizada como valor de fitness do individuo.

3.2.1.1 Heurística v1

A versão inicial do algoritmo evolucionário fazia uso de uma heurística composta baseada em duas métricas: *throughput* e consumo energético

Esta heurística encontra-se implementada na função abaixo.

```
1.  def heuristic_v1 (energy, throughput, packet_loss=0.0, minimum_throughput=0.0):
2.      if (throughput <= minimum_throughput):
3.          return sys.maxsize
4.      return (energy/throughput)
```

Esta função combina o consumo energético e o *throughput*, definidos na Tabela 8, numa heurística composta: consumo energético por bit (J/bit). Este valor é posteriormente utilizado como *fitness* do individuo, tendo os melhores indivíduos valores menores.

Existe ainda a possibilidade de definição de um *throughput* médio mínimo, passando-o como argumento à função. Neste caso, o indivíduo necessitará de um desempenho mínimo neste componente da heurística, caso o mesmo não seja cumprido, este incorre na penalização máxima possível, sendo-lhe atribuído o valor de fitness "sys.maxsize" (linha 2-3).

Um problema desta implementação é a possibilidade de originar casos cujo consumo de energia por bit é reduzido, mas a perda de pacotes é extremamente elevada.

3.2.1.2 Heurística v2

De forma a combater o problema mencionado na subsecção anterior, foi criada uma versão melhorada da Heurística v1.

A diferença entre a versão 2 e a versão inicial da heurística é adição de uma penalização proporcional ao *packet loss ratio* (linha 4 do código seguinte).

```
1.  def heuristic_v2 (energy, throughput, packet_loss=0.0, minimum_throughput=0.0):
2.      if (throughput <= minimum_throughput):
3.          return sys.maxsize
4.      return (energy/throughput*(1.0+packet_loss)
```

3.2.2 Sistema de seleção

Uma parte extremamente importante do algoritmo evolucionário é o sistema de seleção dos progenitores. Com o objetivo de favorecer os melhores indivíduos, mantendo alguma diversidade, adotei uma abordagem de seleção baseada em *ranking* linear. Este sistema de seleção encontra-se implementado no seguinte excerto de código:

```
1.  def rank_pop (population, n):
2.      pop = copy.deepcopy(population)
3.      pop.sort(reverse=True, key=lambda x: x[-1])
4.      probs = [(2*i)/(pop_size*(pop_size + 1)) for i in range (1, pop_size + 1)]
5.      parents = []
6.      for _ in range (n):
7.          value = random.uniform(0,1)
8.          index = 0
9.          total = probs [index]
10.         while total < value:
11.             index += 1
12.             total += probs[index]
13.         parents.append(pop[index])
14.       return parents
```

Nesta abordagem os indivíduos são organizados por ordem decrescente de *fitness* (linha 3). É-lhes seguidamente atribuída uma probabilidade de seleção baseada na posição que ocupam nesta lista. Uma vez que neste algoritmo evolucionário, quanto maior o valor de *fitness*, pior o individuo, os primeiros indivíduos da lista recebem uma probabilidade de seleção (linha 4). Posteriormente, nas linhas 6 a 13, são selecionados n indivíduos com base numa distribuição uniforme.

3.2.3 Indivíduos estrangeiros

Uma vez que a geração de descendentes é baseada num operador de mutação bastante controlada, ou seja, que envolve apenas um gene, poderá ocorrer uma convergência prematura do algoritmo evolucionário. Esta convergência poderá levar o algoritmo a ficar "preso" num mínimo local. De forma a evitar este problema, são efetuadas injeções de indivíduos "estrangeiros", em cada geração. Estes indivíduos são gerados aleatoriamente, e constituem um novo ponto de partida para possíveis soluções, evitando convergência do algoritmo em mínimos locais.

3.2.4 Operador de mutação

De forma a escolher o operador de mutação mais adequado para cada individuo foram realizados diversos testes num cenário simples de simulação. Estes testes utilizaram não só diferentes operadores de mutação, como diferentes configurações dos mesmos.

Tanto as configurações do cenário de simulação como os testes encontram-se descritos detalhadamente na secção de validação do algoritmo evolucionário (3.5.2).

3.2.4.1 Single gene mutation operator

O Single Gene Mutation Operator (SGMO) é um operador de mutação que afeta um único gene. Este operador de mutação encontra-se implementado na função *mutate_one()*, do código providenciado no Apêndice B, apresento no excerto seguinte.

```
1.  def mutate_one(original_indiv):
2.      global param
3.      # indiv = [technology, frequency, channelWidth, useUDP, useRts, guardInterval,
    enableObssPd, useExtendedBlockAck, mcs]
4.      indiv = original_indiv
5.      indiv [-1] = -1
6.
7.      i=random.randint (0,len(original_indiv) - 2)
8.
9.      while ((i==1) and (len(param[1][indiv[0]])==1)): #case tech=1 == 802.11n, and
    there's only 5GHz
10.         i=random.randint (0,len(original_indiv) - 2)
11.
12.     # frequency and channel width
13.     if (i == 1) or (i == 2):
14.         temp=indiv[i]
15.         while (temp == indiv[i]):
16.             temp = param[i][indiv[i-1]][random.randint(0, len(param[i][indiv[i-1]])
    - 1)]
17.         indiv[i] = temp
18.
19.     #guardInterval and mcs
20.     elif (i == 5) or (i == 8):
21.         temp=indiv[i]
22.         while (temp == indiv[i]):
23.             temp = param[i][indiv[0]][random.randint(0, len(param[i][indiv[0]]) -
    1)]
24.         indiv[i] = temp
25.
26.     #other
27.     else:
28.         temp=indiv[i]
29.         while (temp == indiv[i]):
30.             temp = param[i][random.randint(0, len(param[i]) - 1)]
31.         indiv[i] = temp
32.
33.     #dependent parameters
34.     if indiv[1] not in param[1][indiv[0]]:
35.         indiv[1] = param[1][indiv[0]][random.randint(0, len(param[1][indiv[0]]) -
    1)]
36.     if indiv[2] not in param[2][indiv[1]]:
37.         indiv[2] = param[2][indiv[1]][random.randint(0, len(param[2][indiv[1]]) -
    1)]
38.     if indiv[5] not in param[5][indiv[0]]:
39.         indiv[5] = param[5][indiv[0]][random.randint(0, len(param[5][indiv[0]]) -
    1)]
40.     if indiv[8] not in param[8][indiv[0]]:
41.         indiv[8] = param[8][indiv[0]][random.randint(0, len(param[8][indiv[0]]) -
    1)]
42.
43.     return indiv
```

Esta função seleciona aleatoriamente um gene (linha 7) e modifica o mesmo selecionando um valor diferente, de entre os permitidos para aquele gene (linhas 9-31). No final da função (linhas 33-41), de forma a evitar a execução de combinações de parâmetros inválidas, os parâmetros dependentes de outros são verificados e, caso necessário, alterados para valores gerados aleatoriamente.

Com base nos testes realizados, apresentados na secção 3.5.2, este operador promove a existência de uma população diversa, não impedindo, no entanto, a convergência da mesma.

3.2.4.2 Multi-gene probabilistic mutation operator

O segundo operador de mutação implementado, denominado multi-gene probabilistic mutation operator (MGMO), é um operador de mutação que pode mutar diversos genes. Ao

executar este operador sobre um cromossoma, existe uma probabilidade independente de ocorrer mutação em cada gene, podendo assim, ocorrer diversas mutações no mesmo cromossoma.

Embora não seja utilizado, na versão final do algoritmo, devido a desempenho inferior ao outro operador de mutação implementado, foi uma das principais opções para operador de mutação e encontra-se implementado na função *mutate_prob()* do algoritmo evolucionário (Apêndice B), a qual disponibilizo abaixo.

```
1.  def mutate_prob(original_indiv):
2.      global param
3.      global mutation_prob
4.      #indiv = [technology, frequency, channelWidth, useUDP, useRts, guardInterval,
    enableObssPd, useExtendedBlockAck]
5.      indiv = original_indiv
6.      indiv [-1] = -1
7.
8.      for i in range(len(indiv) - 1):
9.          if random.random() < mutation_prob:
10.             if ((i==1) and (len(param[1][indiv[0]])==1)): #case tech=1 == 802.11n,
    and there's only 5GHz
11.                 continue
12.
13.             #frequency and channel width
14.             if (i == 1) or (i == 2):
15.                 temp=indiv[i]
16.                 while (temp == indiv[i]):
17.                     temp = param[i][indiv[i-1]][random.randint(0,
    len(param[i][indiv[i-1]]) - 1)]
18.                 indiv[i] = temp
19.
20.             #guardInterval and mcs
21.             elif (i == 5) or (i == 8):
22.                 temp=indiv[i]
23.                 while (temp == indiv[i]):
24.                     temp = param[i][indiv[0]][random.randint(0,
    len(param[i][indiv[0]]) - 1)]
25.                 indiv[i] = temp
26.
27.             #other
28.             else:
29.                 temp=indiv[i]
30.                 while (temp == indiv[i]):
31.                     temp = param[i][random.randint(0, len(param[i]) - 1)]
32.                 indiv[i] = temp
33.
34.             #dependent parameters
35.             if indiv[1] not in param[1][indiv[0]]:
36.                 indiv[1] = param[1][indiv[0]][random.randint(0,
    len(param[1][indiv[0]]) - 1)]
37.             if indiv[2] not in param[2][indiv[1]]:
38.                 indiv[2] = param[2][indiv[1]][random.randint(0,
    len(param[2][indiv[1]]) - 1)]
39.             if indiv[5] not in param[5][indiv[0]]:
40.                 indiv[5] = param[5][indiv[0]][random.randint(0,
    len(param[5][indiv[0]]) - 1)]
41.             if indiv[8] not in param[8][indiv[0]]:
42.                 indiv[8] = param[8][indiv[0]][random.randint(0,
    len(param[8][indiv[0]]) - 1)]
43.
44.     return indiv
```

Esta função percorre todos os genes determinando aleatoriamente se ocorrerá mutação ou não, do mesmo (linha 9). Em caso afirmativo, são repetidos os passos do operador definido na secção 3.2.4.1, sendo o gene em questão alterado para um valor aleatório, de entre os

permitidos (linhas 13-32). Novamente, no final desta alteração, os parâmetros dependentes de outros são verificados e, caso necessário, alterados para valores válidos gerados aleatoriamente (linhas 34-42).

3.2.5 Otimizações

Ao longo deste projeto foram realizadas inúmeras otimizações e compromissos devido aos constrangimentos temporais provenientes da natureza do mesmo, e também, ao hardware disponível. Estas otimizações tiveram em conta os diversos componentes do projeto, mas, principalmente o design do algoritmo evolucionário.

Uma das características principais deste algoritmo é a forma como avalia indivíduos. Tradicionalmente a avaliação dos indivíduos é efetuada no cenário que se pretende otimizar, mas neste caso, tal não seria possível. Mesmo após diversas otimizações ao cenário de simulação, seria impossível executar a avaliação de centenas, ou milhares de indivíduos com mais de 1000 dispositivos, em tempo útil. Por este motivo tomei outra abordagem. No meu algoritmo os indivíduos são avaliados em cenários progressivamente mais complexos. Este método utiliza três versões do cenário de simulação, incrementado a complexidade ao longo da execução.

As primeiras 25 gerações são avaliadas num cenário de simulação simplificado, as próximas 15 são avaliadas num cenário intermédio, sendo as últimas 5 avaliadas no cenário real. Outra otimização efetuada prende-se com o número de indivíduos por geração. Enquanto as primeiras gerações fazem uso de populações de 25 indivíduos, as seguintes utilizam apenas 10. A maior população inicial, em conjunção com os indivíduos estrangeiros, evita a estagnação da população em mínimos locais, sendo as otimizações finais mais ligeiras. Isto leva a uma redução significativa da complexidade temporal.

Existem ainda otimizações no espaço de procura. Não são gerados, nem executados indivíduos com configurações completamente inviáveis, e a frequência de 2.4 GHz não é considerada devido ao pior desempenho em cenários de elevada densidade [50]. Tal deve-se ao facto de possuir menor número de canais disponíveis do que 5 GHz, não sendo por isso indicada para este tipo de comunicações. Esta disponibiliza ainda pior taxa de transferência de dados.

A principal vantagem da utilização da frequência de 2.4 GHz encontra-se presente em cenários interiores, devido a contornar melhor objetos sólidos do que frequências mais elevadas [49]. Uma vez que este projeto tem como foco cenários de combate a incêndios florestais, tal não é relevante.

3.3 Configurações e Cenários de simulação

A implementação do cenário base de simulação teve início com uma análise de diversos scripts modelo do simulador de redes ns-3 de forma a familiarizar-me com o simulador e as suas implementações dos diversos conceitos. Foi posteriormente efetuada a criação de um cenário preliminar com base nalguns aspetos dos cenários de incêndio, implementando novas funcionalidades do standard 802.11ax, nomeadamente *spatial reuse* com *BSS colouring*, bem como a recolha e processamento de diversas métricas relacionadas com o desempenho das tecnologias.

Além do trabalho referido acima, realizei também uma investigação acerca de implementações de aplicações MQTT e CoAP em ns-3, incluindo características dos mesmos no cenário de simulação.

De forma a permitir a colaboração com o orientador e a efetuar controlo de versões, foi criado um repositório git com o código do script de simulação e do algoritmo evolucionário, os quais disponibilizo em apêndice.

3.3.1 Cenários de simulação para avaliação de indivíduos

Devido à complexidade computacional das simulações com elevado número de dispositivos, em ns-3, foram criadas duas versões simplificadas do cenário final. Estas configurações são utilizadas para avaliar o desempenho das primeiras gerações de indivíduos em cenários progressivamente mais complexos e realistas, desta forma obtém-se uma menor complexidade temporal sem afetar noticiavelmente os resultados.

As configurações dos três cenários utilizados na avaliação dos indivíduos são as seguintes:

3.3.1.1 Cenário básico

- Número de AP: 4
- Número de STA por AP: 4
- Duração: 10s
- Taxa de *upload* de cada STA: 100 Kbps

Este cenário básico é utilizado na fase inicial de otimização com recurso ao algoritmo evolucionário. Nesta fase são selecionadas boas configurações para um cenário básico, as quais serão utilizadas como base para a fase seguinte.

3.3.1.2 Cenário intermédio

- Número de AP: 9
- Número de STA por AP: 16
- Duração: 3s
- Taxa de *upload* de cada STA: 100 Kbps

O segundo cenário constitui um ponto intermédio de otimização. Contrariamente ao cenário básico, este cenário já apresenta alguma complexidade com concentração mais elevada de dispositivos, apresentando, por isso resultados mais relevantes e interessantes.

3.3.1.3 Cenário complexo

- Número de AP: 4
- Número de STA por AP: 32
- Duração: 3s
- Taxa de *upload* de cada STA: 100 Kbps

O último cenário apresenta uma simulação representativa de um cenário real de combate a incêndios com elevada densidade de dispositivos. Por este motivo, este cenário constitui o último passo no processo de otimização.

De forma a diminuir a complexidade temporal e espacial deste cenário de simulação, foram utilizados 4 APs com 32 STAs por AP. Desta forma é possível manter mais de 30 STA por

AP, constituindo, por esse motivo, um cenário de elevada densidade [45]. Outro fator contributivo para a densidade de todos os cenários é a utilização do mesmo canal por parte de todas as STAs ao conectar-se aos APs.

3.3.2 Funcionalidades

Um dos focos deste projeto é o estudo de diferentes configurações de dispositivos, em particular de novas funcionalidades do standard 802.11ax e das suas aplicações, por essa razão a sua implementação é algo de particular importância.

Os cenários de simulação são configuráveis de forma a poder simular a utilização de diversas tecnologias e funcionalidades por elas suportadas, como demonstrado na Tabela 7 da secção 3.2.

3.3.2.1 Spatial Reuse

De forma a utilizar o novo mecanismo de *spatial reuse* do IEEE 802.11ax, configurei o Wi-Fi no modo de *spatial reuse* OBSS/PD (Overlapping BSS/Preamble-Detection), com um determinado *threshold* (-82dB por predefinição).

```
1.    if (enableObssPd)
2.    {
3.      wifi.SetObssPdAlgorithm("ns3::ConstantObssPdAlgorithm",
4.                              "ObssPdLevel", DoubleValue(obssPdThreshold));
5.  }
6.
```

Posteriormente, atribui uma cor a cada AP de forma a diminuir a interferência entre diferentes BSS.

```
1.  if (enableObssPd)
2.  {
3.     ap2Device[i]->GetHeConfiguration()-
    >SetAttribute("BssColor", UintegerValue(i + 1));
4.  }
5.
```

3.3.2.2 Modulation and Coding Scheme

O *Modulation and Coding Scheme* (MCS) é uma parte essencial da configuração de redes Wi-Fi. Este possui uma grande influência não apenas no alcance útil da rede, como no *throughput*.

É possível configurar os cenários de simulação com todos os valores de MCS suportados pelas diferentes tecnologias estudadas.

Quando configurado no modo IEEE 802.11n, é suportado High-Throughput MCS 0-23.

```
1.  std::ostringstream oss;
2.  oss << "HtMcs" << mcs;
3.  wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager",
4.                               "DataMode", StringValue(oss.str()),
5.                               "ControlMode", StringValue(oss.str()));
```

Já o standard IEEE 802.11ax suporta High Efficiency MCS 0-11.

```
6.  std::ostringstream oss;
7.  oss << "HeMcs" << mcs;
8.  wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager",
9.                               "DataMode", StringValue(oss.str()),
10.                              "ControlMode", StringValue(oss.str()));
```

3.3.2.3 Protocolo de comunicação

Os protocolos de comunicação CoAP e MQTT encontram-se assentes em ligações UDP e TCP, respetivamente. Por esse motivo, de forma a simular a recolha de dados dos dispositivos IoT, o cenário de simulação implementa uma arquitetura cliente-servidor baseada em UDP e TCP.

Utilizando sockets UDP ou TCP em conjunção com a classe PacketSink e a aplicação OnOff nos dispositivos, foi possível configurar o cenário com envio constante de dados a um *bit rate* constante. O código seguinte ainda possibilita não só a alteração do *bit rate*, como também a modificação do tamanho da *payload* dos pacotes, através das variáveis datarate (linhas 13 e 30), udpPayloadSize (linha 13) e tcpPayloadSize (linha 30).

```
1.  ApplicationContainer clientApps;
2.  ApplicationContainer serverApps;
3.
4.  if (useUdp)
5.  {
6.    PacketSinkHelper packetSinkHelper("ns3::UdpSocketFactory",
    InetSocketAddress(Ipv4Address::GetAny(), 9));
7.    serverApps = packetSinkHelper.Install(wifiApNodes);
8.
9.    //client
10.   for (int32_t i = 0; i < numAp; i++)
11.   {
12.     OnOffHelper onoff("ns3::UdpSocketFactory",
    Address(InetSocketAddress(apInterfaces.GetAddress(i), 9)));
13.     onoff.SetConstantRate(DataRate(dataRate), udpPayloadSize);
14.
15.     for (int32_t j = 0; j < numSta; j++)
16.     {
17.       clientApps = onoff.Install(wifiStaNodes.Get(i * numSta + j));
18.     }
19.   }
20. }
21. else
22. {
23.   PacketSinkHelper packetSinkHelper("ns3::TcpSocketFactory",
    InetSocketAddress(Ipv4Address::GetAny(), 5000));
24.   serverApps = packetSinkHelper.Install(wifiApNodes);
25.
26.   //client
27.   for (int32_t i = 0; i < numAp; i++)
28.   {
29.     OnOffHelper onoff("ns3::TcpSocketFactory",
    Address(InetSocketAddress(apInterfaces.GetAddress(i), 5000)));
30.     onoff.SetConstantRate(DataRate(dataRate), tcpPayloadSize);
31.
32.     for (int32_t j = 0; j < numSta; j++)
33.     {
34.       clientApps = onoff.Install(wifiStaNodes.Get(i * numSta + j));
35.     }
36.   }
37.  }
```

3.3.2.4 RTS/CTS

Request To Send/Clear To Send (RTS/CTS) é um mecanismo de *handshaking* com o objetivo de resolver o problema do nó oculto, evitando colisões. Este problema consiste na existência de dois nós a comunicar com um AP mas fora de alcance um do outro. O código seguinte altera o *threshold* RTS/CTS para 0 octetos, forçando o envio de frames RTS/CTS para todos os pacotes.

```
1.  if (useRts)
2.  {
3.    Config::SetDefault("ns3::WifiRemoteStationManager::RtsCtsThreshold",
    StringValue("0"));
4.  }
```

A utilização deste mecanismo reduz a colisão entre frames devido a esse problema, o que poderá melhorar o *throughput* em situações de elevada densidade.

3.3.2.5 Intervalo de Guarda

O intervalo de guarda (*Guard Interval* – GI) consiste num intervalo entre símbolos adjacentes. A utilização de intervalos mais curtos aumenta o *throughput*, mas aumenta a interferência entre símbolos.

O standard IEEE 802.11ax suporta três valores para o intervalo de guarda: 800 ns, 1600 ns e 3200 ns [41]. Estes valores são configurados no seguinte excerto de código através da variável guardInterval.

```
1.  Config::Set("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/HeConfiguration/GuardInte
    rval", TimeValue(NanoSeconds(guardInterval)))
```

Já no caso do 802.11n, o intervalo de guarda é 800 ns por predefinição, sendo também possível a utilização de um intervalo de guarda curto (400 ns) [41]. Esta opção é configurada através do código abaixo.

```
1.  Config::Set("/NodeList//DeviceList//$ns3::WifiNetDevice/HtConfiguration/ShortGuardI
    ntervalSupported", BooleanValue(guardInterval));
```

3.3.2.6 Block Acknowledgement

Block Ack é uma funcionalidade dos standards Wi-Fi com origem no IEEE 802.11n. Com o IEEE o tamanho máximo do buffer de receção de A-MPDUs foi incrementado de 64 MPDUs para 256.

A seguinte linha de código possibilita a utilização de Block Ack extenso, alterando o tamanho do buffer para o máximo permitido, quando utilizada a flag useExtendedBlockAck=1.

```
1.  Config::Set("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/HeConfiguration/MpduBuffe
    rSize", UintegerValue(useExtendedBlockAck ? 256 : 64));
```

3.3.3 Recolha de métricas

A recolha de métricas é uma parte fundamental dos cenários de simulação, pois permitem avaliar o desempenho das diversas configurações, bem como efetuar a validação e *debugging* dos mesmos. Por esse motivo, implementei mecanismos de recolha e processamento de

diversos dados e métricas. De entre estas métricas, houve especial foco na energia consumida e no *throughput* ao nível da aplicação.

3.3.3.1 Consumo médio de energia

De forma a efetuar a medição do consumo de energia recorri aos seguintes módulos do ns-3:

- wifi-radio-energy-model-helper.h
- energy-source-container.h
- li-ion-energy-source-helper.h
- li-ion-energy-source.h

Começo por instalar e configurar a fonte de energia com recurso ao LiIonEnergySourceHelper e ao WifiRadioEnergyModelHelper, como exemplificado no seguinte excerto do código em Anexo:

```
1.  /** Energy Model **/
2.  /*********************************************************************/
3.  /* energy source */
4.  LiIonEnergySourceHelper liIonSourceHelper;
5.  // configure energy source
6.  liIonSourceHelper.Set("LiIonEnergySourceInitialEnergyJ",
    DoubleValue(batteryLevel));
7.  // install source
8.  EnergySourceContainer sources = liIonSourceHelper.Install(wifiStaNodes);
9.  /* device energy model */
10. WifiRadioEnergyModelHelper radioEnergyHelper;
11. // configure radio energy model
12. radioEnergyHelper.Set("TxCurrentA", DoubleValue(TxCurrentA));
13. radioEnergyHelper.Set("RxCurrentA", DoubleValue(RxCurrentA));
14. radioEnergyHelper.Set("IdleCurrentA", DoubleValue(IdleCurrentA));
15. radioEnergyHelper.Set("SleepCurrentA", DoubleValue(SleepCurrentA));
16. radioEnergyHelper.Set("CcaBusyCurrentA", DoubleValue(CcaBusyCurrentA));
17. radioEnergyHelper.Set("SwitchingCurrentA", DoubleValue(SwitchingCurrentA));
18. // install device model
19. DeviceEnergyModelContainer deviceModels = DeviceEnergyModelContainer();
20.
21. for (int32_t i = 0; i < numAp; i++)
22.   {
23.   for (int32_t j = 0; j < numSta; j++)
24.   {
25.     deviceModels.Add(radioEnergyHelper.Install(staDevices[i].Get(j), sources.Get(j
    + i * numSta)));
26.   }
27. }
```

Finalmente, para aceder ao valor total da energia consumida por cada STA durante a simulação, percorro os diferentes modelos de energia implementados (um por dispositivo), e chamo o método *GetTotalEnergyConsumption()*, o qual me devolve esse valor em Joules.

```
1.  //get total energy consumed
2.  for (DeviceEnergyModelContainer::Iterator iter = deviceModels.Begin (); iter !=
    deviceModels.End (); iter ++)
3.  {
4.    double energyConsumed = (*iter)->GetTotalEnergyConsumption ();
5.    avg_energy += static_cast<double>(energyConsumed);
6.  }
```

3.3.3.2 Throughput médio

Outra métrica bastante importante para avaliar o desempenho dos dispositivos no cenário de simulação é a média do *throughput* por dispositivo. Uma vez que a avaliação feita se foca particularmente na avaliação de standards e protocolos como meios de transmissão de dados, considerei o *throughput* ao nível aplicacional. Esta métrica ignora *overhead* devido aos protocolos e foca-se apenas na quantidade de dados da aplicação recebidos.

A versão inicial desta métrica, fazia uso do seguinte método baseado em *callbacks* para calcular o *throughput*.

```
1.  std::vector<uint64_t> bytesReceivedPS(MAX_NODES);
2.
3.  void PacketSinkRx (std::string context, Ptr< const Packet > packet, const Address
    &address)
4.  {
5.    std::string sub = context.substr(10);
6.    uint32_t pos = sub.find("/Application");
7.    uint32_t nodeId = atoi(sub.substr(0, pos).c_str());
8.    bytesReceivedPS[nodeId] += packet->GetSize();
9.  }

10. Config::Connect("/NodeList/*/ApplicationList/*/$ns3::PacketSink/Rx",
    MakeCallback(&PacketSinkRx));
```

Nesta implementação a cada receção de pacotes, era executada a função PacketSinkRx, a qual adicionava o tamanho do pacote ao vetor BytesReceivedPS. Um dos problemas desta implementação era a complexidade temporal associada à chamada de uma função por pacote recebido. Por esse motivo, procurei outra solução mais simples, começando com uma pesquisa aos diferentes métodos disponíveis na classe PacketSink. A versão final do cenário de simulação faz uso de um desses métodos: *GetTotalRx()*. Este método devolve a quantidade de bytes recebidos pela aplicação PacketSink, esta é convertida para bits e dividida pela duração do período de transmissão. Este passo é repetido para todos os servidores, sendo, no final, devolvido o *throughput* médio.

3.3.3.3 Packet Loss Ratio

De forma a melhorar os resultados, foi utilizada uma segunda métrica como componente da heurística composta: *packet loss ratio*. Esta métrica analisa a perda de pacotes ao nível da *network layer*. De forma a recolher esta métrica é utilizado o módulo FlowMonitor.

```
1.  for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin ();
2.       i != stats.end (); ++i)
3.    {
4.      std::stringstream ss;
5.      ss << classifier->FindFlow (i->first).sourceAddress;
6.
7.      if (!std::regex_match (ss.str (), server_regex))
8.        {
9.          int lost_packets = (i->second.txPackets - i->second.rxPackets);
10.         avg_packet_loss[r] +=
11.             static_cast<double> (lost_packets) / static_cast<double> (i-
    >second.txPackets);
12.       }
13.   }
14. avg_packet_loss[r] = avg_packet_loss[r] / static_cast<double> (numSta * numAp);
```

De forma a recolher esta métrica, são analisados fluxos de dados no sentido STA->AP. De forma a obter os fluxos pretendidos, estes são filtrados com recurso a regex, ignorando os endereços correspondentes a APs ("^172.*.0.1$").

3.3.3.4 Signal to noise ratio e packet delivery ratio (callbacks)

Outras métricas consideradas durante o desenvolvimento do cenário foram o *signal to noise ratio* (SNR) e o *packet delivery ratio* (PDR).

Tal como a versão inicial do *throughput*, ambas as métricas foram implementadas com recurso aos *callbacks* apresentados no seguinte excerto:

```
1. // Trace function for received packets and SNR
2. void MonitorSniffRx(std::string context, Ptr<const Packe
   t, uint16_t channelFreqMhz, WifiTxVector txVector, MpduInfo aMpd
   u, SignalNoiseDbm signalNoise)
3. {
4.   uint32_t nodeId = ContextToNodeId(context);
5.   packetsReceived[nodeId]++;
6.   signalDbmAvg[nodeId] += ((signalNoise.signal - signalDbmAvg[no
   deId]) / packetsReceived[nodeId]);
7.   noiseDbmAvg[nodeId] += ((signalNoise.noise - noiseDbmAvg[nodeI
   d]) / packetsReceived[nodeId]);
8. }
9.
10. // Trace function for sent packets
11.  void MonitorSniffTx(std::string context, const Ptr<const Packet
   > packet, uint16_t channelFreqMhz, WifiTxVector txVector, MpduIn
   fo aMpdu)
12. {
13.   uint32_t nodeId = ContextToNodeId(context);
14.   packetsSent[nodeId]++;
15. }
```

Para calcular o SNR e implementar o *callback* MonitorSniffRx baseei-me no script wifi-spectrum-per-example.cc dos exemplos do ns-3. Esse *callback* é executado em cada receção de pacotes, sendo efetuado o cálculo o a média do sinal e do ruído por nó.

De forma a calcular a taxa de pacotes entregues, ou *packet delivery ratio* (PDR), implementei também o *callback* MonitorSniffTx que é executado cada vez que um pacote é enviado, e dessa forma me permite calcular o número de pacotes enviados por nó, os quais são utilizados em conjunção com os pacotes recebidos no AP de forma a calcular PDR por AP.

3.3.4 Parametrização dos cenários de simulação

O principal objetivo desde projeto consiste no estudo não só de diversas configurações de dispositivos, como do seu desempenho em diferentes cenários. Com o objetivo de facilitar a configuração dos cenários e experimentação com os mesmos, os parâmetros descritos na Tabela 9 são configuráveis através de argumentos passados ao programa aquando da execução da simulação.

Tabela 9 – Parâmetros configuráveis do cenário de simulação

Pârametro	Descrição	Predefinição
batteryLevel	Energia inicial da bateria dos dispositivos (Joules).	2000000 J
ccaEdTrAp	Limite CCA-ED do AP (dBm).	- 62.0 dBm
ccaEdTrSta	Limite CCA-ED da STA (dBm).	- 62.0 dBm
channelWidth *	Define a largura do canal Wi-Fi (MHz).	20 MHz
dataRate	*Data Rate* de upload de payload (bit/s).	100000 bit/s
distance	Distância entre APs (m).	25 m
duration	Duração da simulação (segundos).	10.0 s
enabledObssPd *	Ativar/desativar OBSS-PD.	true
frequency *	Selecionar frequência (2.4 GHz, 5 GHz ou 6 GHz).	5 GHz
guardInterval *	Escolher *guard interval* (ns).	3200 ns
mcs *	Valor constante MCS para transmitir HE PPDUs.	9
numAp	Número total de APs.	2
numApAntennas	Número de antenas por AP.	4
numApRxSpatialStreams	Define o número de *streams* espaciais de receção por AP.	4
numApTxSpatialStreams	Define o número de *streams* espaciais de transmissão por AP.	4
numSta	Número de STA por AP.	4
numStaAntennas	Número de antenas por STA.	4
numStaRxSpatialStreams	Define o número de *streams* espaciais de receção por STA.	4
numStaTxSpatialStreams	Define o número de *streams* espaciais de transmissão por STA.	4
obssPdThreshold	*Threshold* ObssPd (dBm).	- 82.0 dBm
powAp	Potência do AP (dBm).	21.0 dBm
powSta	Potência da STA (dBm).	10.0 dBm
runs	Definição do número de execuções do cenário.	3
rxSensAp	Sensibilidade do AP (dBm).	- 92.0 dBm
rxSensSta	Sensibilidade da Sta (dBm).	- 92.0 dBm
seed	Definição do número da *RNG seed*.	1
tcpPayloadSize	Define o tamanho da *payload* dos pacotes TCP (bytes).	60 bytes
technology *	Escolher entre tecnologias (802.11ax ou 802.11n).	802.11ax
tracing	Ativar/desativar pcap *tracing*.	false
udpPayloadSize	Define o tamanho da *payload* dos pacotes UDP (bytes).	40 bytes
useExtendedBlockAck *	Ativar/desativar Extended Block Ack.	false
useRts *	Ativar/desativar RTS/CTS.	false
useUdp *	Escolher entre utilizar UDP ou TCP.	false
verbose	Ativar *echo logging*.	false
walk	Ativar mobilidade de STAs.	false

* Parâmetros codificados em indivíduos de algoritmo evolucionário

O algoritmo evolucionário faz uso destes parâmetros exceto quando explicitamente alterados pela configuração dos indivíduos (Tabela 7 da secção 3.2).

3.4 Experimentação

3.4.1 Ferramentas utilizadas

- C++11
- ns-3.33
- 5G-LENA
- Python3
- NumPy
- GNU Screen
- TShark
- Ubuntu 20.04

Apesar do ns-3 suportar Python, o cenário de simulação encontra-se implementado em C++, pois a maioria dos exemplos, bem como a documentação oficial utiliza C++11, sendo a documentação relativa a Python extremamente escassa. O módulo 5G-LENA [28] é uma implementação de 5G para ns-3. Embora não esteja presente na versão final, chegou a ser instalado e configurado como preparação para simulação com tecnologias 5G.

Já o algoritmo evolucionário encontra-se implementado em Python3 devido a ser uma linguagem de programação bastante simples com uma ampla seleção de módulos disponíveis.

De forma a facilitar a execução do algoritmo evolucionário, bem como a sua validação, foram ainda utilizadas as ferramentas GNU Screen e TShark. A primeira permite a manutenção de diversas sessões de terminal, possibilitando, não só a execução simultânea de diferentes programas, como a manutenção de registos das mesmas, em diversos ficheiros. A segunda ferramenta, TShark, possibilita a análise de pacotes, os quais são guardados em ficheiros pcap, aquando da execução do cenário de simulação, para efeitos de *debugging*.

3.4.1.1 Instalação e configuração

ns-3

Existem diversas formas de instalar o simulador de redes ns-3. De forma a poder instalar facilmente uma versão específica, optei pela instalação do ns-3-allinone com recurso à ferramenta git.

Comecei por instalar as dependências necessárias à instalação e funcionamento do ns-3:

```
1. sudo apt install g++ python3 python3-dev pkg-config sqlite sqlite3 libsqlite3-dev
   python3-setuptools git libxml2 libxml2-dev
```

Clonei o repositório ns-3-allinone com o seguinte comando:

```
1. git clone https://gitlab.com/nsnam/ns-3-allinone.git
```

Seguidamente, entrei no repositório e transferi o ns-3.33 executando o script download.py

```
1. cd ns-3-allinone/
2. ./download.py -ns-3.33
```

O próximo passo consiste na configuração e compilação do ns-3, bem como dos exemplos e testes, com *build profile* otimizado.

```
1. cd ns-3.33/
2. ./waf configure --build-profile=optimized --enable-examples --enable-tests
```

Algoritmo evolucionário

O algoritmo evolucionário tem como requisitos a linguagem de programação Python3 e a biblioteca NumPy. A instalação dos mesmos é bastante simples, necessitando apenas da execução dos seguintes comandos:

```
1. sudo apt install python3 python3-pip
2. pip3 install numpy
```

Outras ferramentas

A instalação das ferramentas GNU Screen e TShark é efectuada com recurso ao *package manager* apt:

```
1. sudo apt install screen tshark
```

3.4.2 Ambiente de experimentação

De forma a testar e validar os diversos componentes deste projeto, foram utilizadas duas máquinas virtuais (VM). Estas máquinas virtuais apresentam as seguintes configurações:

VM1:

- Número de vCPU: 16
- RAM: 15978 MB

VM2:

- Número de vCPU: 4
- RAM: 15449 MB

3.4.3 Modelo de gestão de energia no ns-3

De forma a calcular o consumo energético do sistema é utilizada uma bateria *lithium-ion*, ou *Li-ion*.

Uma das principais métricas recolhidas em qualquer cenário de experimentação é o consumo de energia. Esta métrica varia consoante diversos fatores, sendo um dos principais o standard utilizado. De forma a simular estas diferenças foi necessário configurar diferentes modelos para os diferentes standards.

3.4.3.1 IEEE 802.11n

O simulador de redes ns-3 possui uma classe (WifiRadioEnergyModel) a qual representa um modelo de energia Wi-Fi baseado no artigo "Demystifying 802.11n Power Consumption" de Daniel Halperin, Ben Greenstein, Anmol Sheth e David Wetherall [36][37].

- O modelo acima referido define os seguintes atributos:TxCurrentA – corrente, em Amperes, no estado TX (transmissão)
- RxCurrentA – corrente, em Amperes, no estado RX (receção)
- IdleCurrentA – corrente, em Amperes, no estado Idle
- SleepCurrentA – corrente, em Amperes, no estado Sleep
- CcaBusyCurrentA – corrente, em Amperes, no estado CCA Busy
- SwitchingCurrentA – corrente, em Amperes, predefinida para troca de canais

2.4 GHz

Por predefinição, o modelo adota os seguintes valores, representantes do consumo energético do IEEE 802.11n 2.4 GHz:

- TxCurrentA = 0.38
- RxCurrentA = 0.313
- IdleCurrentA = 0.273
- SleepCurrentA = 0.033
- CcaBusyCurrentA = 0.273
- SwitchingCurrentA = 0.273

5 GHz

Quanto à frequência de 5 GHz, foi efetuada uma análise à tese de doutoramento do Dr. Vítor Bernardo: Energy Efficient Multimedia Communications in IEEE 802.11 Networks [38].

Analisando os resultados referentes ao consumo energético do NetworkCard-B, operando nas frequências de 2.4 GHz e 5.0 GHz, foi possível verificar um aumento de ~37.8% e ~33.3% no consumo energético durante os estados TX e RX, respetivamente. Já os estados Sleep e Idle sofreram um aumento de ~6.7% e ~37.1%. Com base nestas proporções adaptei os valores do modelo utilizado na frequência de 2.4 GHz de forma a obter o seguinte modelo:

- TxCurrentA = 0.52364
- RxCurrentA = 0.417229
- IdleCurrentA = 0.374283
- SleepCurrentA = 0.035211
- CcaBusyCurrentA = 0.374283
- SwitchingCurrentA = 0.374283

3.4.3.2 IEEE 802.11ax

De forma a simular o consumo energético do standard IEEE 802.11ax, baseei-me no artigo de H. Yang, D. Deng and K. Chen, "On Energy Saving in IEEE 802.11ax" [40]. Neste artigo é definida a potência em Watts, de diversos modos: *Trasnmission* (1.0 W), *Receive* (0.600 W), *Iddle* (0.300 W) e Doze (0.150 W). Utilizando este valores, e assumindo a uma voltagem de 3.0 V, tal como assumido no WifiRadioEnergyModel do ns-3 [36], obtive os seguintes valores:

- TxCurrentA = 0.333
- RxCurrentA = 0.200
- IdleCurrentA = 0.100

- SleepCurrentA = 0.05
- CcaBusyCurrentA = 0.100
- SwitchingCurrentA = 0.100

3.4.4 Antenas e streams espaciais

Ao configurar o cenário de simulação assumiu-se que cada possui AP quatro antenas e um máximo de 4 streams espaciais de transmissão ou receção. Esta definição deve-se não só ao facto do standard 802.11n possuir um limite máximo de 4 antenas como a limitações do software de simulação [39].

3.4.5 Modelo de mobilidade

Como esquematizado na Figura 9, cada sub-rede – representada por um quadrado colorido – encontra-se distribuída por uma área quadrangular, existindo sobreposição entre as diferentes redes. Cada quadrado colorido tem uma aresta de 50 metros. No centro de cada sub-rede encontra-se um AP – losango azul – estando as suas estações – círculos verdes – distribuídas em grelha pelo quadrado. Este padrão repete-se independentemente do tamanho das redes e sub-redes.

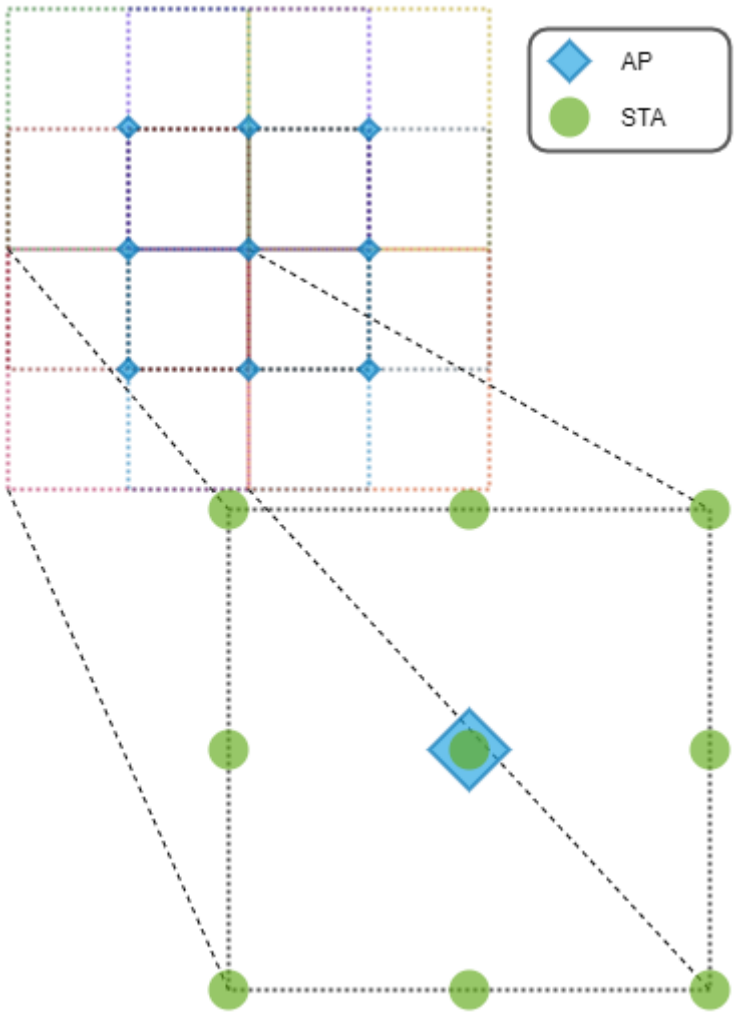


Figura 9 – Distribuição de nós

Inicialmente, foi utilizado o modelo de mobilidade "RandomWalk2dMobilityModel". Este modelo consiste no movimento de um objeto a uma velocidade aleatória em direção a um ponto aleatório, sendo o processo de seleção de um novo ponto repetido após um determinado intervalo de tempo. Tanto velocidade como coordenadas do ponto (x e y) são selecionadas de uma distribuição uniforme com máximos e mínimos definidos pelo utilizador.
Este modelo é principalmente utilizado para simular a movimentação de humanos a caminhar numa área delimitada. Este modelo foi utilizado, pois o combate a incêndios florestais/rurais geralmente encontra-se associado a uma determinada zona, com velocidades de movimentação dentro de intervalos, sendo este o modelo do ns-3 que melhor representa tal cenário.

No entanto, este foi posteriormente alterado para o "ConstantPositionModel", o qual mantem todos os dispositivos na posição inicial. Esta escolha de design prende-se na necessidade de obter resultados consistentes, com relativamente baixa complexidade temporal. Caso optasse pelo primeiro modelo de mobilidade, seria necessário maior número de simulações de forma a combater a aleatoriedade associada ao "RandomWalk2dMobilityModel", algo que não acontece. Outro fator contribuinte para esta opção, foi que o combate a incêndios florestais geralmente não envolve grandes deslocações em torno do veículo, obtendo-se assim resultados suficientemente próximos através das posições iniciais.

3.5 Validação

Ao longo do desenvolvimento do cenário de simulação e do algoritmo evolucionário foram efetuados diversos testes aos diferentes componentes.

3.5.1 Throughput

Os protocolos de comunicação MQTT e CoAP são um componente essencial da comunicação entre dispositivos IoT no cenário de simulação. Estes encontram-se simulados com recurso ao envio de pacotes TCP e UDP, respetivamente. Por esse motivo, considero a sua transmissão, bem como a medição do *throughput*, essenciais para o sucesso do projeto. Com isso em conta, houve particular atenção em garantir que o mesmo se encontra completamente funcional. De forma a validar o seu funcionamento, foi utilizado tanto o módulo FlowMonitor do ns-3 e implementações de *throughput* baseadas em exemplos, como a ferramenta TShark.

Os testes ao throughput foram efetuados recorrendo ao seguinte cenário:

- Cenário base (1 AP, 1 STA)
- Cenário simples (2 STA, 4 AP/STA)

Comecei os testes recorrendo ao cenário simples de simulação, mas rapidamente alterei o cenário de teste para o cenário base, de forma a simplificar a simulação e diminuir a influência de outras variáveis, como por exemplo a interferência.

O primeiro cenário de simulação testado, recorria ao FlowMonitor para verificar os diversos fluxos de dados entre STAs e APs, bem como para recolher estatísticas relevantes, nomeadamente PLR, *data rate* e *throughput* na camada de rede. O FlowMonitor é um módulo do ns-3 utilizado para avaliar performance de redes ao nível da camada de rede. Este módulo

providência diversas estatísticas relevantes, tal como o número de pacotes enviados e recebidos, por fluxo de dados.

De forma a validar o *throughput*, utilizei o método de medição de *throughput* incluído abaixo, o qual se encontra implementado no exemplo wifi-spatial-reuse.cc.

```
1.  void MonitorSniffRx(std::string context, Ptr<const Packet> packet, uint16_t
    channelFreqMhz, WifiTxVector txVector, MpduInfo aMpdu, SignalNoiseDbm signalNoise,
    uint16_t staId)
2.  {
3.    std::string sub = context.substr(10);
4.    uint32_t pos = sub.find("/Device");
5.    uint32_t nodeId = atoi(sub.substr(0, pos).c_str());
6.    bytesReceived[nodeId] += packet->GetSize();
7.  }

1.  Config::Connect("/NodeList/*/DeviceList/*/Phy/MonitorSnifferRx",
    MakeCallback(&MonitorSniffRx));
```

Este método já havia sido utilizado, numa implementação inicial do cenário de simulação e recorre ao *callback* MonitorSnifferRx do módulo WifiPhy para aceder ao tamanho do pacote em cada receção, por parte do AP. Através deste método obtive um valor de *throughput* (0.342717 Mbps) substancialmente mais elevado do que o total medido com recurso ao FlowMonitor (Figura 10): 0.294995 Mbps.

```
Throughput for BSS 1: 0.342717 Mbit/s
Flow;source;src_port;destiny;dst_port;proto;direction;tx_packets;tx_bytes;tx_of
fered_raw;tx_offered_mbps;rx_bytes;rx_throughput_raw;rx_throughput_mbps;mean_de
lay(ms);mean_jitter(ms);rx_packets;lost_packets;packet_loss_ratio
1;172.1.0.2:49153;172.1.0.1:5000;TCP;upload;3125;287464;229971.200000;0.229971;
287464;229971.200000;0.229971;0.006168;3125;0;0.000000
2.172.1.0.1:5000;172.1.0.2:49153;TCP;download;1563;81280;65024.000000;0.065024;
81280;65024.000000;0.065024;0.144422;0.005284;1563;0;0.000000
2.06383 0.294995
```

Figura 10 – Resultados testes FlowMonitor

Tal discrepância levou-me a verificar toda a implementação do cenário de simulação, testando diversas implementações do envio de pacotes, utilizando diferentes classes do ns-3 (OnOff, UDPServer…) mas os resultados mantiveram-se iguais.

Enquanto realizava os testes acima referidos, comecei o processo de testar o algoritmo evolucionário recorrendo a uma configuração com cinco gerações e uma população de 25, em conjunção com apenas um cenário de simulação simples, com 2 AP, 4 STA/AP e duração de 10 segundos. Após alguns testes deparei-me com um possível problema: a grande maioria dos melhores indivíduos possuíam configurações que recorriam ao protocolo TCP.

Por este motivo, decidi analisar detalhadamente os pacotes com recurso à ferramenta TShark (Figura 11). Através desta análise retirei duas conclusões: Primeiro observei que a discrepância nas medições do *throughput* se devia às medições serem efetuadas em diferentes camadas, sendo que MonitorSnifferRx devolve o pacote na camada física, enquanto o FlowMonitor efetua cálculos com base na L3. Por último, conclui, também, que o favorecimento do TCP se deve ao tamanho dos cabeçalhos associados a cada pacote, os quais são significativamente maiores nos pacotes TCP.

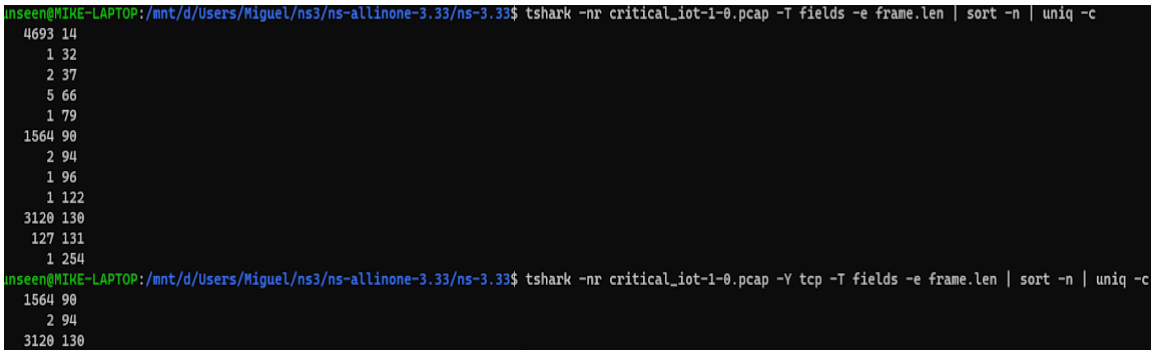


Figura 11 – Testes TShark

Esta última observação levou-me a efetuar alterações ao *throughput* utilizado. Uma vez que pretendo comparar a performance de MQTT (TCP) e CoAP (UDP) como protocolos de comunicação para transmitir dados de sensores, passei a considerar o *throughput* medido na camada aplicacional. Desta forma evito o favorecimento de TCP devido ao tamanho superior dos cabeçalhos.

3.5.2 Algoritmo evolucionário

O algoritmo evolucionário é compatível com diferentes operadores de mutação e configurações. De modo a testá-lo e a escolher a configuração mais adequada, foram realizados diversos testes com populações de 25 indivíduos, ao longo de cinco gerações, excluindo a população inicial, ou "geração 0".

3.5.2.1 Estatísticas

De forma a avaliar a performance do algoritmo são recolhidas diversas estatísticas, as quais se encontram descritas na Tabela 10.

Tabela 10 – Estatísticas do Algoritmo Evolucionário

Máximo do melhor	O melhor individuo da sua geração com o consumo energético médio (J/bit) máximo, ou seja, o "pior melhor individuo".
Mínimo do melhor	O melhor individuo da sua geração com o consumo energético médio (J/bit) mínimo; o melhor individuo de todas as populações.
Média do melhor	A média do consumo energético médio (J/bit) do melhor individuo de cada geração.
Média geracional	A média do consumo energético médio de todas as gerações.
Diversidade média	Média da diversidade das diferentes populações.

De forma a calcular estas estatísticas faço uso de duas funções do Apêndice B: *update_stats()* e *calculate_stats()*.

A função apresentada abaixo, *update_stats()*, é responsável pela recolha de estatísticas relevantes à geração atual.

```
1.  def update_stats(population, metric=hamming_distance):
2.
3.      global best_per_gen
4.      global avg_per_gen
5.      global diversity
6.      p_size = len(population)
7.
8.      population.sort(key=lambda x: x[-1])
9.      best_per_gen.append(population[0][-1])
10.     avg_per_gen.append(sum([f[-1] for f in population]) / p_size)
11.
12.     count=0
13.     for i in range (p_size):
14.         for j in range (i+1, p_size):
15.             distance = metric(population[i], population[j])
16.             if distance !=0:
17.                 count+=1
18.
19.     div = (2.0*count)/(p_size*(p_size-1))
20.     diversity.append(div)
```

Estas estatísticas consistem no melhor elemento da geração (linhas 8-9), consumo energético médio da geração (linha 10), e na diversidade (linhas 12-20), a qual se encontra explicada detalhadamente, abaixo, na sua subseção.

No final da execução do algoritmo, estas estatísticas são processadas pela seguinte função *calculate_stats()*:

```
1.  def calculate_stats():
2.
3.      global best_per_gen
4.      global avg_per_gen
5.      global diversity
6.      global avgBest
7.      global stdBest
8.      global avgGen
9.      global stdGen
10.     global maxFit
11.     global minFit
12.     global genMin
13.     global genMax
14.     global avgDiversity
15.     global stdDiversity
16.
17.     avgBest = numpy.mean(best_per_gen)
18.     stdBest = numpy.std(best_per_gen)
19.     avgGen = numpy.mean(avg_per_gen)
20.     stdGen = numpy.std(avg_per_gen)
21.     maxFit = max(best_per_gen)
22.     minFit = min(best_per_gen)
23.     genMax = best_per_gen.index(maxFit)
24.     genMin = best_per_gen.index(minFit)
25.     avgDiversity = numpy.mean(diversity)
26.     stdDiversity = numpy.std(diversity)
```

Esta função calcula não só médias e respetivos desvio padrão das diversas estatísticas recolhidas pela *update_stats()* (linhas 17-20 e 25-26), bem como o melhor e o pior individuo de entre os melhores de cada geração (linhas 21-24).

Diversidade

A diversidade é uma das estatísticas mais importantes para avaliar o funcionamento do algoritmo. Esta estatística representa o quão diversa é a população, apresentando essa diversidade de forma normalizada.

Nesta implementação a diversidade é calculada comparando cada individuo da população com os restantes (linha 15). De forma a efetuar esta comparação recorro à seguinte função do código do Apêndice B:

```
1.  def hamming_distance(v1, v2):
2.      return sum([1 for i in range (len(v1)-1) if v1[i] != v2 [i]])# heuristic=v[-1]
```

Esta função devolve a distância de Hamming entre os dois indivíduos (v1 e v2). A distância de Hamming é um método para comparação de strings binárias com base no número de bits diferentes [46]. Embora os indivíduos não sejam strings binárias, considero-o um método adequado, pois devolve o número de genes diferentes entre dois indivíduos.

3.5.2.2 Testes iniciais

Nos testes iniciais foi utilizada a seguinte configuração base:

- Cenário:
  - Número de AP: 2
  - Número de STA por AP: 4
  - Duração: 10s
  - Taxa de *upload* de cada STA: 100 Kbps
- População: 25
- Gerações: 5

Observando os resultados dos testes, rapidamente reparei que existia algum problema na minha implementação. Em particular, observando os resultados dos testes abaixo deparei-me com o seguinte problema: as estatísticas dos mesmos indicavam que o melhor indivíduo de todas as gerações, ou seja, o indivíduo com o fitness mínimo, estava presente numa geração, mas não na seguinte.

Single gene mutation operator:

- **Máximo do melhor:** 2.536202429342271e-06 **-> geração**: 1/5
- **Mínimo do melhor:** 2.3968966391481406e-06 **-> geração**: 0/5
- **Média do melhor:** 2.424290557269672e-06 **+-** 5.020663727187831e-08
- **Média geracional:** 4.726570109374378e-06 **+-** 1.3534205390114886e-06
- **Diversidade média:** 0.9805555555555555 **+-** 0.008695819912499176

Multi-gene probabilistic mutation operator - Probabilidade de mutação 50%:

- **Máximo do melhor:** 2.5446834640678233e-06 **-> geração:** 5/5
- **Mínimo do melhor:** 2.3922202416320062e-06 **-> geração:** 3/5
- **Média do melhor:** 2.4500119715916124e-06 **+-** 6.218680853872704e-08
- **Média geracional:** 5.738491971541811e-06 **+-** 1.4355385342507625e-06
- **Diversidade média:** 0.9805555555555555 **+-** 0.008695819912499176

Após uma observação direta aos indivíduos de cada geração, confirmei que os melhores indivíduos de uma geração, nem sempre se encontravam presentes na seguinte. Tal não

deveria acontecer, pois a minha implementação pressupõe a manutenção dos mesmos entre gerações.

Outro problema com que me deparei foi a complexidade temporal da execução do algoritmo. Devido ao método de geração de descendentes, assumindo uma população de *N* indivíduos, cada geração necessitava de executar *N* simulações apenas para selecionar os descendentes.

O primeiro problema levou-me a encontrar um defeito no código que foi rapidamente resolvido, já o segundo problema necessitou de uma abordagem um pouco mais complexa. De forma a diminuir a quantidade de simulações por geração, passei a gerar apenas o número de descendentes necessários, e os seus progenitores passaram a ser selecionados através de um método probabilístico de *rank selection*.

Após as modificações acima referidas, voltei a executar os testes, obtendo, desta vez resultados mais próximos do esperado. Deparei-me, no entanto, com novos casos de melhores soluções que desapareceram de uma geração para a seguinte, no seguinte teste:

Single gene mutation operator:

- **Máximo do melhor:** 4.81434298299821e-06 **-> geração**: 2/5
- **Mínimo do melhor:** 4.800171132485065e-06 **-> geração**: 0/5
- **Média do melhor:** 4.809619032827162e-06 **+-** 6.6806743998709615e-09
- **Média geracional:** 3.0744573456182586e+17 **+-** 4.4764774652013235e+17
- **Diversidade média:** 0.9838888888888889 **+-** 0.008695819912499174

Após correção do defeito responsável voltei a executar os testes. No entanto, analisando manualmente os indivíduos de cada geração, reparei que existiam diversos casos de indivíduos repetidos, em ambos os operadores. Estes casos encontravam-se presentes em todos os testes, mas principalmente no MGMO com probabilidades de 10% e 30%. Por este motivo, descartei estas configurações de testes futuros, ficando apenas com o SGMO e o MGMO 50%.

3.5.2.3 Testes intermédios

Após uma nova verificação do código, bem como pequenas melhorias e correções, executei novos testes, mas desta vez ao longo de 10 gerações, recorrendo à seguinte configuração:

- Cenário:
    - Número de AP: 2
    - Número de STA por AP: 4
    - Duração: 10s
    - Taxa de *upload* de cada STA: 100 Kbps
- População: 25
- Gerações: 10

Através destes testes obtive as seguintes estatísticas:

Single gene mutation operator:

- **Máximo do melhor:** 4.935857009301368e-06 **-> geração:** 0/10
- **Mínimo do melhor:** 4.800171132485065e-06 **-> geração:** 6/10
- **Média do melhor:** 4.834883620556745e-06 **+-** 4.941334037338508e-08
- **Média geracional:** 1.2133075957952207e-05 **+-** 6.317397800480049e-06
- **Diversidade média:** 0.9757575757575758 **+-** 0.01119982946216747

Multi-gene probabilistic mutation operator - Probabilidade de mutação 50%:

- **Máximo do melhor:** 5.090438007099392e-06 **-> geração:** 0/10
- **Mínimo do melhor:** 4.800171132485065e-06 **-> geração:** 7/10
- **Média do melhor:** 4.834108738546203e-06 **+-** 8.131585328685223e-08
- **Média geracional:** 1.5973154684036716e-05 **+-** 1.4412214668178456e-05
- **Diversidade média:** 0.9827272727272728 **+-** 0.007496555682941204

Com recurso aos resultados dos testes elaborei, ainda, os seguintes gráficos:

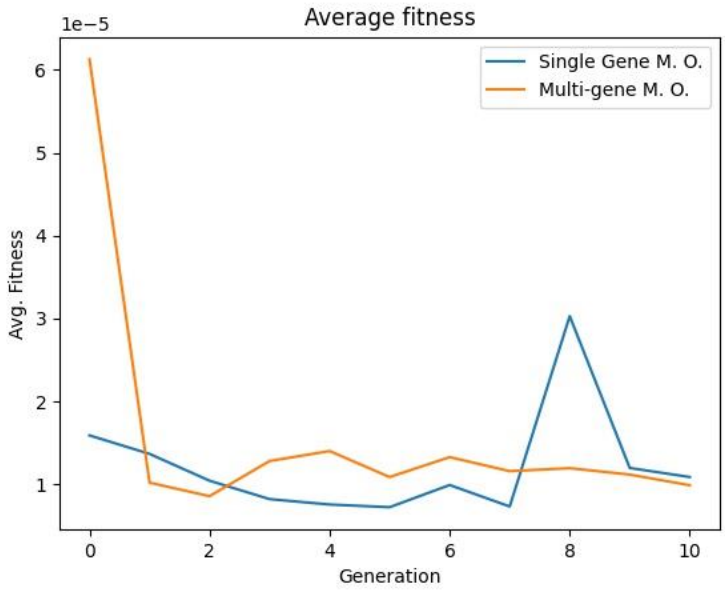


Figura 12 – Média geracional por operador de mutação

Analisando a Figura 12 é possível verificar que o SGMO tende a apresentar melhores médias geracionais, as quais tendem a manter-se baixas com exceção de pequenos picos provocados pela inserção de indivíduos estrangeiros. Já o MGMO apresenta em média piores resultados por geração.

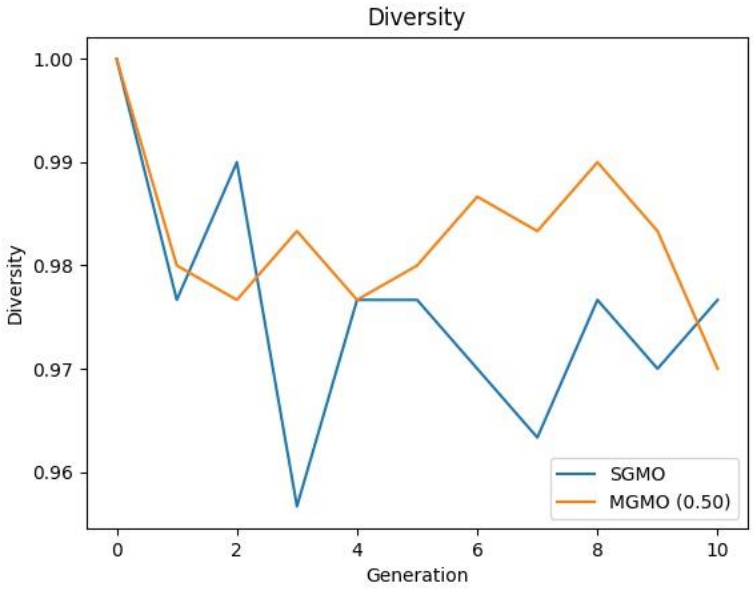


Figura 13 – Diversidade média por operador de mutação

Quanto à diversidade, observando a Figura 13, verifico que o MGMO apresenta uma diversidade superior ao SGMO. No entanto esta deve-se principalmente à constante geração de indivíduos quase aleatórios através dos pais, devido à sua elevada probabilidade de mutação, o que explica as piores médias geracionais da Figura 12.

Estes resultados levaram-me a selecionar o SGMO, pois acredito que em conjunção com a injeção de indivíduos estrangeiros providência os melhores resultados.

3.6 Resultados preliminares e publicação

De forma a recolher dados para elaboração um artigo, foi executada uma versão preliminar do algoritmo evolucionário (EvoMCS_v1). Esta versão serviu também como teste final ao algoritmo e teve quatro objetivos principais:

1. Obter resultados para elaboração e publicação de um artigo.
2. Validar globalmente algoritmo;
3. Avaliar desempenho do cenário intermédio;
4. Avaliar desempenho de heurística entre cenários;

Este artigo teve como alvo a conferência MSWIM 21: Modeling, Analysis and Simulation of Wireless and Mobile Systems [47], e encontra-se no Apêndice C.

O algoritmo evolucionário foi executado utilizando os seguintes cenários:

- Cenário básico:
    - Número de AP: 4
    - Número de STA por AP: 4
    - Duração: 10s
    - Taxa de *upload* de cada STA: 100 Kbps

- Cenário intermédio (teste):
    - Número de AP: 9
    - Número de STA por AP: 16
    - Duração: 10s
    - Taxa de *upload* de cada STA: 100 Kbps

Os cenários acima definidos são idênticos aos definidos na secção 3.3.1, no entanto, o cenário intermédio possui duração de 10 segundos em vez de 3 segundos.

Devido a constrangimentos temporais a execução do algoritmo evolucionário foi interrompida a meio do processo de avaliação dos indivíduos após troca. No entanto, tal avaliação foi suficiente para verificar o funcionamento deste mecanismo.

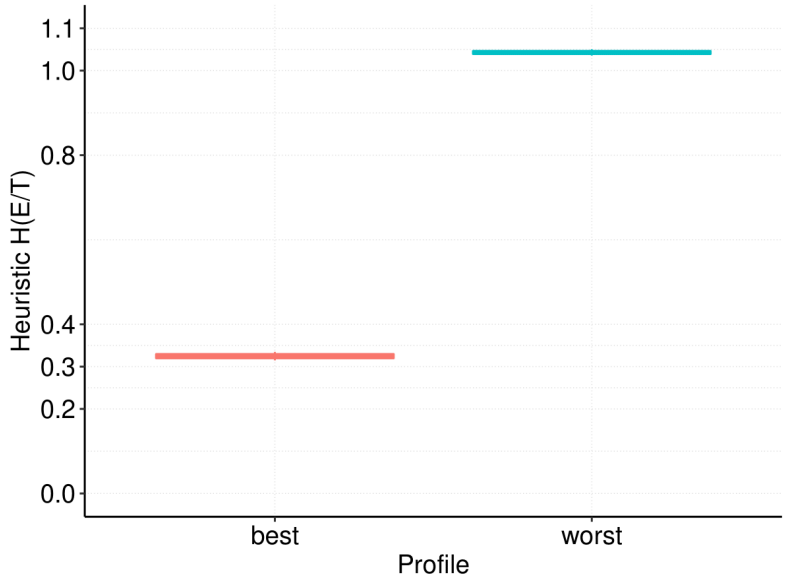


Figura 14 – Heurística (J/bit) no cenário intermédio de teste (melhor e pior)

De forma a cumprir os objetivos 2 e 3, foi efetuada uma avaliação manual ao pior e melhor individuo da vigésima geração. Esta foi constituída pela execução do cenário intermédio com as respetivas configurações e posterior análise dos resultados.

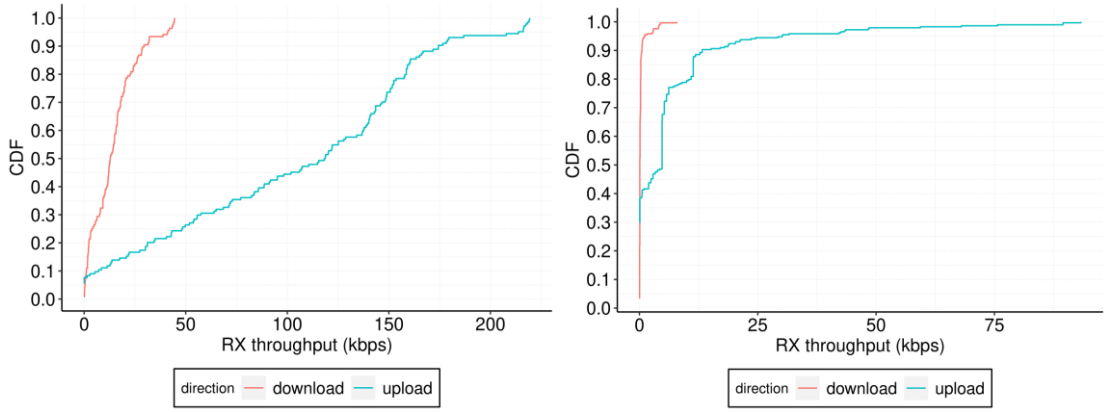


Figura 15 – CDF do Throughput no cenário intermédio de teste (melhor e pior)

Analisando os resultados, foi possível observar que existe não só uma diferença significativa no consumo energético por bit (J/bit), entre ambas as configurações (Figura 14), como no *throughput* (Figura 15).

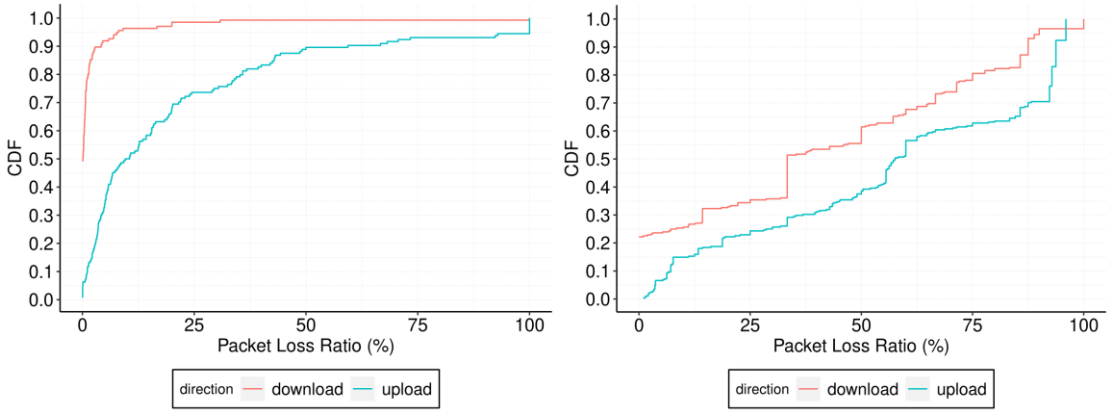


Figura 16 – CDF do PLR no cenário intermédio de teste (melhor e pior)

Apesar dos resultados positivos, segundo a distribuição cumulativa de valores (CDF), existe uma probabilidade significativa (10%) de existirem dispositivos com PLR acima de 50% (Figura 16). Num sistema crítico, tal número deverá ser o mais baixo possível. Isto deve-se ao facto de tais sistemas requererem um elevado nível de fiabilidade.

De forma a combater tais perdas de pacotes, procedi à alteração da heurística composta utilizada. A heurística v2, é idêntica à primeira versão, no entanto inclui uma penalização igual à percentagem de pacotes perdidos.

Embora o artigo tenha sido rejeitado, no dia 28 de Agosto de 2021, recebi feedback extremamente útil que pretendo utilizar numa nova submissão, na EvoApplications [48].

Capítulo 4
Resultados

De forma a obter configurações otimizadas foram realizadas experiências utilizando duas versões do algoritmo evolucionário (EvoMCS). A diferença entre estas versões reside na heurística utilizada para calcular o *fitness* dos indivíduos. A versão 1 faz uso da Heurística (secção 3.2.1.1), já a versão 2 faz uso da heurística atualizada definida na secção 3.2.1.2.

A principal diferença entre as heurísticas reside na existência ou não de uma penalização igual ao rácio de perda de pacotes (PLR). Como mencionado anteriormente, o principal objetivo desta penalização é combater a probabilidade significativa de perda de pacotes presente nalguns indivíduos, melhorando os resultados.

O EvoMCS_v1 foi avaliado na VM2, já o Evo_MCS_v2 foi executado na VM1.

4.1 Configurações otimizadas

Nas experiências realizadas obteve-se duas configurações otimizadas. Estas configurações são baseadas nas duas versões da heurística utilizadas no algoritmo evolucionário.

Tabela 11 – Configurações otimizadas

Versão	Tecnol.	Freq.	Banda	MCS	GI	RTS	Extended Block Ack	OBSS/PD	Prot.
v1	.11ax	6 GHz	20 MHz	3	800 ns	0	0	0	TCP
v2	.11ax	6 GHz	160 MHz	4	800 ns	0	0	1	TCP

Ambos os algoritmos apresentam soluções bastante semelhantes. A principal diferença entre as configurações reside na utilização de *Spatial Reuse* com *BSS Colouring* (OBSS/PD) em conjunção com uma maior largura de banda do canal (160 MHz), por parte do EvoMCS_v2. A utilização de OBSS/PD diminui a interferência entre APs, atribuindo-lhes diferentes cores. Já a maior largura de banda do canal leva a uma disponibilização de *data rates* mais elevados, aumentando o *throughput* na camada aplicacional.

4.2 Cenário Básico

Como mencionado na secção 3.3.1, o algoritmo evolucionário faz uso de diferentes cenários com diversos níveis de complexidade. A complexidade do cenário de avaliação utilizado aumenta com o número de gerações. Nas primeiras 25 gerações os indivíduos são avaliados no cenário básico.

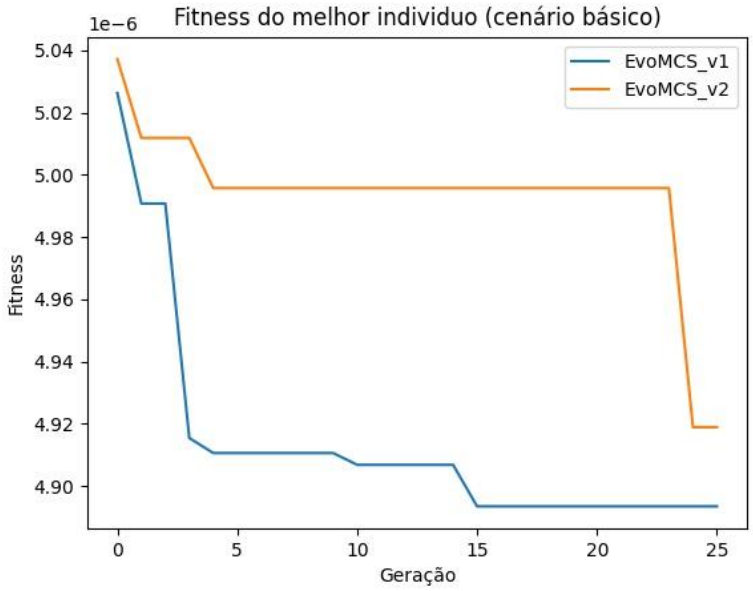


Figura 17 – Fitness por geração no cenário básico (melhor individuo)

Analisando o fitness médio do melhor individuo de cada geração (Figura 17), podemos verificar que existe uma diferença significativa entre ambos os algoritmos. Tal diferença deve-se à penalização implementada na segunda versão. Esta penalização é proporcional à perda de pacotes.

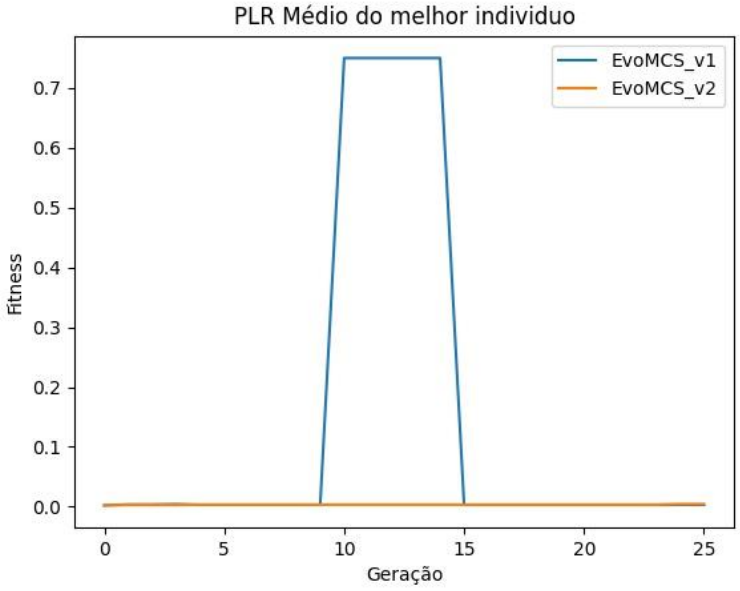


Figura 18 – PLR no cenário básico (melhor individuo)

Analisando a Figura 18 é possível observar uma diferença considerável entre ambas as implementações do algoritmo evolucionário. A introdução da penalização baseada em *packet loss ratio* (PLR) aparenta contribuir para uma diminuição considerável da amplitude das flutuações de PLR. Esta contribuição resulta na manutenção de valores extremamente reduzidos da mesma ao longo das primeiras 25 gerações.

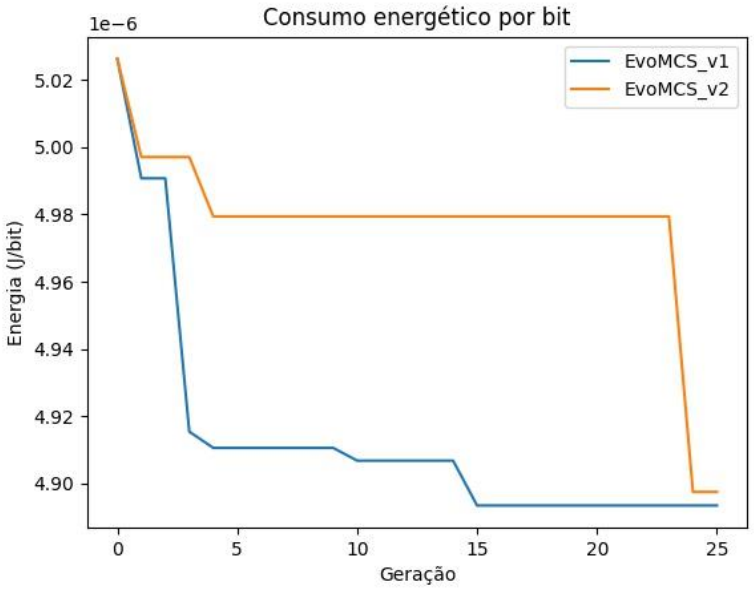


Figura 19 – Consumo energético médio por bit no cenário básico (melhor individuo)

O consumo energético médio por bit, é significativamente mais elevado, ao longo das primeiras gerações, quando utilizada a nova versão da heurística. No entanto, este resultado sofre alterações nas gerações finais do cenário básico. Na vigésima-quarta geração, verifica-se uma descida acentuada do consumo energético para valores bastante próximos dos do EvoMCS_v1.

Tendo os resultados anteriores em conta, considero que o EvoMCS_v2 possui um melhor desempenho do que a primeira versão, pois apesar de obter resultados ligeiramente piores em termos energéticos, tal diferença é insignificante quando comparada com o efeito obtido na diminuição do PLR.

4.3 Cenário intermédio e complexo

Devido a constrangimentos de memória, foi necessário remover paralelismo na avaliação de indivíduos no cenário intermédio e no cenário complexo. Esta remoção levou a um aumento significativo da complexidade temporal do algoritmo evolucionário. Por esse motivo, após três semanas de execução, o EvoMCS_v2 ainda se encontrava a avaliar a trigésima geração de indivíduos (5ª no cenário intermédio). De forma a obter resultados relevantes atempadamente, foi efetuada uma paragem na execução e forçada uma troca para o cenário complexo após a vigésima-nona geração.

Devido a limitações de hardware da VM2, não foi possível testar os indivíduos do EvoMCS_v1 no cenário intermédio. Por esse motivo, nesta secção, utilizo dados resultantes da elaboração do artigo. Estes dados, encontram-se disponíveis na pasta results do repositório git (https://github.com/MiguelArieiro/critical_IoT), e contêm a execução do melhor elemento da vigésima geração do EvoMCS_v1. Este individuo apresenta, também, o melhor desempenho após as primeiras 25 gerações resultantes da última execução do algoritmo (secção 4.2).

Tabela 12 – Consumo energético por bit no cenário intermédio

Algoritmo	**Consumo energético por bit (J/bit)**
EvoMCS_v1 (artigo)	7.98347424e-6
EvoMCS_v2 (29º geração)	8.14900261e-6

Analisando os dados presentes na Tabela 12 podemos verificar que existe um aumento de 2.07% no consumo energético por bit transmitido. Este aumento é extremamente insignificante e possivelmente equilibrado pela menor perda de pacotes resultante da utilização da nova heurística composta.

Observando os resultados dos testes, verifica-se que o melhor individuo se mantem constante ao longo das quatro gerações. Este elemento encontra-se definido na Tabela 11 da secção 4.1, e os resultados da sua execução encontram-se presentes na Tabela 13.

Tabela 13 – Métricas do melhor indivíduo do EvoMCS_v2

Cenário	**Consumo energético (J)**	***Throughput* (bit/s)**	***Packet Loss Ratio***	***Fitness***
Intermédio	0.942041	115602.0	0.284979	1.047129722789398e-05
Complexo	0.936965	150900.0	0.131134	7.02341264e-06

Os resultados presentes na Tabela 12 apresentam uma ligeira melhoria no desempenho da mesma configuração quando avaliada no cenário mais complexo. Uma das hipóteses para esta diminuição é a maioria da interferência, e consequente perda de pacotes, ser causada entre diferentes sub-redes. Logo, a diminuição do número de sub-redes para 4 com maior número de STA por AP, leva a um aumento do *throughput* na camada aplicacional.

Capítulo 5
Conclusão

Ao longo deste projeto, realizei diversas pesquisas de forma a familiarizar-me não só com o estado da arte das tecnologias IoT, como com os protocolos de comunicação mais utilizados, as tecnologias 5G e IEEE 802.11ax, e as diversas métricas associadas às mesmas.

Com recurso a esta investigação e a um processo de aprendizagem associado à utilização do simulador de redes ns-3, ao longo do segundo semestre, desenvolvi um algoritmo evolucionário para busca de configurações otimizadas de dispositivos em cenários críticos de elevada densidade. Este algoritmo faz uso de um cenário de simulação de redes IoT de elevada densidade para tecnologias IEEE 802.11ax e IEEE 802.11n, desenvolvido com recurso ao simulador de redes ns-3.

Recorrendo a estas ferramentas obtive resultados interessantes que suportam a utilização da tecnologia IEEE 802.11ax com *BSS colouring*, em conjunto com o protocolo de comunicação MQTT (TCP), em cenários críticos de elevada densidade.

Penso que as maiores limitações deste projeto residem na complexidade espacial e temporal da execução do cenário de simulação. Tais limitações, não só tiveram um impacto significativo na minha capacidade para ajustar os diversos parâmetros do algoritmo evolucionário, e no processo de recolha de resultados, como restringiram significativamente o design do cenário de simulação.

Apesar destes contratempos, espero que estes resultados preliminares possam ser utilizados como base no desenvolvimento de aplicações práticas destas tecnologias em cenários críticos, como é o caso do combate a incêndios florestais, de forma a contribuir para o progresso desta área. Não foi considerada a tecnologia 5G, inicialmente planeada. No entanto, o algoritmo evolucionário (EvoMCS) foi desenvolvido de forma modular, tendo em consideração métricas/propriedades especificas do 5G. Isto possibilita a sua fácil adaptação para otimizar as configurações de dispositivos móveis nestas redes.

A nível de desafios, será interessante evoluir o algoritmo genético para considerar cenários com a múltiplas tecnologias (5G, WiFi 6, e outras) de forma a maximizar aspetos de resiliência e até de performance das aplicações nos cenários críticos.

Referências

1. D. Raposo, A. Rodrigues, S. Sinche, J. S. Silva, and F. Boavida, “Industrial IoT monitoring: Technologies and architecture proposal,” Sensors (Switzerland), 2018
2. E. Khorov, A. Kiryanov, A Lyakhov, G. Bianchi, “A Tutorial on IEEE 802.11ax High Efficiency WLANs”, IEEE Comunications Surveys & Tutorials, Vol. 21, No. 1, First Quarter 2019
3. M. Natkaniecm, Ł. Prasnal, M. Szymakowski, “A Performance Analysis of IEEE 802.11ax Networks”, International Journal of Electronics and Telecomunications, 2020, vol. 66, No. 1, PP. 225-230
4. G. A. Akpakwu, B. J. Silva, G.P. Hancke, A. M. Abu-Mahfouz, “A Survey on 5G Networks for the Internet of Things: Communication Technologies and Challenges”, IEEE Access, 2017
5. Masek, P., Stusek, M., Zeman, K., Drapela, R., Ometov, A., & Hosek, J. (2019). Implementation of 3GPP LTE Cat-M1 Technology in NS-3: System Simulation and Performance. In 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops, ICUMT 2019 (International Congress on Ultra Modern Telecommunications and Control Systems and Workshops). IEEE
6. “Wi-Fi Alliance® introduces Wi-Fi 6”, https://www.wi-fi.org/news-events/newsroom/wi-fi-alliance-introduces-wi-fi-6, 2018 (acedido 20/05/2020)
7. AccessAgility, “AccessAgility Blog”, https://www.accessagility.com/hs-fs/hubfs/Images/Blog%20Posts/50%20-%20Demystifying%20802.11ax%20-%20WiFi%206/bss-coloring.png?width=801&name=bss-coloring.png, 2019 (obtido 07/08/2020)
8. Sharma, Priya & Singh, Gurpreet. (2016). “Comparison of Wi-Fi IEEE 802.11 Standards Relating to Media Access Control Protocols”. International Journal of Computer Science and Information Security,. 14. 856-862
9. Selinis, Ioannis & Katsaros, Konstantinos & Vahid, Seiamak & Tafazolli, Rahim. (2018). “Control OBSS/PD Sensitivity Threshold for IEEE 802.11ax BSS Color”. 10.1109/PIMRC.2018.8580778
10. Toby Jaffey, “MQTT and CoAP, IoT Protocols, https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php (acedido 11/04/2020)
11. The HiveMQ Team, https://www.hivemq.com/blog/mqtt-security-fundamentals-x509-client-certificate-authentication/, 2015 (acedido 05/06/2020)
12. https://www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/qos.html, 2018 (acedido 24/05/2020)
13. The HiveMQ Team, https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/, 2015 (acedido 08/06/2020)
14. Intel support page, https://www.intel.com/content/www/us/en/support/articles/000005725/network-and-i-o/wireless-networking.html, (acedido 12/06/2020)
15. Cisco, “802.11ac: The Fifth Generation of Wi-Fi Technical White Paper”, https://www.cisco.com/c/dam/en/us/products/collateral/wireless/aironet-3600-series/white-paper-c11-713103.pdf, 2018 (obtido 11/06/2020)
16. GSMA, “NB-IOT Deployment Guide to Basic Feature Set Requirements”, https://www.gsma.com/iot/wp-content/uploads/2019/07/201906-GSMA-NB-IoT-Deployment-Guide-v3.pdf, June 2019 (obtido 17/06/2020)
17. GSMA, “LTE-M Deployment Guide to Basic Feature Set Requirements”, https://www.gsma.com/iot/wp-content/uploads/2019/08/201906-GSMA-LTE-M-Deployment-Guide-v3.pdf, June 2019 (obtido 17/06/2020)
18. Site oficial do protocolo MQTT, http://mqtt.org/ (acedido 18/06/2020)
19. Site oficial do protocolo CoAP, https://coap.technology/ (acedido 18/06/2020)
20. https://www.pickdata.net/news/mqtt-vs-coap-best-iot-protocol (acedido 18/06/2020)
21. Ronan Le Bras, “What is the Difference in Data Throughput between LTE-M/NB-IoT and 3G or 4G?” https://www.gsma.com/iot/resources/what-is-the-difference-in-data-throughput-between-lte-m-nb-iot-and-3g-or-4g/, 29 Oct 2019 (acedido 20/06/2020)

22. Haltian, “NB1 vs. NB2”, https://haltian.com/connectivity-garage/iot-protocols-overview/nb-iot/nb1-vs-nb2/ (acedido 21/06/2020)
23. “IEEE P802.11 EXTREMELY HIGH THROUGHPUT Study Group”, http://www.ieee802.org/11/Reports/ehtsg_update.htm (acedido 23/06/2020)
24. E. Khorov, I. Levitsky and I. F. Akyildiz, "Current Status and Directions of IEEE 802.11be, the Future Wi-Fi 7," in IEEE Access, vol. 8, pp. 88664-88688, 2020, doi: 10.1109/ACCESS.2020.2993448, 2020 (obtido 24/06/2020)
25. Wi-Fi Alliance, “Wi-Fi HaLow”, https://www.wi-fi.org/discover-wi-fi/wi-fi-halow (acedido 24/06/2020)
26. Zhang, Qi & Fitzek, Frank. (2015). Mission Critical IoT Communication in 5G. 35-41. 10.1007/978-3-319-27072-2_5. (obtido 26/06/2020)
27. ADAI, UC, O COMPLEXO DE INCÊNDIOS DE PEDRÓGÃO GRANDE E CONCELHOS LIMÍTROFES, INICIADO A 17 DE JUNHO DE 2017
28. 5G-LENA simulador, https://5g-lena.cttc.es/ (acedido 28/06/2020)
29. K. G. Panda, S. Das, D. Sen, and W. Arif, “Design and Deployment of UAV-Aided Post-Disaster Emergency Network,” *IEEE Access*, vol. 7, pp. 102985–102999, 2019.
30. M. O. Ojo, S. Giordano, G. Procissi, and I. N. Seitanidis, “A Review of Low-End, Middle-End, and High-End Iot Devices,” *IEEE Access*, vol. 6, pp. 70528–70554, 2018.
31. S. Sendra, L. García, J. Lloret, I. Bosch, and R. Vega-Rodríguez, “LoRaWAN network for fire monitoring in rural environments,” *Electron.*, vol. 9, no. 3, 2020.
32. O. M. Bushnaq, A. Chaaban, and T. Y. Al-Naffouri, “The Role of UAV-IoT Networks in Future Wildfire Detection,” pp. 1–13, Jul. 2020.
33. C. Esposito *et al.*, “On the disaster resiliency within the context of 5G networks: The RECODIS experience,” vol. 15127, pp. 3–6, 2018.
34. W. Yu, H. Xu, J. Nguyen, E. Blasch, A. Hematian, and W. Gao, “Survey of Public Safety Communications: User-Side and Network-Side Solutions and Future Directions,” *IEEE Access*, vol. 6, pp. 70397–70425, 2018.
35. Ruckus, “Dealing with density: The model to Small-Cell Architectures”, white paper, 2018. Available at: https://www.commscope.com/globalassets/digizuite/1523-1353-wp-dealing-with-density.pdf
36. https://www.nsnam.org/doxygen/classns3_1_1_wifi_radio_energy_model.html (acedido 10/02/2020)
37. Daniel Halperin, Ben Greenstein, Anmol Sheth, David Wetherall, "Demystifying 802.11n power consumption", Proceedings of HotPower'10
38. Vítor Manuel Henriques Bernardo, “Energy Efficient Multimedia Communications in IEEE 802.11 Networks”
39. https://www.nsnam.org/docs/models/html/wifi-design.html (acedido 09/04/2021)
40. H. Yang, D. Deng and K. Chen, "On Energy Saving in IEEE 802.11ax", in IEEE Access, vol. 6, pp. 47546-47556, 2018, doi: 10.1109/ACCESS.2018.2865763.
41. https://support.huawei.com/enterprise/en/doc/EDOC1100102755 (acedido 17/05/2021)
42. Leitão, António & Machado, Penousal. (2015). Mate Choice in Evolutionary Computation. 10.1007/978-3-319-20883-1_7. 2.
43. C. Darwin. “The origin of species”, 1859
44. S. E. Collier, "The Emerging Enernet: Convergence of the Smart Grid with the Internet of Things," in IEEE Industry Applications Magazine, vol. 23, no. 2, pp. 12-16, March-April 2017, doi: 10.1109/MIAS.2016.2600737.
45. Cisco Meraki, “High Density Wi-Fi Deployments”, https://documentation.meraki.com/Architectures_and_Best_Practices/Cisco_Meraki_Best_Practice_Design/Best_Practice_Design_-_MR_Wireless/High_Density_Wi-Fi_Deployments (acedido 02/09/2021)
46. Hamming Distance, https://www.sciencedirect.com/topics/engineering/hamming-distance (acedido 05/09/2021)
47. MSWIM, https://dl.acm.org/conference/mswim (acedido 05/09/2021)
48. EvoApplications, http://www.evostar.org/2022/evoapps/ (acedido 02/09/2021)
49. https://www.centurylink.com/home/help/internet/wireless/which-frequency-should-you-use.html (acedido 06/09/2021)
50. https://www.router-switch.com/faq/why-is-5-ghz-wi-fi-faster-than-2-4-ghz-wi-fi.html (acedido 06/09/2021)

Apêndices

Apêndice A

```
1.  #include "ns3/command-line.h"
2.  #include "ns3/config.h"
3.  #include "ns3/string.h"
4.  #include "ns3/spectrum-wifi-helper.h"
5.  #include "ns3/ssid.h"
6.  #include "ns3/mobility-helper.h"
7.  #include "ns3/application-container.h"
8.  #include "ns3/multi-model-spectrum-channel.h"
9.  #include "ns3/wifi-net-device.h"
10. #include "ns3/ap-wifi-mac.h"
11. #include "ns3/he-configuration.h"
12. #include "ns3/wifi-radio-energy-model-helper.h"
13. #include "ns3/energy-source-container.h"
14. #include "ns3/li-ion-energy-source-helper.h"
15. #include "ns3/li-ion-energy-source.h"
16.
17. #include <iomanip>
18. #include <iostream>
19.
20. #include <vector>
21. #include <cmath>
22. #include <regex>
23.
24. #include "ns3/flow-monitor-helper.h"
25. #include "ns3/ipv4-flow-classifier.h"
26. #include "ns3/internet-module.h"
27.
28. #include "ns3/on-off-helper.h"
29. #include "ns3/packet-sink-helper.h"
30. #include "ns3/packet-sink.h"
31.
32. #include "ns3/rng-seed-manager.h"
33.
34. #include "ns3/rectangle.h"
35.
36. using namespace ns3;
37.
38. #define MAX_NODES 2048
39.
40. bool verbose = false;
41.
42. std::vector<uint64_t> bytesReceived(MAX_NODES);
43.
44. void MonitorSniffRx(std::string context, Ptr<const Packet> packet, uint16_t channel
    FreqMhz,
45.                     WifiTxVector txVector, MpduInfo aMpdu, SignalNoiseDbm signalNoi
    se, uint16_t staId)
46. {
47.   std::string sub = context.substr(10);
48.   uint32_t pos = sub.find("/Device");
49.   uint32_t nodeId = atoi(sub.substr(0, pos).c_str());
50.   bytesReceived[nodeId] += packet->GetSize();
51. }
52.
53. // std::vector<uint64_t> bytesReceivedPS(MAX_NODES);
54.
55. // void PacketSinkRx (std::string context, Ptr< const Packet > packet, const Addres
    s &address)
56. // {
57. //   std::string sub = context.substr(10);
58. //   uint32_t pos = sub.find("/Application");
59. //   uint32_t nodeId = atoi(sub.substr(0, pos).c_str());
60. //   bytesReceivedPS[nodeId] += packet->GetSize();
61. // }
```

```
62.
63. // Trace function for remaining energy at node.
64. void RemainingEnergy(std::string context, double oldValue, double remainingEnergy)
65. {
66.
67.   int32_t nodeId = std::stoi(context);
68.   if (verbose)
69.   {
70.     NS_LOG_UNCOND(Simulator::Now().GetSeconds()
71.                   << "s " << nodeId << " Current remaining energy = " << remainingE
    nergy << "J");
72.   }
73.   //fprintf("%lf,%lf\n", Simulator::Now().GetSeconds(), remainingEnergy);
74. }
75.
76. // Trace function for total energy consumption at node.
77. void TotalEnergy(std::string context, double oldValue, double totalEnergy)
78. {
79.   int32_t nodeId = std::stoi(context);
80.   if (verbose)
81.   {
82.     NS_LOG_UNCOND(Simulator::Now().GetSeconds()
83.                   << "s " << nodeId << " Total energy consumed by radio = " << tota
    lEnergy
84.                   << "J");
85.   }
86.   //fprintf(energyConsumedFile[nodeId], "%lf,%lf\n", Simulator::Now().GetSeconds(),
     totalEnergy);
87. }
88. /*****************************************************************************/
89.
90. int main(int argc, char *argv[])
91. {
92.   // double d1 = 30.0; // meters
93.   // double d2 = 30.0; // meters
94.   // double d3 = 150.0; // meters
95.   uint32_t seed = 1;
96.   uint32_t runs = 3;
97.   bool tracing = false;
98.   bool walk = false;
99.
100.    double duration = 10.0;           // seconds
101.    double powAp = 21.0;              // dBm
102.    double powSta = 10.0;             // dBm
103.    double ccaEdTrAp = -62.0;         // dBm
104.    double ccaEdTrSta = -62.0;        // dBm
105.    double rxSensAp = -92.0;          // dBm
106.    double rxSensSta = -92.0;         // dBm
107.    uint32_t tcpPayloadSize = 60;     // bytes
108.    uint32_t udpPayloadSize = 40;     // bytes
109.    int32_t mcs = 9;                  // MCS value
110.    uint64_t dataRate = 100000;       // bit/s
111.    double distance = 25;             // mobility model side = 2 * distance
112.    int technology = 0;               // technology to be used 802.11ax = 0, 802.11
    n = 1;
113.    int frequency = 5;                // frequency selection
114.    int channelWidth = 20;            // channel number
115.    int numApAntennas = 4;            // number of AP Antenas
116.    int numApRxSpatialStreams = 4;    // number of AP Rx Spatial Streams
117.    int numApTxSpatialStreams = 4;    // number of AP Tx Spatial Streams
118.    int numStaAntennas = 4;           // number of STA Antenas
119.    int numStaRxSpatialStreams = 4;   // number of STA Rx Spatial Streams
120.    int numStaTxSpatialStreams = 4;   // number of STA Tx Spatial Streams
121.    bool enableObssPd = true;         // spatial reuse
122.    double obssPdThreshold = -82.0;   // dBm
123.    bool useUdp = false;              // udp or tcp
124.    bool useRts = false;              // enable RTS/CTS
125.    bool useExtendedBlockAck = false; // enable Extended Block Ack
126.    int guardInterval = 3200;         // guard interval
127.    double batteryLevel = 2000000;    // initial battery energy
```

```
128.
129.    //default ns3 energy values 802.11n (2.4GHz)
130.    double TxCurrentA = 0.38;
131.    double RxCurrentA = 0.313;
132.    double IdleCurrentA = 0.273;
133.    double SleepCurrentA = 0.033;
134.    double CcaBusyCurrentA = 0.273;
135.    double SwitchingCurrentA = 0.273;
136.
137.    uint32_t timeStartServerApps = 1000;
138.    uint32_t timeStartClientApps = 2000;
139.
140.    int32_t numAp = 2;
141.    int32_t numSta = 4;
142.
143.    CommandLine cmd(__FILE__);
144.    cmd.AddValue("verbose", "Activate logging", verbose);
145.    cmd.AddValue("runs", "Sets number of runs", runs);
146.    cmd.AddValue("seed", "Sets RNG seed number", seed);
147.    cmd.AddValue("tracing", "Enable pcap tracing", tracing);
148.
149.    cmd.AddValue("ccaEdTrAp", "CCA-ED Threshold of AP (dBm)", ccaEdTrAp);
150.    cmd.AddValue("ccaEdTrSta", "CCA-ED Threshold of STA (dBm)", ccaEdTrSta);
151.    cmd.AddValue("channelWidth", "Defines wifi channel number", channelWidth);
152.    cmd.AddValue("dataRate", "Data rate (bit/s)", dataRate);
153.    cmd.AddValue("distance", "Distance between networks", distance);
154.    cmd.AddValue("duration", "Duration of simulation (s)", duration);
155.    cmd.AddValue("enableObssPd", "Enable/disable OBSS_PD", enableObssPd);
156.    cmd.AddValue("frequency", "Wifi device frequency. 2 - 2.4GHz, 5 - 5GHz, 6 - 6GH
    z", frequency);
157.    cmd.AddValue("guardInterval", "Set guard interval (ns)", guardInterval);
158.    cmd.AddValue("mcs", "The constant MCS value to transmit HE PPDUs", mcs);
159.    cmd.AddValue("useUdp", "UDP if set to 1, TCP otherwise", useUdp);
160.    cmd.AddValue("batteryLevel", "Initial energy level (J)", batteryLevel);
161.    cmd.AddValue("numAp", "Number of Wifi APs", numAp);
162.    cmd.AddValue("numApAntennas", "Number of AP Antennas", numApAntennas);
163.    cmd.AddValue("numApRxSpatialStreams", "Number of AP Rx Spatial Streams", numApR
    xSpatialStreams);
164.    cmd.AddValue("numApTxSpatialStreams", "Number of AP Tx Spatial Streams", numApT
    xSpatialStreams);
165.    cmd.AddValue("numSta", "Number of Wifi Stations per AP", numSta);
166.    cmd.AddValue("numStaAntennas", "Number of STA Antennas", numStaAntennas);
167.    cmd.AddValue("numStaRxSpatialStreams", "Number of STA Rx Spatial Streams",
168.                 numStaRxSpatialStreams);
169.    cmd.AddValue("numStaTxSpatialStreams", "Number of STA Tx Spatial Streams",
170.                 numStaTxSpatialStreams);
171.    cmd.AddValue("powAp", "Power of AP (dBm)", powAp);
172.    cmd.AddValue("powSta", "Power of STA (dBm)", powSta);
173.    cmd.AddValue("rxSensAp", "RX Sensitivity of AP (dBm)", rxSensAp);
174.    cmd.AddValue("rxSensSta", "RX Sensitivity of STA (dBm)", rxSensSta);
175.    cmd.AddValue("tcpPayloadSize", "TCP packet size", tcpPayloadSize);
176.    cmd.AddValue("technology", "Select technology to be used. 0 = 802.11ax, 1 = 802
    .11n, 3 = 5G",
177.                 technology);
178.    cmd.AddValue("udpPayloadSize", "UDP packet size", udpPayloadSize);
179.    cmd.AddValue("useExtendedBlockAck", "Enable/disable use of Extended Block Ack",
180.                 useExtendedBlockAck);
181.    cmd.AddValue("useRts", "Enable/disable RTS/CTS", useRts);
182.    cmd.AddValue("walk", "Enable STA mobility", walk);
183.    // cmd.AddValue("TxCurrentA", "Transmission current (A)", TxCurrentA);
184.    // cmd.AddValue("RxCurrentA", "Reception current (A)", RxCurrentA);
185.    // cmd.AddValue("SleepCurrentA", "Sleep current (A)", SleepCurrentA);
186.    // cmd.AddValue("IdleCurrentA", "Iddle current (A)", IdleCurrentA);
187.    cmd.Parse(argc, argv);
188.
189.    ns3::RngSeedManager::SetSeed(seed);
190.
191.    if ((numAp * (numSta + 1) + 1) > MAX_NODES)
192.    {
193.      std::cout << "Error: Limit number of nodes: " << MAX_NODES << std::endl;
```

```
194.     }
195.
196.     if (useRts)
197.     {
198.       Config::SetDefault("ns3::WifiRemoteStationManager::RtsCtsThreshold", StringVa
    lue("0"));
199.     }
200.
201.     std::vector<double> avg_throughput(runs);
202.     std::vector<double> avg_energy(runs);
203.     std::vector<double> avg_packet_loss(runs);
204.
205.     for (uint32_t r = 0; r < runs; r++)
206.     {
207.       SeedManager::SetRun(r + 1);
208.
209.       NodeContainer wifiStaNodes;
210.       wifiStaNodes.Create(numSta * numAp);
211.
212.       NodeContainer wifiApNodes;
213.       wifiApNodes.Create(numAp);
214.
215.       //spectrum definition
216.       /*****************************************************************************/
217.       SpectrumWifiPhyHelper spectrumPhy;
218.       Ptr<MultiModelSpectrumChannel> spectrumChannel = CreateObject<MultiModelSpect
    rumChannel>();
219.       Ptr<FriisPropagationLossModel> lossModel = CreateObject<FriisPropagationLossM
    odel>();
220.       spectrumChannel->AddPropagationLossModel(lossModel);
221.       Ptr<ConstantSpeedPropagationDelayModel> delayModel =
222.           CreateObject<ConstantSpeedPropagationDelayModel>();
223.       spectrumChannel->SetPropagationDelayModel(delayModel);
224.
225.       spectrumPhy.SetChannel(spectrumChannel);
226.       spectrumPhy.SetErrorRateModel("ns3::YansErrorRateModel");
227.       spectrumPhy.SetPreambleDetectionModel("ns3::ThresholdPreambleDetectionModel")
    ;
228.
229.       WifiHelper wifi;
230.       //IEEE 802.11ax
231.       if (technology == 0)
232.       {
233.         switch (frequency)
234.         {
235.         case 2:
236.           wifi.SetStandard(WIFI_STANDARD_80211ax_2_4GHZ);
237.           Config::SetDefault("ns3::LogDistancePropagationLossModel::ReferenceLoss",
238.                              DoubleValue(40));
239.
240.           TxCurrentA = 0.333;
241.           RxCurrentA = 0.200;
242.           IdleCurrentA = 0.100;
243.           SleepCurrentA = 0.05;
244.           CcaBusyCurrentA = 0.100;
245.           SwitchingCurrentA = 0.100;
246.           break;
247.         case 5:
248.           wifi.SetStandard(WIFI_STANDARD_80211ax_5GHZ);
249.
250.           TxCurrentA = 0.333;
251.           RxCurrentA = 0.200;
252.           IdleCurrentA = 0.100;
253.           SleepCurrentA = 0.05;
254.           CcaBusyCurrentA = 0.100;
255.           SwitchingCurrentA = 0.100;
256.           break;
257.         case 6:
258.           wifi.SetStandard(WIFI_STANDARD_80211ax_6GHZ);
259.           Config::SetDefault("ns3::LogDistancePropagationLossModel::ReferenceLoss",
```

```
260.                             DoubleValue(48));
261.
262.           TxCurrentA = 0.333;
263.           RxCurrentA = 0.200;
264.           IdleCurrentA = 0.100;
265.           SleepCurrentA = 0.05;
266.           CcaBusyCurrentA = 0.100;
267.           SwitchingCurrentA = 0.100;
268.           break;
269.         default:
270.           std::cout << "Wrong frequency." << std::endl;
271.           return 0;
272.         }
273.
274.         if (enableObssPd)
275.         {
276.           wifi.SetObssPdAlgorithm("ns3::ConstantObssPdAlgorithm", "ObssPdLevel",
277.                                   DoubleValue(obssPdThreshold));
278.         }
279.
280.         std::ostringstream oss;
281.         oss << "HeMcs" << mcs;
282.         wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager", "DataMode",
283.                                      StringValue(oss.str()), "ControlMode",
284.                                      StringValue(oss.str()));
285.       }
286.
287.       //IEEE 802.11n
288.       else if (technology == 1)
289.       {
290.         if (guardInterval != 1)
291.         {
292.           guardInterval = 0;
293.         }
294.
295.         if (numApAntennas > 4)
296.         {
297.           numApAntennas = 4;
298.         }
299.         if (numApRxSpatialStreams > numApAntennas)
300.         {
301.           numApRxSpatialStreams = numApAntennas;
302.         }
303.         if (numApTxSpatialStreams > numApAntennas)
304.         {
305.           numApTxSpatialStreams = numApAntennas;
306.         }
307.
308.         switch (frequency)
309.         {
310.         case 2:
311.           wifi.SetStandard(WIFI_STANDARD_80211n_2_4GHZ);
312.           Config::SetDefault("ns3::LogDistancePropagationLossModel::ReferenceLoss",
313.                              DoubleValue(40.046));
314.
315.           TxCurrentA = 0.38;
316.           RxCurrentA = 0.313;
317.           IdleCurrentA = 0.273;
318.           SleepCurrentA = 0.033;
319.           CcaBusyCurrentA = 0.273;
320.           SwitchingCurrentA = 0.273;
321.           break;
322.         case 5:
323.           wifi.SetStandard(WIFI_STANDARD_80211n_5GHZ);
324.
325.           //val aproximados
326.           TxCurrentA = 0.52364;
327.           RxCurrentA = 0.417229;
328.           IdleCurrentA = 0.374283;
329.           SleepCurrentA = 0.035211;
```

```
330.           CcaBusyCurrentA = 0.374283;
331.           SwitchingCurrentA = 0.374283;
332.           break;
333.         default:
334.           std::cout << "Wrong frequency." << std::endl;
335.           return 0;
336.         }
337.
338.         std::ostringstream oss;
339.         oss << "HtMcs" << mcs;
340.         wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager", "DataMode",
341.                                      StringValue(oss.str()), "ControlMode",
342.                                      StringValue(oss.str()));
343.       }
344.       else
345.       {
346.         //if no supported technology is selected
347.         return 0;
348.       }
349.
350.       spectrumPhy.Set("ChannelWidth", UintegerValue(channelWidth));
351.
352.       WifiMacHelper mac;
353.       Ssid ssid;
354.       std::vector<NetDeviceContainer> staDevices(numAp);
355.       NetDeviceContainer apDevices = NetDeviceContainer();
356.
357.       Ptr<WifiNetDevice> apDevice;
358.       for (int32_t i = 0; i < numAp; i++)
359.       {
360.         ssid = Ssid(std::to_string(i)); //The IEEE 802.11 SSID Information Element.
361.
362.         staDevices[i] = NetDeviceContainer();
363.
364.         //STA creation
365.         spectrumPhy.Set("Antennas", UintegerValue(numStaAntennas));
366.         spectrumPhy.Set("MaxSupportedTxSpatialStreams", UintegerValue(numStaTxSpati
    alStreams));
367.         spectrumPhy.Set("MaxSupportedRxSpatialStreams", UintegerValue(numStaRxSpati
    alStreams));
368.
369.         spectrumPhy.Set("TxPowerStart", DoubleValue(powSta));
370.         spectrumPhy.Set("TxPowerEnd", DoubleValue(powSta));
371.         spectrumPhy.Set("CcaEdThreshold", DoubleValue(ccaEdTrSta));
372.         spectrumPhy.Set("RxSensitivity", DoubleValue(rxSensSta));
373.
374.         mac.SetType("ns3::StaWifiMac", "Ssid", SsidValue(ssid), "ActiveProbing",
375.                     BooleanValue(false));
376.
377.         for (int32_t j = 0; j < numSta; j++)
378.         {
379.           staDevices[i].Add(
380.               wifi.Install(spectrumPhy, mac, wifiStaNodes.Get(i * numSta + j)));
381.         }
382.
383.         //AP creation
384.         spectrumPhy.Set("Antennas", UintegerValue(numApAntennas));
385.         spectrumPhy.Set("MaxSupportedTxSpatialStreams", UintegerValue(numApTxSpatia
    lStreams));
386.         spectrumPhy.Set("MaxSupportedRxSpatialStreams", UintegerValue(numApRxSpatia
    lStreams));
387.
388.         spectrumPhy.Set("TxPowerStart", DoubleValue(powAp));
389.         spectrumPhy.Set("TxPowerEnd", DoubleValue(powAp));
390.         spectrumPhy.Set("CcaEdThreshold", DoubleValue(ccaEdTrAp));
391.         spectrumPhy.Set("RxSensitivity", DoubleValue(rxSensAp));
392.
393.         mac.SetType("ns3::ApWifiMac", "Ssid", SsidValue(ssid));
394.
395.         apDevices.Add(wifi.Install(spectrumPhy, mac, wifiApNodes.Get(i)));
```

```
396.
397.         //Sets BSS color
398.         if ((technology == 0) && enableObssPd)
399.         {
400.           apDevice = apDevices.Get(i)->GetObject<WifiNetDevice>();
401.           apDevice->GetHeConfiguration()-
    >SetAttribute("BssColor", UintegerValue(i + 1));
402.         }
403.       }
404.
405.       //Set guard interval and MPDU buffer size
406.
407.       if (technology == 0)
408.       { //802.11ax
409.
410.         Config::Set(
411.             "/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/HeConfiguration/MpduBuffe
    rSize",
412.             UintegerValue(useExtendedBlockAck ? 256 : 64));
413.         Config::Set("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/HeConfiguration/G
    uardInterval",
414.                     TimeValue(NanoSeconds(guardInterval)));
415.         ;
416.       }
417.       else
418.       { //802.11n
419.         Config::Set("/NodeList//DeviceList//$ns3::WifiNetDevice/HtConfiguration/"
420.                     "ShortGuardIntervalSupported",
421.                     BooleanValue(guardInterval));
422.       }
423.
424.       /** Mobility Model **/
425.       /*********************************************************************/
426.       MobilityHelper mobility;
427.
428.       int32_t edge_size = (ceil(sqrt(numAp)));
429.       int32_t sta_edge_size = (ceil(sqrt(numSta)));
430.       int32_t counter = 0;
431.       for (int32_t y = 0; (y < edge_size) && (counter < numAp); y++)
432.       {
433.         for (int32_t x = 0; (x < edge_size) && (counter < numAp); x++, counter++)
434.         {
435.           //positionAlloc-
    >Add(Vector((double)x*distance, (double)y*distance, 0.0));
436.           mobility.SetPositionAllocator(
437.               "ns3::GridPositionAllocator", "MinX", DoubleValue((x - 1) * distance)
    , "MinY",
438.               DoubleValue((y - 1) * distance), "DeltaX",
439.               DoubleValue(2 * distance / sta_edge_size), "DeltaY",
440.               DoubleValue(2 * distance / sta_edge_size), "GridWidth",
441.               UintegerValue(sta_edge_size), "LayoutType", StringValue("RowFirst"));
442.
443.           if (walk)
444.           {
445.             mobility.SetMobilityModel(
446.                 "ns3::RandomWalk2dMobilityModel", "Bounds",
447.                 RectangleValue(Rectangle(((x - 1) * distance), ((x + 1) * distance)
    ,
448.                                          ((y - 1) * distance), ((y + 1) * distance)
    )));
449.           }
450.           else
451.           {
452.             mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
453.           }
454.
455.           for (int32_t j = 0; j < numSta; j++)
456.           {
457.             //std::cout << "x:" << x << " y:" << y << " distance:" << distance << "
    \n";
```

```
458.             mobility.Install(wifiStaNodes.Get(counter * numSta + j));
459.           }
460.         }
461.       }
462.
463.       //AP pos
464.       mobility.SetPositionAllocator("ns3::GridPositionAllocator", "MinX", DoubleVal
    ue(0), "MinY",
465.                                     DoubleValue(0), "DeltaX", DoubleValue(distance)
    , "DeltaY",
466.                                     DoubleValue(distance), "GridWidth", UintegerVal
    ue(edge_size),
467.                                     "LayoutType", StringValue("RowFirst"));
468.       mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
469.       mobility.Install(wifiApNodes);
470.       //std::cout << "Mobility model configured\n";
471.
472.       //Routing
473.       InternetStackHelper stack;
474.
475.       Ipv4StaticRoutingHelper staticRoutingHelper;
476.       stack.Install(wifiApNodes);
477.       stack.SetRoutingHelper(staticRoutingHelper);
478.       stack.Install(wifiStaNodes);
479.
480.       Ipv4AddressHelper address;
481.       // address.SetBase("10.1.1.0", "255.255.255.0");
482.
483.       // Ipv4InterfaceContainer csmaInterfaces;
484.       // csmaInterfaces = address.Assign(csmaDevices);
485.       address.SetBase("172.1.0.0", "255.255.0.0");
486.       Ipv4InterfaceContainer apInterfaces;
487.       for (int32_t i = 0; i < numAp; i++)
488.       {
489.         apInterfaces.Add(address.Assign(apDevices.Get(i))); //BS_
490.         address.Assign(staDevices[i]);                      //BS_
491.         address.NewNetwork();
492.       }
493.
494.       for (int32_t i = 0; i < numAp; i++)
495.       {
496.         Ptr<Ipv4StaticRouting> staticRouting;
497.         std::string wifiApIP = "172." + std::to_string(i + 1) + ".0.1";
498.         // std::string csmaApIP = "10.1.1." + std::to_string(i + 1);
499.         for (int32_t j = 0; j < numSta; j++)
500.         {
501.           staticRouting = Ipv4RoutingHelper::GetRouting<Ipv4StaticRouting>(
502.               wifiStaNodes.Get(i * numSta + j)->GetObject<Ipv4>()-
    >GetRoutingProtocol());
503.           staticRouting->SetDefaultRoute(wifiApIP.c_str(), 1);
504.         }
505.         // staticRouting = Ipv4RoutingHelper::GetRouting<Ipv4StaticRouting>(csmaNod
    es.Get(numAp)->GetObject<Ipv4>()->GetRoutingProtocol());
506.         // staticRouting-
    >AddNetworkRouteTo(wifiApIP.c_str(), "255.255.0.0", csmaApIP.c_str(), 1);
507.       }
508.       ApplicationContainer clientApps;
509.       ApplicationContainer serverApps;
510.
511.       if (useUdp)
512.       {
513.         PacketSinkHelper packetSinkHelper("ns3::UdpSocketFactory",
514.                                           InetSocketAddress(Ipv4Address::GetAny(),
    9));
515.         serverApps = packetSinkHelper.Install(wifiApNodes);
516.
517.         //client
518.         for (int32_t i = 0; i < numAp; i++)
519.         {
520.           OnOffHelper onoff("ns3::UdpSocketFactory",
```

```
521.                           Address(InetSocketAddress(apInterfaces.GetAddress(i), 9
    )));
522.           onoff.SetConstantRate(DataRate(dataRate), udpPayloadSize);
523.
524.           for (int32_t j = 0; j < numSta; j++)
525.           {
526.             clientApps = onoff.Install(wifiStaNodes.Get(i * numSta + j));
527.           }
528.         }
529.       }
530.       else
531.       {
532.         PacketSinkHelper packetSinkHelper("ns3::TcpSocketFactory",
533.                                           InetSocketAddress(Ipv4Address::GetAny(),
    5000));
534.         serverApps = packetSinkHelper.Install(wifiApNodes);
535.
536.         //client
537.         for (int32_t i = 0; i < numAp; i++)
538.         {
539.           OnOffHelper onoff("ns3::TcpSocketFactory",
540.                           Address(InetSocketAddress(apInterfaces.GetAddress(i), 5
    000)));
541.           onoff.SetConstantRate(DataRate(dataRate), tcpPayloadSize);
542.
543.           for (int32_t j = 0; j < numSta; j++)
544.           {
545.             clientApps = onoff.Install(wifiStaNodes.Get(i * numSta + j));
546.           }
547.         }
548.       }
549.
550.       serverApps.Start(MilliSeconds(timeStartServerApps));
551.       serverApps.Stop(Seconds(duration + 3));
552.       clientApps.Start(MilliSeconds(timeStartClientApps)); //2.0
553.       clientApps.Stop(Seconds(duration + 2));
554.
555.       /** Energy Model **/
556.       /***************************************************************************/
557.       /* energy source */
558.       LiIonEnergySourceHelper liIonSourceHelper;
559.       // configure energy source
560.       liIonSourceHelper.Set("LiIonEnergySourceInitialEnergyJ", DoubleValue(batteryL
    evel));
561.       // install source
562.       EnergySourceContainer sources = liIonSourceHelper.Install(wifiStaNodes);
563.       /* device energy model */
564.       WifiRadioEnergyModelHelper radioEnergyHelper;
565.       // configure radio energy model
566.       radioEnergyHelper.Set("TxCurrentA", DoubleValue(TxCurrentA));
567.       radioEnergyHelper.Set("RxCurrentA", DoubleValue(RxCurrentA));
568.       radioEnergyHelper.Set("IdleCurrentA", DoubleValue(IdleCurrentA));
569.       radioEnergyHelper.Set("SleepCurrentA", DoubleValue(SleepCurrentA));
570.       radioEnergyHelper.Set("CcaBusyCurrentA", DoubleValue(CcaBusyCurrentA));
571.       radioEnergyHelper.Set("SwitchingCurrentA", DoubleValue(SwitchingCurrentA));
572.
573.       // install device model
574.       DeviceEnergyModelContainer deviceModels = DeviceEnergyModelContainer();
575.
576.       for (int32_t i = 0; i < numAp; i++)
577.       {
578.         for (int32_t j = 0; j < numSta; j++)
579.         {
580.           deviceModels.Add(
581.               radioEnergyHelper.Install(staDevices[i].Get(j), sources.Get(j + i * n
    umSta)));
582.         }
583.       }
584.
585.       /** connect trace sources **/
```

```
586.      /**************************************************************************/
587.      //energy source
588.
589.      if (tracing && verbose)
590.      {
591.        for (int32_t i = 0; i < numAp * numSta; i++)
592.        {
593.          Ptr<LiIonEnergySource> basicSourcePtr =
594.              DynamicCast<LiIonEnergySource>(sources.Get(i));
595.
596.          basicSourcePtr->TraceConnect("RemainingEnergy", std::to_string(i),
597.                                       MakeCallback(&RemainingEnergy));
598.
599.          // device energy model
600.          Ptr<DeviceEnergyModel> basicRadioModelPtr =
601.              basicSourcePtr-
    >FindDeviceEnergyModels("ns3::WifiRadioEnergyModel").Get(0);
602.          NS_ASSERT(basicRadioModelPtr != NULL);
603.
604.          basicRadioModelPtr-
    >TraceConnect("TotalEnergyConsumption", std::to_string(i),
605.                                        MakeCallback(&TotalEnergy));
606.        }
607.      }
608.
609.      //Config::Connect("/NodeList/*/DeviceList/*/Phy/WifiRadioEnergyModel", MakeCa
    llback(&TotalEnergy));
610.      //Config::Connect("/NodeList/*/DeviceList/*/Phy/LiIonEnergySource", MakeCallb
    ack(&RemainingEnergy));
611.      //Config::Connect("/NodeList/*/ApplicationList/*/$ns3::PacketSink/Rx", MakeCa
    llback(&PacketSinkRx));
612.
613.      if (tracing)
614.        Config::Connect("/NodeList/*/DeviceList/*/Phy/MonitorSnifferRx",
615.                        MakeCallback(&MonitorSniffRx));
616.
617.      //Flow monitor logging
618.      /**************************************************************************/
619.      Ptr<FlowMonitor> flowMonitor;
620.      FlowMonitorHelper flowMonHelper;
621.      flowMonitor = flowMonHelper.InstallAll();
622.      //Ipv4GlobalRoutingHelper::PopulateRoutingTables();
623.
624.      Simulator::Stop(Seconds(duration + 3));
625.
626.      if (tracing == true)
627.      {
628.        spectrumPhy.EnablePcap("critical_iot", apDevices);
629.      }
630.
631.      Simulator::Run();
632.
633.      if (tracing)
634.      {
635.        for (int32_t i = 0; i < numAp; i++)
636.        {
637.          double throughput = static_cast<double>(bytesReceived[numSta * numAp + i]
    ) * 8.0 /
638.                              1000.0 / 1000.0 / duration;
639.          std::cout << "Physical throughput for BSS " << i + 1 << ": " << throughpu
    t
640.                    << " Mbit/s" << std::endl;
641.        }
642.      }
643.
644.      std::string outputDir = "";
645.      std::string simTag = "evoMCS" + std::to_string(r);
646.      std::string file = outputDir + "testflow.xml";
647.      flowMonitor->SerializeToXmlFile(file.c_str(), false, true);
648.      // Print per-flow statistics
```

```
649.      flowMonitor->CheckForLostPackets();
650.      Ptr<Ipv4FlowClassifier> classifier =
651.          DynamicCast<Ipv4FlowClassifier>(flowMonHelper.GetClassifier());
652.      FlowMonitor::FlowStatsContainer stats = flowMonitor->GetFlowStats();
653.
654.      std::regex server_regex("^172.*.0.1$");
655.      double averageFlowThroughput = 0.0;
656.      double averageFlowDelay = 0.0;
657.
658.      if (verbose)
659.      {
660.
661.        std::ofstream outFile;
662.
663.        std::string filename = simTag;
664.        outFile.open(filename.c_str(), std::ofstream::out | std::ofstream::trunc);
665.        if (!outFile.is_open())
666.        {
667.          std::cerr << "Can't open file " << filename << std::endl;
668.          return 1;
669.        }
670.
671.        outFile.setf(std::ios_base::fixed);
672.
673.        outFile << "Flow;source;src_port;destiny;dst_port;proto;direction;tx_packet
    s;tx_bytes;tx_"
674.                   "offered_raw;tx_offered_mbps;rx_bytes;rx_throughput_raw;rx_throu
    ghput_mbps;"
675.                   "mean_delay(ms);mean_jitter(ms);rx_packets;lost_packets;packet_l
    oss_ratio \n";
676.        for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.beg
    in();
677.             i != stats.end(); ++i)
678.        {
679.          Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(i->first);
680.          std::stringstream protoStream;
681.          protoStream << (uint16_t)t.protocol;
682.          if (t.protocol == 6)
683.          {
684.            protoStream.str("TCP");
685.          }
686.          if (t.protocol == 17)
687.          {
688.            protoStream.str("UDP");
689.          }
690.          outFile << i->first << ";";
691.          outFile << t.sourceAddress << ":" << t.sourcePort << ";" << t.destination
    Address
692.                  << ":" << t.destinationPort << ";";
693.          outFile << protoStream.str() << ";";
694.
695.          std::stringstream ss;
696.          ss << t.sourceAddress;
697.          if (std::regex_match(ss.str(), server_regex))
698.          {
699.            outFile << "download"
700.                    << ";";
701.          }
702.          else
703.          {
704.            outFile << "upload"
705.                    << ";";
706.          }
707.
708.          outFile << i->second.txPackets << ";";
709.          outFile << i->second.txBytes << ";";
710.          outFile << i->second.txBytes * 8.0 / duration << ";";
711.          outFile << static_cast<double>(i-
    >second.txBytes) * 8 / duration / 1000 / 1000
712.                  << ";";
```

```
713.           outFile << i->second.rxBytes << ";";
714.           if (i->second.rxPackets > 0)
715.           {
716.             // double rxDuration = (timeStartServerApps - timeStartClientApps) / 10
    00.0;
717.             double rxDuration = duration;
718.             averageFlowThroughput += i-
    >second.rxBytes * 8.0 / rxDuration / 1000 / 1000;
719.             averageFlowDelay += 1000 * i->second.delaySum.GetSeconds() / i-
    >second.rxPackets;
720.
721.             outFile << i->second.rxBytes * 8.0 / rxDuration << ";";
722.             outFile << static_cast<double>(i-
    >second.rxBytes) * 8 / rxDuration / 1000 / 1000
723.                     << ";";
724.             outFile << 1000 * i->second.delaySum.GetSeconds() / i-
    >second.rxPackets << ";";
725.             outFile << 1000 * i->second.jitterSum.GetSeconds() / i-
    >second.rxPackets << ";";
726.           }
727.           else
728.           {
729.             outFile << "0;";
730.             outFile << "0;";
731.             outFile << "0;";
732.             outFile << "0;";
733.           }
734.           outFile << i->second.rxPackets << ";";
735.           int lost_packets = (i->second.txPackets - i->second.rxPackets);
736.           outFile << lost_packets << ";";
737.           outFile << (lost_packets * 1.0 / i->second.txPackets) * 100.0 << "\n";
738.         }
739.
740.         outFile.close();
741.         std::ifstream f(filename.c_str());
742.         if (f.is_open())
743.         {
744.           std::cout << f.rdbuf();
745.         }
746.       }
747.
748.       for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin
    ();
749.            i != stats.end(); ++i)
750.       {
751.         std::stringstream ss;
752.         ss << classifier->FindFlow(i->first).sourceAddress;
753.
754.         if (!std::regex_match(ss.str(), server_regex))
755.         {
756.           int lost_packets = (i->second.txPackets - i->second.rxPackets);
757.           avg_packet_loss[r] +=
758.               static_cast<double>(lost_packets) / static_cast<double>(i-
    >second.txPackets);
759.         }
760.       }
761.       avg_packet_loss[r] = avg_packet_loss[r] / static_cast<double>(numSta * numAp)
    ;
762.
763.       if (verbose)
764.       {
765.         std::cout << "Avg packet loss - run " << r << " " << avg_packet_loss[r] <<
    std::endl;
766.       }
767.
768.       //get total energy consumed
769.       for (DeviceEnergyModelContainer::Iterator iter = deviceModels.Begin();
770.            iter != deviceModels.End(); iter++)
771.       {
772.         double energyConsumed = (*iter)->GetTotalEnergyConsumption();
```

```
773.         avg_energy[r] += static_cast<double>(energyConsumed) /
774.                          static_cast<double>(duration * numAp * numSta);
775.
776.         if (verbose)
777.         {
778.           std::cout << static_cast<double>(energyConsumed) /
779.                             static_cast<double>(duration * numAp * numSta)
780.                     << std::endl;
781.         }
782.       }
783.
784.       if (verbose)
785.       {
786.         std::cout << "Avg energy - run " << r << " " << avg_energy[r] << std::endl;
787.       }
788.
789.       //geet throughput
790.       for (int32_t i = 0; i < numAp; i++)
791.       {
792.         double bytesRx =
793.             static_cast<double>(DynamicCast<PacketSink>(serverApps.Get(i))-
    >GetTotalRx());
794.         avg_throughput[r] +=
795.             static_cast<double>(bytesRx) / static_cast<double>(duration * numAp * n
    umSta);
796.
797.         if (verbose)
798.         {
799.           std::cout << static_cast<double>(bytesRx) /
800.                             static_cast<double>(duration * numAp * numSta) * 8.0
801.                     << std::endl;
802.         }
803.       }
804.
805.       if (verbose)
806.       {
807.         std::cout << "Avg throughput - run " << r << " " << avg_throughput[r] * 8.0
     << std::endl;
808.       }
809.
810.       //End Simulator
811.       Simulator::Destroy();
812.     }
813.
814.     double t_energy = 0.0;
815.     double t_bitrate = 0.0;
816.     double t_packet_loss = 0.0;
817.
818.     for (uint32_t r = 0; r < runs; r++)
819.     {
820.       t_energy += avg_energy[r] / static_cast<double>(runs);
821.       t_bitrate += avg_throughput[r] * 8.0 / static_cast<double>(runs);
822.       t_packet_loss += avg_packet_loss[r] / static_cast<double>(runs);
823.     }
824.
825.     std::cout << t_energy << "\t" << t_bitrate << "\t" << t_packet_loss << std::end
    l;
826.     return 0;
827.   }
```

Apêndice B

```
1.  from concurrent.futures import thread
2.  from typing import List
3.  import subprocess
4.  import random
5.  import sys
6.  import copy
7.  import numpy
8.  import concurrent.futures
9.  import pickle
10. from math import sqrt
11.
12. __author__ = 'Miguel Arieiro'
13.
14. directory = "../../tests/"
15.
16. verbose = True
17. seed = 1
18. restart = 0
19. max_threads = None
20.
21. # parameters
22. pop_size = [25, 10, 10]
23. number_generations = 40
24. runs_per_scen = [25, 15, 5]
25. elite_per = 0.3
26. random_per = 0.2
27. minimum_throughput = 0.0
28. mutation_prob = 0.3
29.
30. best_per_gen = list()
31. avg_per_gen = list()
32. diversity = list()
33. avgBest = 0
34. stdBest = 0
35. avgGen = 0
36. stdGen = 0
37. avgDiversity = 0
38. stdDiversity = 0
39. maxFit = 0
40. minFit = 0
41. genMin = 0
42. genMax = 0
43.
44. # {num_cen: [numAp, numSta, duration, dataRate]}
45. scenario = {0: [4, 4, 10, 100000], 1: [
46.     9, 16, 3, 100000], 2: [4, 32, 3, 100000]}
47.
48. # [0 - 802.11ax, 1 - 802.11n]
49. technology = [0, 1]
50.
51. # {technology: [2.4GHz, 5GHz, 6GHz]}
52. frequency = {0: [5, 6], 1: [5]}
53.
54. # {frequency: [20MHz, 40MHz, 80MHz, 160 MHz]}
55. channelWidth = {2: [20, 40], 5: [20, 40, 80, 160], 6: [20, 40, 80, 160]}
56.
57. # {technology: [800ns, 1600ns, 3200ns]}
58. guardInterval = {0: [800, 1600, 3200], 1: [0, 1]}
59.
60. # {technology: [mcs#]}
61. mcs = {0: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11], 1: [0, 1, 2, 3, 4, 5,
62.                                                      6, 7, 8, 9, 10, 11, 12, 13, 1
    4, 15, 16, 17, 18, 19, 20, 21, 22, 23]}
63.
```

```
#param = [technology, frequency, channelWidth, useUDP, useRts, guardInterval, enableObssPd, useExtendedBlockAck, mcs]
param = [technology, frequency, channelWidth, [0, 1],
         [0, 1], guardInterval, [0, 1], [0, 1], mcs]

#indiv = [technology, frequency, channelWidth, useUDP, useRts, guardInterval, enableObssPd, useExtendedBlockAck, mcs, energy, throughput, packet_loss, fitness]

cmd_str = "evoMCS -runs=3 -verbose=0 -tracing=0 -seed=%d -numAp=%d -numSta=%d -duration=%d -dataRate=%d -technology=%d -frequency=%d -channelWidth=%d -useUdp=%d -useRts=%d -guardInterval=%d -enableObssPd=%d -useExtendedBlockAck=%d -mcs=%d"

def gen_indiv():
    global param

    # param[][random.randint(0, len(param[]) - 1)]
    tech = param[0][random.randint(0, len(param[0]) - 1)]
    freq = param[1][tech][random.randint(0, len(param[1][tech]) - 1)]

    indiv = [tech, freq, param[2][freq][random.randint(0, len(param[2][freq]) - 1)], param[3][random.randint(0, len(param[3]) - 1)], param[4][random.randint(0, len(param[4]) - 1)], param[5][tech][random.randint(
        0, len(param[5][tech]) - 1)], param[6][random.randint(0, len(param[6]) - 1)], param[7][random.randint(0, len(param[7]) - 1)], param[8][tech][random.randint(0, len(param[8][tech]) - 1)], -1, -1, -1, -1]
    return indiv

def heuristic_v1(energy, throughput, packetLoss=0.0, minimum_throughput=0.0):
    if (throughput <= minimum_throughput):
        return sys.maxsize
    return (energy/throughput)

def heuristic_v2(energy, throughput, packetLoss=0.0, minimum_throughput=0.0):
    if (throughput <= minimum_throughput):
        return sys.maxsize
    return (energy/throughput*(1.0+packetLoss))

def run_indiv(indiv, scen):
    global minimum_throughput
    global cmd_str
    command = cmd_str % tuple([1] + scen + indiv[0:-4])
    result = subprocess.run(
        ['./waf', '--run-no-build', command], capture_output=True, text=True)

    try:
        res = result.stdout.splitlines()[1].split()
        energy = float(res[0])
        throughput = float(res[1])
        packet_loss = float(res[2])
    except:
        command = cmd_str % tuple([3] + scen + indiv[0:-4])
        result = subprocess.run(
            ['./waf', '--run-no-build', command], capture_output=True, text=True)
        try:
            res = result.stdout.splitlines()[1].split()
            energy = float(res[0])
            throughput = float(res[1])
            packet_loss = float(res[2])
        except:
            print(
                "*******************************Errror************************************:\n"+command)
            print(result)
            energy = sys.maxsize
            throughput = - 1.0
            packet_loss = 1.0

    indiv[-4] = energy
    indiv[-3] = throughput
    indiv[-2] = packet_loss
```

```
125.
126.      if verbose:
127.          print("energy: " + str(energy) + "\tthroughput: " +
128.                str(throughput) + "\tpacket_loss: " + str(packet_loss))
129.
130.      return heuristic_v2(energy, throughput, packet_loss, minimum_throughput)
131.
132.  def run_all(population, current_scen, all=False):
133.
134.      global max_threads
135.
136.      res = list()
137.      if all == True:
138.          with concurrent.futures.ThreadPoolExecutor(max_workers=max_threads) as ex
    ecutor:
139.              for i in range(len(population)):
140.                  f = executor.submit(run_indiv, population[i], current_scen)
141.                  res.append(f)
142.          for i in range(len(population)):
143.              population[i][-1] = res[i].result()
144.
145.      else:
146.          count = 0
147.          with concurrent.futures.ThreadPoolExecutor(max_workers=max_threads) as ex
    ecutor:
148.              for i in range(len(population)):
149.                  if (population[i][-1] == (-1)):
150.                      f = executor.submit(run_indiv, population[i], current_scen)
151.                      res.append(f)
152.
153.          for i in range(len(population)):
154.              if (population[i][-1] == (-1)):
155.                  population[i][-1] = res[count].result()
156.                  count += 1
157.
158.      return population
159.
160.  def gen_population(pop_size, current_scen):
161.      population = []
162.      for _ in range(pop_size):
163.          indiv = gen_indiv()
164.          population.append(indiv)
165.
166.      population = run_all(population, current_scen)
167.      return population
168.
169.  def gen_new_population(parents, offspring, current_scen):
170.      global elite_per
171.      global random_per
172.      size = len(parents)
173.      comp_elite = int(size * elite_per)
174.      comp_random = int(size * random_per)
175.      offspring.sort(key=lambda x: x[-1])
176.      parents.sort(key=lambda x: x[-1])
177.      new_population = copy.deepcopy(parents[:comp_elite]) + offspring[:size -
178.                                                                      comp_elite -
     comp_random] + gen_population(comp_random, current_scen)
179.      return new_population
180.
181.  def mutate_prob(original_indiv):
182.      global param
183.      global mutation_prob
184.      #indiv = [technology, frequency, channelWidth, useUDP, useRts, guardInterval,
     enableObssPd, useExtendedBlockAck, mcs, energy, throughput, packet_loss, fitness]
185.      indiv = original_indiv
186.      indiv[-4:] = [-1, -1, -1, -1]
187.
188.      for i in range(len(indiv) - 5):
189.          if random.random() < mutation_prob:
190.              # case tech=1 == 802.11n, and there's only 5GHz
```

```
191.             if ((i == 1) and (len(param[1][indiv[0]]) == 1)):
192.                 continue
193.
194.             # frequency and channel width
195.             if (i == 1) or (i == 2):
196.                 temp = indiv[i]
197.                 while (temp == indiv[i]):
198.                     temp = param[i][indiv[i-1]
199.                                    ][random.randint(0, len(param[i][indiv[i-
    1]]) - 1)]
200.                 indiv[i] = temp
201.
202.             # guardInterval and mcs
203.             elif (i == 5) or (i == 8):
204.                 temp = indiv[i]
205.                 while (temp == indiv[i]):
206.                     temp = param[i][indiv[0]][random.randint(
207.                         0, len(param[i][indiv[0]]) - 1)]
208.                 indiv[i] = temp
209.
210.             # other
211.             else:
212.                 temp = indiv[i]
213.                 while (temp == indiv[i]):
214.                     temp = param[i][random.randint(0, len(param[i]) - 1)]
215.                 indiv[i] = temp
216.
217.             # dependent parameters
218.             if indiv[1] not in param[1][indiv[0]]:
219.                 indiv[1] = param[1][indiv[0]][random.randint(
220.                     0, len(param[1][indiv[0]]) - 1)]
221.             if indiv[2] not in param[2][indiv[1]]:
222.                 indiv[2] = param[2][indiv[1]][random.randint(
223.                     0, len(param[2][indiv[1]]) - 1)]
224.             if indiv[5] not in param[5][indiv[0]]:
225.                 indiv[5] = param[5][indiv[0]][random.randint(
226.                     0, len(param[5][indiv[0]]) - 1)]
227.             if indiv[8] not in param[8][indiv[0]]:
228.                 indiv[8] = param[8][indiv[0]][random.randint(
229.                     0, len(param[8][indiv[0]]) - 1)]
230.
231.     return indiv
232.
233.  def mutate_one(original_indiv):
234.     global param
235.     #indiv = [technology, frequency, channelWidth, useUDP, useRts, guardInterval,
    enableObssPd, useExtendedBlockAck, mcs, energy, throughput, packet_loss, fitness]
236.     indiv = original_indiv
237.     indiv[-4:] = [-1, -1, -1, -1]
238.
239.     i = random.randint(0, len(indiv) - 5)
240.
241.     # case tech=1 == 802.11n, and there's only 5GHz
242.     while ((i == 1) and (len(param[1][indiv[0]]) == 1)):
243.         i = random.randint(0, len(indiv) - 5)
244.
245.     # frequency and channel width
246.     if (i == 1) or (i == 2):
247.         temp = indiv[i]
248.         while (temp == indiv[i]):
249.             temp = param[i][indiv[i-1]
250.                            ][random.randint(0, len(param[i][indiv[i-1]]) - 1)]
251.         indiv[i] = temp
252.
253.     # guardInterval and mcs
254.     elif (i == 5) or (i == 8):
255.         temp = indiv[i]
256.         while (temp == indiv[i]):
257.             temp = param[i][indiv[0]][random.randint(
258.                 0, len(param[i][indiv[0]]) - 1)]
```

```
259.          indiv[i] = temp
260.
261.      # other
262.      else:
263.          temp = indiv[i]
264.          while (temp == indiv[i]):
265.              temp = param[i][random.randint(0, len(param[i]) - 1)]
266.          indiv[i] = temp
267.
268.      # dependent parameters
269.      if indiv[1] not in param[1][indiv[0]]:
270.          indiv[1] = param[1][indiv[0]][random.randint(
271.              0, len(param[1][indiv[0]]) - 1)]
272.      if indiv[2] not in param[2][indiv[1]]:
273.          indiv[2] = param[2][indiv[1]][random.randint(
274.              0, len(param[2][indiv[1]]) - 1)]
275.      if indiv[5] not in param[5][indiv[0]]:
276.          indiv[5] = param[5][indiv[0]][random.randint(
277.              0, len(param[5][indiv[0]]) - 1)]
278.      if indiv[8] not in param[8][indiv[0]]:
279.          indiv[8] = param[8][indiv[0]][random.randint(
280.              0, len(param[8][indiv[0]]) - 1)]
281.
282.      return indiv
283.
284.  def rank_pop(population, n):
285.      pop = copy.deepcopy(population)
286.      pop.sort(reverse=True, key=lambda x: x[-1])
287.      probs = [(2*i)/(len(pop)*(len(pop) + 1)) for i in range(1, len(pop) + 1)]
288.      parents = []
289.      for _ in range(n):
290.          value = random.uniform(0, 1)
291.          index = 0
292.          total = probs[index]
293.          while total < value:
294.              index += 1
295.              total += probs[index]
296.          parents.append(pop[index])
297.      return parents
298.
299.  def mutate_all(population, mutation_op=mutate_one):
300.      global random_per
301.      global elite_per
302.      offspring = []
303.
304.      size = len(population)
305.      n = size-int(size * elite_per)-int(size * random_per)
306.
307.      for i in rank_pop(population, n):
308.          offspring.append(mutation_op(i))
309.
310.      return offspring
311.
312.  def hamming_distance(v1, v2):
313.      # heuristic = v[-1]
314.      return sum([1 for i in range(len(v1)-4) if v1[i] != v2[i]])
315.
316.  def euclidean_distance(v1, v2):
317.      pairs = list(zip(v1[:-4], v2[:-4]))
318.      # heuristic = v[-1]
319.      return sqrt(sum([(pair[0] - pair[1])**2 for pair in pairs]))
320.
321.  def reset_stats():
322.      global best_per_gen
323.      global avg_per_gen
324.      global diversity
325.      global avgBest
326.      global stdBest
327.      global avgGen
328.      global stdGen
```

```
329.      global maxFit
330.      global minFit
331.      global genMin
332.      global genMax
333.      global avgDiversity
334.      global stdDiversity
335.
336.      best_per_gen = list()
337.      avg_per_gen = list()
338.      diversity = list()
339.      avgBest = 0
340.      stdBest = 0
341.      avgGen = 0
342.      stdGen = 0
343.      avgDiversity = 0
344.      stdDiversity = 0
345.      maxFit = 0
346.      minFit = 0
347.      genMin = 0
348.      genMax = 0
349.
350.  def update_stats(population, metric=hamming_distance):
351.
352.      global best_per_gen
353.      global avg_per_gen
354.      global diversity
355.      p_size = len(population)
356.
357.      population.sort(key=lambda x: x[-1])
358.      best_per_gen.append(population[0][-1])
359.      avg_per_gen.append(sum([f[-1] for f in population]) / p_size)
360.
361.      count = 0
362.      for i in range(p_size):
363.          for j in range(i+1, p_size):
364.              distance = metric(population[i], population[j])
365.              if distance != 0:
366.                  count += 1
367.
368.      div = (2.0*count)/(p_size*(p_size-1))
369.      diversity.append(div)
370.
371.  def calculate_stats():
372.
373.      global best_per_gen
374.      global avg_per_gen
375.      global diversity
376.      global avgBest
377.      global stdBest
378.      global avgGen
379.      global stdGen
380.      global maxFit
381.      global minFit
382.      global genMin
383.      global genMax
384.      global avgDiversity
385.      global stdDiversity
386.
387.      avgBest = numpy.mean(best_per_gen)
388.      stdBest = numpy.std(best_per_gen)
389.      avgGen = numpy.mean(avg_per_gen)
390.      stdGen = numpy.std(avg_per_gen)
391.      maxFit = max(best_per_gen)
392.      minFit = min(best_per_gen)
393.      genMax = best_per_gen.index(maxFit)
394.      genMin = best_per_gen.index(minFit)
395.      avgDiversity = numpy.mean(diversity)
396.      stdDiversity = numpy.std(diversity)
397.
398.  def stats_string():
```

```
399.      global best_per_gen
400.      global avg_per_gen
401.      global diversity
402.      global avgBest
403.      global stdBest
404.      global avgGen
405.      global stdGen
406.      global maxFit
407.      global minFit
408.      global genMin
409.      global genMax
410.
411.      string = '\nMédia: {}'.format(str(avg_per_gen))
412.      string += '\nDiversidade: {}'.format(str(diversity))
413.      string += '\nMáximo do melhor: {} -> geração: {}/{}'.format(
414.          maxFit, genMax, number_generations)
415.      string += '\nMínimo do melhor: {} -> geração: {}/{}'.format(
416.          minFit, genMin, number_generations)
417.      string += '\nMédia do melhor: {} +- {}'.format(avgBest, stdBest)
418.      string += '\nMédia geracional: {} +- {}'.format(avgGen, stdGen)
419.      string += '\nDiversidade média: {} +- {}'.format(
420.          avgDiversity, stdDiversity)
421.      return string
422.
423.  def log_file(path, string):
424.      with open(path, "a+") as file:
425.          file.write(string+'\n')
426.
427.  def save_random_state(run):
428.      global directory
429.      file_path = directory+'random_state_'+str(run)
430.      with open(file_path, 'wb') as random_state_file:
431.          pickle.dump(random.getstate(), random_state_file)
432.
433.  def load_random_state(run):
434.      global directory
435.      file_path = directory+'random_state_'+str(run)
436.      with open(file_path, 'rb') as random_state_file:
437.          random_state = pickle.load(random_state_file)
438.          random.setstate(random_state)
439.
440.  def main():
441.      global pop_size
442.      global number_generations
443.      global elite_per
444.      global random_per
445.      global minimum_throughput
446.      global scenario
447.      global runs_per_scen
448.      global seed
449.      global restart
450.      global directory
451.      global max_threads
452.      metric = hamming_distance
453.      mutation_op = mutate_one
454.
455.      filename = "%d_%d_%d_%.2f_%.2f_%s_%.2f.log" % (
456.          pop_size[0], number_generations, runs_per_scen[0], elite_per, random_per,
      mutation_op.__name__, mutation_prob)
457.      file_path = directory + filename
458.      reset_stats()
459.
460.      start = 0
461.
462.      if restart:
463.          # set values
464.          count = 25
465.          num_scen = 1
466.          current_scen = scenario[num_scen]
467.          start = 25
```

```
468.          load_random_state(start-1)
469.          # set last pop
470.          population = []
471.      else:
472.          random.seed(seed)
473.          count = 0
474.          num_scen = 0
475.          current_scen = scenario[num_scen]
476.          log_file(
477.              file_path, "[technology, frequency, channelWidth, useUDP, useRts, gua
    rdInterval, enableObssPd, useExtendedBlockAck, mcs]")
478.          # gen original population
479.          population = gen_population(pop_size[0], current_scen)
480.
481.      update_stats(population, metric)
482.
483.      if verbose:
484.          print(population)
485.      log_file(file_path, str(population))
486.
487.      for run in range(start, number_generations):
488.          if (count == runs_per_scen[num_scen]):
489.              count = 0
490.              num_scen += 1
491.              current_scen = scenario[num_scen]
492.              max_threads = 1
493.              population = population[:pop_size[num_scen]]
494.
495.              calculate_stats()
496.
497.              if verbose:
498.                  print(stats_string())
499.              log_file(file_path, stats_string())
500.
501.              print("Scenario: %d" % (num_scen))
502.              log_file(file_path, str(num_scen))
503.
504.              reset_stats()
505.              run_all(population, current_scen, all=True)
506.
507.          if verbose:
508.              print("Run: %d\t Scenario: %d" % (run, num_scen))
509.
510.          offspring = mutate_all(population, mutation_op)
511.          run_all(offspring, current_scen)
512.          population = gen_new_population(population, offspring, current_scen)
513.          run_all(population, current_scen)
514.
515.          update_stats(population, metric)
516.
517.          if verbose:
518.              print(population)
519.          log_file(file_path, str(population))
520.
521.          save_random_state(run)
522.
523.          count += 1
524.
525.      calculate_stats()
526.
527.      if verbose:
528.          print(stats_string())
529.
530.      log_file(file_path, stats_string())
531.
532.  if __name__ == "__main__":
533.      main()
```

Apêndice C

Artigo

EvoMCS: Optimising Energy and Throughput of Mission Critical Services in dense environments

Miguel Arieiro
DEI, UC
Coimbra, Portugal
marieiro@student.dei.uc.pt

Bruno Sousa
CISUC, DEI, UC
Coimbra, Portugal
bmsousa@dei.uc.pt

ABSTRACT

Mission critical services have stringent requirements in terms of reliability, energy-efficiency and performance. In this regard, optimisation approaches are required to maximise missions' service time by minimising the consumption of energy without compromising performance like the throughput in scenarios with high density in terms of connected devices. EvoMCS is an evolutionary algorithm approach able to determine the optimal configurations for IEEE 802.11 wireless networks in dense environments with multiple objectives such as minimisation of consumed energy and maximisation of the throughput in each device. EvoMCS is evaluated in wildfire scenarios, determining the optimal values for configuration parameters in 802.11n and 802.11ax technologies. The achieved results demonstrate that EvoMCS is able to provide optimal configuration values, that reduces in a factor of three the energy that is consumed by devices.

CCS CONCEPTS

• **General and reference** → Evaluation; • **Networks** → **Network performance modeling**; • **Theory of computation** → **Evolutionary algorithms**.

KEYWORDS

mission critical services, IEEE 802.11ax, energy efficiency, evolutionary algorithm

ACM Reference Format:
Miguel Arieiro and Bruno Sousa. 2021. EvoMCS: Optimising Energy and Throughput of Mission Critical Services in dense environments. In *MSWiM '21: ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, November 22–26 2021, Alicante, ES.* ACM, New York, NY, USA, 8 pages.

1 INTRODUCTION

Mission Critical Services (MCS) have stringent Quality of Service (QoS) requirements, in terms of resilience and delay-sensitivity since they can be associated with people's lives. Wildfires, in Mediterranean countries like Portugal, have a high occurrence probability in summer, requiring strategical plans at national level [1] to coordinate human and technical resources for an efficient fire combat. Public Mobile Radio (PMR) technologies, like TETRA are outdated and do not fulfil the required capabilities of first responders [21], for instance no support for video or IoT data communications. In this regard, technologies like LTE and 5G [4] have included several feature to enhance the support of MCS (e.g. PROSE), within the same security levels of PMR technologies [12].

The wildfires of Pedrogão Grande, Portugal [9] lead to damages in communication infrastructures, which had an impact in the operations of the first responders. Thus, there is the need for solutions to support fast deployment that can be easily managed and that scale/adapt according to the evolution of the wildfire combating scenario, for instance, new teams, which require communication facilities, arriving at the scene.

In this regard, the evolution in IEEE 802.11ax standard, also known as WiFi 6, brings additional features for simplified deployments with a high number of end-user devices and more efficient power saving mechanisms [13]. IEEE 802.11ax enhances throughput, through the adoption of OFDMA at MAC and PHY layers. In addition, the standard also introduces the Basis Service Set (BSS) colouring scheme, which allows to distinct inter- and intra-BSS frames based on their preamble, thus decreasing collision and enabling efficient channel access and power saving mechanisms.

With high availability concerns for communication facilities, research [8] defends the coexistence of multiple technologies, such as 5G and IEEE 802.11ax, exploring spectrum sharing features, and using WiFi as cognitive radio with OFDMA approaches.

The research on MCS have been focusing LTE [20] and its evolution (5G and beyond) and Wireless Mesh Networks [16]. The efforts mainly consider the performance of services, or functionalities like Device to device communications and do not entail scenarios without a supporting infrastructure, or integrated with the vehicles providing support for first responders in wildfires combat.

EvoMCS, here in proposed is a multi-objective optimization approach that provides optimal values for configuration parameters in IEEE 802.11n and IEEE 802.11ax networks with dense deployments in terms of simultaneous connected users. The achieved results, demonstrated that EvoMCS is able to reduce in a factor of three the energy consumed by devices, when compared with sub-optimal configuration profiles.

1.1 EvoMCS contributions

Our research in MCS has validated Service Function Chaining policies for critical services [3], with WiFi and LTE as the wireless technologies connecting first responders to support vehicles. Support vehicles (e.g., ambulance, fire combat vehicle) include computational

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MSWiM '21, November 22–26, 2021, Alicante, ES
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00

resources and can support heterogeneous wireless technologies to enable critical services. Despite the validation of policies to decompose services into functions, and chain them to enable higher levels of security and distribution of resources, the assessment and validation of optimised configuration for end-users connected through WiFi 6 has not been researched.

EvoMCS, herein proposed, provides the following contributions:

(1) Design, validation of a multiple-objective Genetic Algorithm to optimise energy-efficiency and throughput in MCS with dense environments (i.e., with high number of connected users per km^2.
(2) Validate, through simulation, the feasibility of employing WiFi 6, as a supporting wireless technology to enable communications between teams in the field.
(3) Identification of optimised configuration sets that can be employed as profiles to extend the battery lifetime of end-user devices or to optimise performance of critical applications.
(4) The implementation of EvoMCS is released as an Open Source github repository[1].

The remaining of this paper is organised as follows: Section 2 compares EvoMCS with the related work, while Section 3 details the algorithm of EvoMCS, which is evaluated according to the experimentation methodology described in Section 4. Achieved results demonstrating the EvoMCS contributions are presented in Section 5.3, while Section 6 concludes the paper.

2 RELATED WORK

Mission Critical Services have been subject of research in several areas. Sabtah et al. [20] propose a parameter selection framework for LTE to enable Mission Critical Networks. The proposed framework tackles the problem of finding the best set of parameters for MCS, considering aspects like eNB deployment, allocated spectrum, achieved throughput and latency. Despite considering aspects at the network and device level, the evaluations conducted via simulation does not consider energy consumption. Contrasting with the work of Sotirios et al. [10], which propose a model to maximise the coverage and minimise the power consumption in LTE networks. The devised model relies on Differential Evolution algorithms and consider real data from the city of Ghent, regarding the location of base stations.

MCS can benefit from wireless mesh networks (WMN), or from multiple wireless technologies. Murugeswari et al. [16] propose a model for routing in WMN relying on generic algorithms with the aim to minimise the transmission count and the respective delay. The results, achieved through simulation, demonstrate a gain in the throughput and that transmission delay is minimised within a variable number of nodes.

Niyato et al. [17] use evolutionary game approaches to optimise the network selection problem in the presence of multiple wireless technologies (IEEE 802.11, WiMAX, etc). The achieved results demonstrate that evolutionary population is able to reach equilibrium faster in comparison to reinforcement learning, but requires centralised components to gather information of users and services.

DeepWiFi [6] introduces deep neural networks to adapt the WiFi spectrum dynamically in order to provide throughput gains, and to enhance the resilience support against jamming attacks. Despite the promising results, the proposed is mainly tailored to the IEEE 802.11ac standard.

Hangjung et al. [24] have researched the issue of composition mission critical applications into tasks and processes, considering functional requirements in order to enhance end-to-end reliability, security, performance and cost of applications. Given such conflicting objectives, a multiple criteria genetic algorithm is proposed to explore the QoS criteria and identify the Pareto optimal multidimensional frontier to trade off conflicting objectives. Despite the interesting approach, the performance objective only considers processing times, communication times and delays.

EvoMCS employs evolutionary algorithms that have been used in the literature to optimize solutions for diverse problems. Soumaya et al. [23] propose a multi-objective evolutionary algorithm to improve network planning by enhancing the location accuracy of IEEE 802.11 Access Points. In the same line, Alfredo [18] employs a multi-objective evolutionary algorithm to place nodes in wireless sensor networks, tackling the nodes placement and the total energy dissipated with the placement.

As per the analysed works, distinct optimisation approaches have been considered to tackle energy efficiency, service performance, devices locations, among others. EvoMCS enables a multiple-objective evolutionary algorithm aiming to optimise energy efficiency and throughput in mission critical scenarios with stringent requirements and dense environments with high number of connected users in the same area.

3 EVOMCS: MULTI-OBJECTIVE OPTIMIZATION

This section provides the design and the implementation details of EvoMCS as a multiple-objective optimisation approach.

3.1 Scenario and technologies

The EvoMCS is tailored to IEEE 802.11ax and IEEE 802.11n technologies, which can be employed in wildfire scenarios, considering legal and physical aspects (i.e. water pressure levels) where the distance

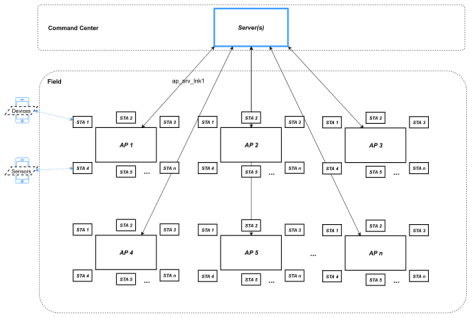


Figure 1: Evaluation scenario

[1] https://github.com/MiguelArieiro/EvoMCS

of firemen to the supporting vehicles cannot exceed 100m [15]. In addition, EvoMCS consider that such technologies are deployed in a fast fashion and can be easily integrated with existing resources (i.e. vehicles and respective functionalities, like fire combat, water tanks), as illustrated in Figure. 1.

3.2 Evolutionary Algorithm

The reasoning to chose the parameters summarised in Table 1 for the individuals in the population of genetic algorithm in EvoMCS is related with their impact in the performance of 802.11 wireless technologies in dense environments [7, 13].

Table 1: Information of individuals of the population

Tech	Freq. (GHz)	Channel Band.	moCS	GI	RTS	Ext. Block	OBSS	UDP/ TCP
.11ax	{5,6}	{20, 40, 80, 160}	[0,11]	{800, 1600, 3200}	{0,1}	{0,1}	{0,1}	{0,1}
.11n	{5}	{20, 40, 80, 160}	[0,23]	{400, 800}	{0,1}	n/a	n/a	{0,1}

EvoMCS considers the values of channel bandwidth in MHz, the Modulation Coding Scheme (moCS) as per the possible values in each technology. The Guard Interval (GI), reported in nanoseconds (ns), corresponds to the parameter that defines the wait time between symbols being transmitted. Higher values (3200) of GI lead to higher delays in the spread propagation, thus with better resilience, while lower values (800) lead to better throughput values but with higher levels of interference, which is impacts devices in dense environments. The Request to Send (RTS) parameter aims to enhance transferring performance solving the hidden terminal problem [7]. This issue occurs when a station is transmitting but the another one in the same service set is not aware of such transmission. RTS allows to inform stations in the same service set regarding the transmission of others, thus reducing the collision probability [13]. The Extended Block feature allows to perform multiple acknowledgements in a single frame, enabling bandwidth savings.

The type of application also impacts the energy-efficiency in wireless scenarios. EvoMCS considers applications sending data (i.e., UL-uplink direction) using protocols employed in IoT context, like MQTT (TCP-based) and CoAP (UDP-based) [11]. The reasoning to choose UL data applications is related with the enhancements introduces with WiFi 6 (11.ax) regarding UL transmissions. The utilisation of TCP, with reliability mechanisms can lead to higher levels of consumption of energy, since a give unit of data needs to be acknowledge, or re-transmitted multiple times, if loss is verified.

Figure 2 depicts the diverse steps of the EvoMCS, relying on a Genetic algorithm approach. The first step corresponds to the generation of the initial population, in a random fashion, and considering the possible values for genomes, as per the parameter's values in Table 1. The second step includes the evaluation of the population through simulation, that is assessing the energy consumed and the achieved throughput in a specific scenario. After the simulation, the individuals, corresponding to possible configurations are ranked in step 3, considering the devised heuristic. Each generation includes the best individuals of the original population and the descendants and individuals generated randomly (strangers). The number of

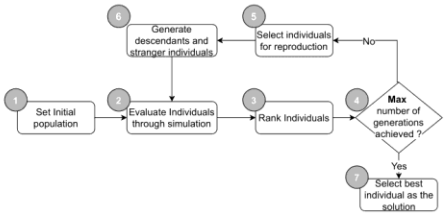


Figure 2: Evolutionary algorithm

maximum generations serves as a limit to stop the finding process of best possible configurations, after n generations. At the end, in step 7, the best solution are selected.

3.3 Heuristic for fitness

Individuals are evaluated according to the $H(E/T)$- heuristic that corresponds to the fitness function in the evolutionary algorithm, and employed in step 2 and step 3 of EvoMCS, as depicted in Figure 2.

The average energy/throughput ratio metric $H(E/T)$ allows to assess how well a specific individual performs in terms of energy efficiency and throughput in a period of time. In other words, the $H(E/T)$ heuristic assesses the required average energy to send a bit of information (Joules/bit) in a time period. This heuristic allows to rank as best individuals, those that present lower values, that is the ones that are able to send more information with less energy.

Listing 1: Heuristic $H(E/T)$ - Energy/throughput ratio

```
1 def heuristic (energy, throughput, min_thr=0.0):
2     if (throughput < min_thr):
3         return sys.maxsize
4     return (energy/throughput)
```

$H(E/T)$ assumes that an individual has data to send over a period of time, which will be sent according to a certain throughput, thus throughput is higher than zero, as illustrated in the Python code Listing 1. Indeed, if an individual does not reach a minimum throughput level (line 2) it is penalised with the maximum possible value, which is set to *sys.maxsize*, as per line 3.

3.4 Selection Strategy

The selection strategy in EvoMCS is performed in step 5, when the maximum number of generations, which is assessed in step 4, has not been achieved.

The implementation of the selection strategy is depicted in listing 2, where individuals are sorted in reverse order (line 3) as per the $H(E/T)$ fitness value. Each individual receives a probability in order to be selected, as per lines 4-5. The probability value considers the position that an individual occupies in the population list, as the best values appear in the first positions, due to the sorting per the heuristic value. Higher values of the fitness correspond to the worst individuals, those that spent more energy to send a specific amount of data, as demonstrated with the reverse order in line 3.

Listing 2: Selection of individuals

```
1  def rank_pop (population, n):
2      pop = copy.deepcopy(population)
3      pop.sort(reverse=True, key=lambda x: x[-1])
4      probs = [(2*i)/(pop_size*(pop_size + 1))
5          for i in range (1, pop_size + 1)]
6      parents = []
7      for _ in range (n):
8          value = random.uniform(0,1)
9          index = 0
10         total = probs [index]
11         while total < value:
12             index += 1
13             total += probs[index]
14         parents.append(pop[index])
15     return parents
```

After setting the probability values, the *n* individuals are selected based on a random distribution, as per lines 7-14. The *n*-number of individuals that are selected can be configured, since this is a parameter in the *rank_pop()* function.

Step 6 of EvoMCS uses the input of the *n* individuals selected by the *rank_pop()* function and combines the selected with stranger individuals. Such combination is required to avoid having non-dominated solutions and to avoid the local minimum issues of genertic algorithms [19]. Indeed, the introduction of stranger individuals acts as a prevention for premature convergence to a local optima. The stranger individuals are randomly generated, as per the logic of the initial population in step 1.

3.5 Operators to generate descendants

The operators to generate descendants in step 6, rely on different approaches: The Single Gene Mutation Operator (SGMO), and the Multi-Gene Probabilistic Mutation Operator (MGMO).

The SGMO only affects a single gene, which is selected randomly, considering the number of genes that compose an individual. After the random selection of a gene, it is verified the corresponding parameter field, for instance if it correlates to the MCS parameter. As per the type of field the random values that can be assigned to the gene, must consider the possible values presented in Table 1, within the specific technology set.

Listing 3: Example code for MGMO

```
1  def mutate_prob(original_indiv):
2    global param
3    global mutation_prob
4    indiv = original_indiv
5    indiv [-1] = -1
6    for i in range(len(indiv) - 1):
7      if random.random() < mutation_prob:
8        if (i == 1) or (i == 2):
9          temp=indiv[i]
10         while (temp == indiv[i]):
11               temp = param[i][indiv[i-1]]
12               [random.randint(0,
13               len(param[i][indiv[i-1]]) - 1)]
14           indiv[i] = temp
15   # partial listing for demo purposes
16   return indiv
```

The MGMO can modify multiple genes according to a probability value, as per the logic depicted in listing 3. MGMO determines a random probability for a gene and if the generated value is below a pre-configured probability that specific gene is modified, as per line 8. The modification follows the logic of the SGMO for that particular gene. Lines 10-16 illustrate the logic to modify the values of the frequency and channel width parameters.

4 EXPERIMENTATION

This section depicts the experimentation methodology to assess the performance of the EvoMCS is finding the optimal configurations, and to validate design choices, such as single-SGMO or multiple-MGMO.

4.1 Validation scenarios

Diverse scenarios have been devised, as summarised in Table 2. The Basic scenario was employed to validate the workflow of running the EvoMCS algorithm implemented in Python and interacting with ns-3 to validate the individuals configuration. This particular workflow involved mainly steps 2 and step 3.

Table 2: Validations scenarios

Scenario	#Access Points	#Stations per AP	Duration	Bit Rate
Basic	4	4	10s	100 Kbps
Intermediate	9	16	10s	100 Kbps

The Intermediate scenario, is more complex and leads to a total of 144 stations, and is close to the deployments that can be observed in real wildfire combating scenarios [1]. This is the scenario that is employed to evaluate the performance of the optimal profiles determined by EvoMCS.

4.2 Configuration parameters

Diverse configuration parameters were considered in the experimentation, as documented in this subsection. Such parameters are associated with the energy model of WiFi stations and access points (assumed to be on support vehicles). The WiFi stations follow the *LiIonEnergySource* model, which states the capacity of the battery in mAh and the respective voltage. The technical characteristics of the Samsung Note 10 have been considered for the energy model, in particular with 4300mAh and 3.85Volts. The access points are assumed to operate without energy constraints, they are connected to the power source of the vehicles[2].

The radio energy follows the WiFi model that includes how much energy is spent in the different states. For instance, how much is spent in the transmission (*TxCurrentA*), in reception (*RxCurrentA*), sleep mode (*SleepCurrentA*), and the amount of energy spent in the state transition (*SwitchingCurrentA*). The values of such parameters have been considered as per Vitor et al. [2] and Yang et al.[22] for the IEEE 802.11 technologies.

No mobility has been considered in the scenarios evaluating the EvoMCS to avoid having values affected by the location variation of

[2]Such power source relies normally on diesel generators.

stations. For this, the *RandomWalk2dMobilityModel* was configured with no velocity. Stations and access points are placed in a grid layout, with the access point in the centre. Stations are put at a distance to the access points around 50 meters.

The mobility is set to uniform values in the range of $[2, 4]m/s$ in the profiles evaluation, where the aim is to assess the gains with the optimal configurations determined by EvoMCS.

Other relevant configuration parameters include the number of antennas in stations and access points that is set to 4, which with 4 spatial streams for tx-transmission and rx-reception. The threshold for OBSS is set to -82dBm with the *ns3::ConstantObssPdAlgorithm*.

The configuration parameters of the Genetic algorithm of EvoMCS are summarised in Table 3

Table 3: GA Configuration parameters in EvoMCS

Parameter	Value(s)	Description
popSize	25	Population Size
nGeneration	15	16 Generations with the initial population as generation '0'
muteOperat	{SGMO, MGMO}	Mutation Operator
muteProb	50%	Mutation probability of a single gene in MGMO

The population size is set 25, as per the recommendations in the literature, regarding the higher performance values when employing populations with low size [5].

4.3 Evaluation Metrics

The evaluation metrics employed to evaluate an individual are summarised in Table 4. These metrics are used to determine the $H(E/T)$ heuristic.

Table 4: Evaluation Metrics used in fitness $H(E/T)$

Metric	Description	How is measured
(E) conEne	Average energy consumed by all the stations in Joules	$E = \frac{\sum conEne_i}{nSta}$
(T) goodPut	Throughput measured at application layer in bits/s (discards other layers)	$T = \frac{\sum throughput_i}{nSta}$

The evaluation metrics of the Genetic algorithm of EvoMCS are summarised in Table 5.

Table 5: GA evaluation metrics in EvoMCS

Metric	Description	How is measured
bestMax	Maximum of Best	$max(H(E/T))$
bestMin	Minimum of Best	$min(H(E/T))$
avBest	Average of Best	$[\sum H_b(E/T)]/nBest$
$avFit_x$	Average of Fitness in generation x	$[\sum H_i(E/T)]/nGeneration$
avDiver	Average Diversity	$\frac{2.0*nCountDiff}{popSize*(popSize-1)}$

The *nBest*- number of best individuals corresponds to percentage of elitism that is configured, in this case 30%. The remaining individuals are the descendants of the population, in a ratio of 50%, while 20% are configured for the stranger individuals. The *avBest*-metric is determined the *b*- best individuals (in the ratio of 30%).

The $nCountDiff$ corresponds to the count of individuals that are different in a population. The difference is determined according to the Hamming distance, as commonly employed in genetic algorithms [14].

4.4 Profiles validation - inputs from EvoMCS

The output of EvoMCS is then evaluated in the intermediary scenario (recall Table 2) with mobility activated in the WiFi Stations. The mobility considers velocities varying uniformly in the $[2, 4]m/s$ range. This scenario is also validated according to the Packet Error Ratio (%), the mean delay of a flow (ms), the direction of the flow, download and upload and the receiving throughput measure in kbit/s.

The performance of EvoMCS is also assessed, considering the difference between the profiles in terms of the required average energy to send a bit of information (Joules/bit).

5 RESULTS

This sections presents and discusses the achieved results.

5.1 Operators for the EvoMCS

The choice of the mutation operators are evaluated considering the $avFit_x$average of the fitness value in the diverse generations of a population, and the *avDiver*- average diversity.

Figure. 3 depicts the average fitness in the diverse generations for the SGMO and MGMO with a probability of 50%.

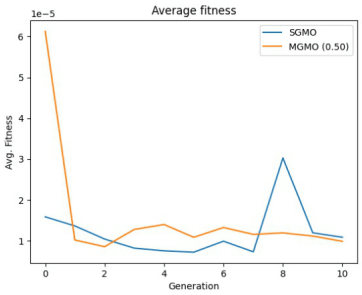


Figure 3: Average fitness per generation

The SGMO has the lowest average value for fitness, since the MGMO starts initially with the worst value of fitness. As pictured the variation of the fitness value is higher in the MGMO approach, as opposed to the SGMO, which has average fitness values varying in the range of $7.36x10^{-06}$ and $3.03x10^{-05}$.

Figure 4 depicts the diversity metric, demonstrating a better performance with the MGMO (higher values are more interesting). MGMO is able to introduce more diversity due to the high probability value of 50% to mutate genes in individuals. Despite the lower values of diversity for SGMO the difference is minimal.

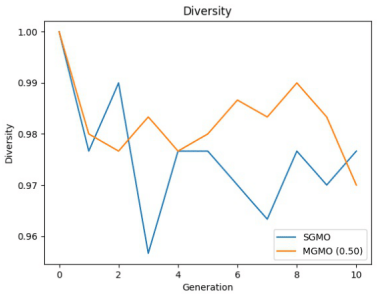


Figure 4: Average diversity per generation

The SGMO can provide better results when integrated with the introduction of stranger individuals. Thus SGMO was the approach that was employed regarding the operator to mutate individuals in the populations of EvoMCS.

5.2 Optimal configurations

On the experiments performed, EvoMCS was able to identify the configurations in Table 6 as the best and worst configurations/profiles to be employed by stations in MCS scenarios.

Table 6: Information of individuals of the population

Profile	Tech	Freq.	Channel Band.	moCS	GI	RTS	Ext. Block	OBSS	App
Best	.11ax	6	20	3	800	0	0	0	TCP
Worst	.11n	5	160	23	1600	1	n/a	n/a	TCP

The values summarised in Table 6 corresponds to the overall configuration that is assessed as optimal, according to $H(E/T)$ heuristics, and the one that provides the worst configuration.

The best configuration/profile stands out as being the IEEE 802.11ax with TCP applications in the 6GHz Frequency. The lowest value of the Guard Interval set to 800ns for better values of throughput. The 16-QAM is proposed as the modulation coding scheme, which can provide a theoretical data rate of 36 MBit/s, with the 20MHz bandwidth channel.

TCP applications, due to the reliability mechanisms (e.g. acknowledgement, re-transmissions, etc) is known to introduce more impact on the energy. Nonetheless, EvoMCS discards the employment of UDP due to lower values in the *goodPut* metric (recall Table 4). UDP does not provide reliability mechanisms, which lead to higher packet losses.

5.3 Optimal profiles in scenarios with dense-environments

The results of packet loss are demonstrated in Figure 5 for both directions of traffic for the bast and worst profiles identified by EvoMCS.

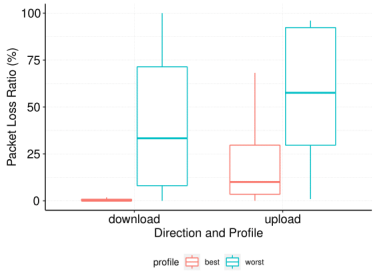
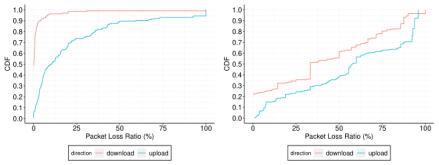


Figure 5: Packet Loss Ratio (best and worst)

The download traffic is sent from the servers to the stations, for instance considering the acknowledgements messages in the reliability mechanisms of TCP. The difference in the achieved performance (average of all the stations) is better with the best profile, being more noticeable in the downlink direction. The upload traffic



(a) Best **(b) Worst**

Figure 6: CDF for Packet Loss Ratio

with the best profile has losses below 25%. Indeed, as demonstrated with the Cumulative Distribution values (CDF) of the packet loss ratio in Figure 6a more than 75% of nodes have packet losses below 25% in the upload direction. Which, clearly contrasts with the worst profile, where 75% of the nodes have packet losses close to 75% levels, as pictured in Figure 6b.

As illustrated, in Figure 6 the download performance is also affected, where 80% of the nodes have losses below 75%, while in the best profile, more than 80% have losses below 15%.

The mean delay per station is illustrated in Figure 7 for the best and worst configurations. The best profile is able to provide lower values of delay, the y-scale varies between 0 and 1500, as opposed to the range of 0 and 6000 in the worst configuration. In addition to the y-scale difference, the delay varies more between the different stations, the area in the boxplots is higher, in the

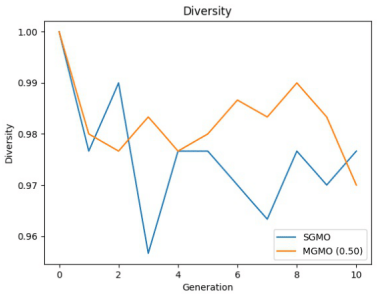


Figure 4: Average diversity per generation

The SGMO can provide better results when integrated with the introduction of stranger individuals. Thus SGMO was the approach that was employed regarding the operator to mutate individuals in the populations of EvoMCS.

5.2 Optimal configurations

On the experiments performed, EvoMCS was able to identify the configurations in Table 6 as the best and worst configurations/profiles to be employed by stations in MCS scenarios.

Table 6: Information of individuals of the population

Profile	Tech	Freq.	Channel Band.	moCS	GI	RTS	Ext. Block	OBSS	App
Best	.11ax	6	20	3	800	0	0	0	TCP
Worst	.11n	5	160	23	1600	1	n/a	n/a	TCP

The values summarised in Table 6 corresponds to the overall configuration that is assessed as optimal, according to $H(E/T)$ heuristics, and the one that provides the worst configuration.

The best configuration/profile stands out as being the IEEE 802.11ax with TCP applications in the 6GHz Frequency. The lowest value of the Guard Interval set to 800ns for better values of throughput. The 16-QAM is proposed as the modulation coding scheme, which can provide a theoretical data rate of 36 MBit/s, with the 20MHz bandwidth channel.

TCP applications, due to the reliability mechanisms (e.g. acknowledgement, re-transmissions, etc) is known to introduce more impact on the energy. Nonetheless, EvoMCS discards the employment of UDP due to lower values in the *goodPut* metric (recall Table 4). UDP does not provide reliability mechanisms, which lead to higher packet losses.

5.3 Optimal profiles in scenarios with dense-environments

The results of packet loss are demonstrated in Figure 5 for both directions of traffic for the bast and worst profiles identified by EvoMCS.

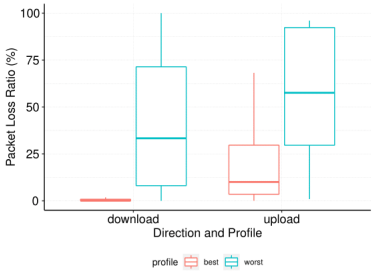
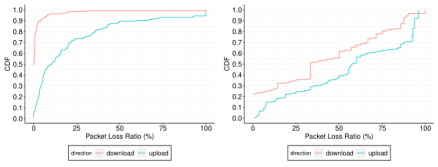


Figure 5: Packet Loss Ratio (best and worst)

The download traffic is sent from the servers to the stations, for instance considering the acknowledgements messages in the reliability mechanisms of TCP. The difference in the achieved performance (average of all the stations) is better with the best profile, being more noticeable in the downlink direction. The upload traffic



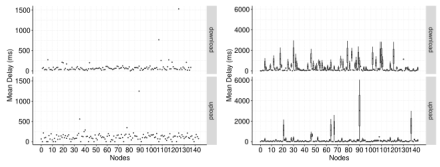
(a) Best **(b) Worst**

Figure 6: CDF for Packet Loss Ratio

with the best profile has losses below 25%. Indeed, as demonstrated with the Cumulative Distribution values (CDF) of the packet loss ratio in Figure 6a more than 75% of nodes have packet losses below 25% in the upload direction. Which, clearly contrasts with the worst profile, where 75% of the nodes have packet losses close to 75% levels, as pictured in Figure 6b.

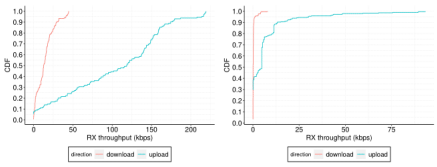
As illustrated, in Figure 6 the download performance is also affected, where 80% of the nodes have losses below 75%, while in the best profile, more than 80% have losses below 15%.

The mean delay per station is illustrated in Figure 7 for the best and worst configurations. The best profile is able to provide lower values of delay, the y-scale varies between 0 and 1500, as opposed to the range of 0 and 6000 in the worst configuration. In addition to the y-scale difference, the delay varies more between the different stations, the area in the boxplots is higher, in the



(a) Best (b) Worst

Figure 7: Mean Delay variation per Station (ms)



(a) Best (b) Worst

Figure 9: CDF for RX Throughput

download and upload directions. One can observe such evidence with the station/node with the id 90.

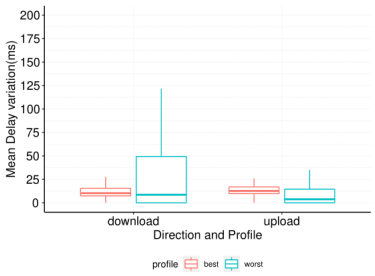


Figure 8: Delay variation (in ms)

The optimal profile is also able to introduce more stabilisation in the mean delay. This trend is also depicted in the delay variation in Figure 8. The standard error deviations are not pictured, but are lower when compared with the worst configuration. Also, as illustrated in Figure 7a the area in the boxplots is reduced a point, since all the quartiles have the same value.

Regarding the delay variation, it should be noticed that the values tend to be higher with the best profile. This is related with the higher volumes of data that is transferred with such configuration. The worst configuration, on the other hand with less volume of data successfully transferred, is characterised with high variations regarding jitter. Such variation is more evident in the download traffic with download direction, as depicted in Figure 8

The metric of RX throughput measures the amount of data that is received per unit of time. The simulation was configured constant data rates of 1000 kbps. Figure 9 depicts the CDF values for the RX throughput in kbps, where the difference relies in the x-scale with the best profile providing the data rates close to 200 kbps. Indeed, almost 50% of the nodes have a throughput higher than 150 kbps, as illustrated in Figure 9a.

The RX throughput with the worst configuration leads to unacceptable values, where more than 90% of the nodes have a throughput bellow 25 kbps. Considering the higher packet loss ratios, the worst configurations, with the IEEE 802.11n lead to performance values that do not meet the required SLAs for critical mission services.

The consumed energy plays a key roles in mission critical scenarios, in particular wildfires scenarios, since these missions last days. Figure 10 depicts the ratio of the consumed energy per a bit of data. As demonstrated, the performance of the best profile leads to lower values in the consumed energy per each bit that is received and transmitted. The worst configuration is three times worse than the best, which means that the battery of the device will last three times less.

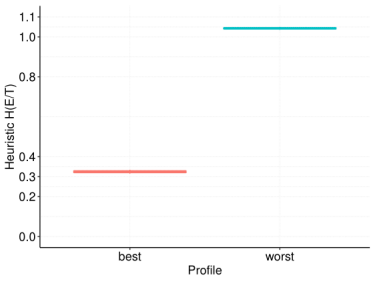


Figure 10: Heuristic with ratio of consumed energy per bit

The achieved results put in evidence the performance achieved with the best profile and the worst. The best, relying on the IEEE 802.11ax, is arguably 3x better then the worst. The characteristics of IEEE 802.11ax lead to more stable results (recall mean delay and the average delay variation results).

6 CONCLUSIONS

EvoMCS is a multi-objective approach relying on a evolutionary approach that is able to determine optimal values for multiple-objectives, that include the maximisation of throughput and minimisation of consumed energy.

Achieved results demonstrate that EvoMCS is able to determine optimal configurations for IEEE 802.11ax and IEEE 802.11n, that improve the ratio of energy consumed and throughput in a order three. Indeed, the difference between the best profile is three times better when compared with the worst profile configuration.

Our next steps include the optimisation for devices with heterogeneous technologies, integrating 5G and IEEE 802.11 support and heuristics that perform the dynamic selection of best profiles, according to the performance observed in real-time.

ACKNOWLEDGMENTS

This work was supported in part by the European Regional Development Fund (FEDER), through the Regional Operational Programme of Lisbon (POR LISBOA 2020); in part by the Competitiveness and Internationalization Operational Programme (COMPETE 2020) of the Portugal 2020 framework [Project OREOS with Nr. 049029 (POCI-01-0247-FEDER-049029)]; in part by the national funds through the FCT - Foundation for Science and Technology, I.P., within the scope of the project CISUC - UID/CEC/00326/2020; and in part by the European Social Fund, through the Regional Operational Program Centro 2020.

REFERENCES

[1] ANPC. 2020. Diretiva Operacional Nacional n.2 - DECIR - Dispositivo Especial de Combate a Incêndios Rurais.

[2] Vitor Manuel Henriques Bernardo. 2015. *Energy Efficient Multimedia Communications in IEEE 802.11 Networks*. Ph.D. Dissertation. http://hdl.handle.net/10316/28311

[3] Bruno Miguel Sousa and Henrique Simões Silva and Noé Godinho and Marilia Curado. 2021. Service Function Chaining in Wildfire Scenarios. In *2021 2nd International Workshop on Network Softwarization Techniques for IoT Applications (SoftIoT 2021)*.

[4] CEPT ECC. 2015. Draft ECC Report 218: "Harmonised conditions and spectrum bands for the implementation of future European Broadband Public Protection and Disaster Relief (BB-PPDR) systems".

[5] Chen, Tianshi and Tang, Ke and Chen, Guoliang and Yao, Xin. 2012. A large population size can be unhelpful in evolutionary algorithms. *Theoretical Computer Science* 436 (2012), 54–70. arXiv:1208.2345

[6] Kemal Davaslioglu, Sohraab Soltani, Tugba Erpek, and Yalin E. Sagduyu. 2019. DeepWiFi: Cognitive WiFi with deep learning. *arXiv* 1233, c (2019), 1–15. arXiv:1910.13315

[7] Der Jiunn Deng, Kwang Cheng Chen, and Rung Shiang Cheng. 2014. IEEE 802.11ax: Next generation wireless local area networks. In *Proceedings of the 2014 10th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, QSHINE 2014*, Vol. 1. 77–82.

[8] Der Jiunn Deng, Shao Yu Lien, Jorden Lee, and Kwang Cheng Chen. 2016. On Quality-of-Service Provisioning in IEEE 802.11ax WLANs. *IEEE Access* 4 (2016), 6086–6104.

[9] Domingos Xavier Viegas et al. 2017. O Complexo de incêndios de Pedrogão Grande e Concelhos Limítrofes, Iniciado a 17 de Junho de 2017.

[10] Sotirios K. Goudos, Margot Deruyck, David Plets, Luc Martens, and Wout Joseph. 2017. Optimization of Power Consumption in 4G LTE Networks Using a Novel Barebones Self-adaptive Differential Evolution Algorithm. *Telecommunication Systems* 66, 1 (sep 2017), 109–120.

[11] Cenk Gündoğan, Peter Kietzmann, Martine Lenders, Hauke Petersen, Thomas C. Schmidt, and Matthias Wählisch. 2018. NDN, COAP, and MQTT: A comparative measurement study in the IoT. *ICN 2018 - Proceedings of the 5th ACM Conference on Information-Centric Networking* (2018), 159–171. arXiv:1806.01444

[12] Kaliyammal Prabhu et al. 2020. A Reliability-Aware, Delay Guaranteed, and Resource Efficient Placement of Service Function Chains in Softwarized 5G Networks. *IEEE Trans. on Cloud Comp.* (2020).

[13] Evgeny Khorov, Anton Kiryanov, Andrey Lyakhov, and Giuseppe Bianchi. 2019. A tutorial on IEEE 802.11ax high efficiency WLANs. *IEEE Communications Surveys and Tutorials* 21, 1 (2019), 197–216.

[14] Yong-Hyuk Kim and Byung-Ro Moon. 2004. Distance Measures in Genetic Algorithms. In *Genetic and Evolutionary Computation – GECCO 2004*, Kalyanmoy Deb (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 400–401.

[15] Ministério da Administração Interna - Autoridade Nacional de Emergência e Proteção Civil. 2016. Despacho n.º 7316/2016.

[16] R. Murugeswari, S. Radhakrishnan, and D. Devaraj. 2016. A multi-objective evolutionary algorithm based QoS routing in wireless mesh networks. *Applied Soft Computing* 40 (mar 2016), 517–525.

[17] Dusit Niyato and Ekram Hossain. 2009. Dynamics of Network Selection in Heterogeneous Wireless Networks: An Evolutionary Game Approach. *IEEE Transactions on Vehicular Technology* 58, 4 (may 2009), 2008–2017.

[18] Alfredo J. Perez. 2018. M-SPOT: A Hybrid Multiobjective Evolutionary Algorithm for Node Placement in Wireless Sensor Networks. In *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, Vol. 2018-Janua. IEEE, 264–269.

[19] Miguel Rocha and José Neves. 1999. Preventing premature convergence to local optima in genetic algorithms via random offspring generation. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 127–136.

[20] Ayman Sabbah, Abdallah Jarwan, Larry Bonin, and Mohamed Ibnkahla. 2019. A High-Level Parameter Selection Framework for Irregular LTE-Based Mission Critical Networks. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, Vol. 2019-April. IEEE, 1–6.

[21] International Forum to Advance First Responders Forum. 2021. Common Capability Gaps. https://www.internationalresponderforum.org/capability_gaps.

[22] Hang Yang, Der-Jiunn Deng, and Kwang-Cheng Chen. 2018. On energy saving in IEEE 802.11 ax. *IEEE Access* 6 (2018), 47546–47556.

[23] Soumaya Zirari. 2012. IEEE 802.11 Network Planning based on ESBEA Evolutionary Algorithm to Improve Location Accuracy. November (2012), 13–15.

[24] Hangjung Zo, Derek L. Nazareth, and Hemant K. Jain. 2019. Service-oriented Application Composition with Evolutionary Heuristics and Multiple Criteria. *ACM Transactions on Management Information Systems* 10, 3 (nov 2019), 1–28.

Email feedback

From: MSWiM'21
Sent: 28 de agosto de 2021 16:44
To: Miguel Arieiro; Bruno Miguel Sousa
Cc: Noura Aljeri; Azzedine Boukerche; Rodolfo W. L. Coutinho; Carlo Giannelli; Jun Zheng
Subject: [MSWiM'21] Your paper #1570736535 ('EvoMCS: Optimising Energy andThroughput of Mission Critical Services in dense environments')

Dear Dr. Miguel Arieiro:

We regret to inform you that your paper #1570736535 ('EvoMCS: Optimising Energy and Throughput of Mission Critical Services in dense environments') cannot be accepted for publication in the ACM Proceedings of the 24th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems.

This year, we have received a high number of submissions and several high-quality papers did not find space due to limitations in the number of accepted papers that can fit in the conference program. The acceptance rate for this year is 19.5%.

We regret that your paper cannot be accepted, and we thank you for considering MSWIM as the outlet for your work.

The reviews are below or can be found at https://edas.info/showPaper.php?m=1570736535, using your EDAS login name marieiro@student.dei.uc.pt.

Best regards,

Carlo Giannelli and Jun Zheng
TPC Co-Chairs for ACM MSWiM 2021

======= Review 1 =======

> *** Summary: Please comment briefly on the following: What are the major contributions of the paper? Do you consider them important? Comment on the degree of relevance, novelty, technical depth and presentation quality of the paper. What are the strengths and weaknesses of the paper?

Authors propose EvoMCS, a multi-objective approach relying on a evolutionary approach that is able to determine optimal values for multiple objectives such as the maximisation of throughput and minimisation
of consumed energy.

> *** Originality and Impact: Assess the originality of the work and its contribution to the area of research
Has merit but mostly incremental (3)

> *** Presentation: Assess the quality of the presentation in terms of English, organization and completeness

Some work needed (2)

> *** Technical Correctness: Assess the technical correctness of the work
Flaws but Easy to Correct (3)

> *** Relevance: How relevant is the paper to MSWiM?
Somewhat Relevant (3)

> *** Reviewer Familiarity: Please assess your familiarity with the subject matter of the paper
Familiar with this area of research (3)

> *** Recommendation: What is your overall recommendation for the paper?
Accept if room (Top 30% but not top 20%) (3)

> *** Detailed Comments: Please provide detailed comments and suggestions to the authors for improving the paper.

Authors propose EvoMCS, a multi-objective approach relying on a evolutionary approach that is able to determine optimal values for multiple objectives such as the maximisation of throughput and minimisation
of consumed energy. It is not clear in the GA applied to optimize the metrics how the convergence can be affected by the parameters tuning. Please, explain better the GA steps such as fitness function selection, chromosome definition for the specific problem, mutation probability generation and cross-over operations.

======= Review 2 =======

> *** Summary: Please comment briefly on the following: What are the major contributions of the paper? Do you consider them important? Comment on the degree of relevance, novelty, technical depth and presentation quality of the paper. What are the strengths and weaknesses of the paper?

This paper presents an evolutionary algorithm to determine the optimal configurations for IEEE 802.11 wireless networks in dense environments with multiple objectives such as minimization of consumed energy and maximization of the throughput in each device.

This is an important problem, but the current version of this work could have more details/discussions about the proposed solution.

> *** Originality and Impact: Assess the originality of the work and its contribution to the area of research
Has merit but mostly incremental (3)

> *** Presentation: Assess the quality of the presentation in terms of English, organization and completeness
Good (4)

> *** Technical Correctness: Assess the technical correctness of the work

Several Flaws (1)

> *** Relevance: How relevant is the paper to MSWiM?
Definitely Relevant (5)

> *** Reviewer Familiarity: Please assess your familiarity with the subject matter of the paper
Familiar with this area of research (3)

> *** Recommendation: What is your overall recommendation for the paper?
Likely reject (Top 50%, but not top 30%) (2)

> *** Detailed Comments: Please provide detailed comments and suggestions to the authors for improving the paper.

This paper presents an evolutionary algorithm to determine the optimal configurations for IEEE 802.11 wireless networks in dense environments with multiple objectives such as minimization of consumed energy and maximization of the throughput in each device.

In Section 2, it's not clear how the proposed mechanism advances the state of art.

In Section 3.1, it's said that the proposed mechanism can be employed in wildfire scenarios. Only this kind of scenario? Please, describe the class of applications suitable for your proposal.

What are the main advantages of using an evolutionary algorithm for this problem? Any comments on the convergence to find the best parameters?

At the introduction, you mention dense environments, but the network sizes in Table 2 are relatively small and not dense. Did you consider larger and denser scenarios?

======= Review 3 =======

> *** Summary: Please comment briefly on the following: What are the major contributions of the paper? Do you consider them important? Comment on the degree of relevance, novelty, technical depth and presentation quality of the paper. What are the strengths and weaknesses of the paper?

The paper is readable. But the contributions are not considerable. It is primarily engineering the genetic algorithm model for determining parameters of wireless communication.

> *** Originality and Impact: Assess the originality of the work and its contribution to the area of research
Marginal contribution (2)

> *** Presentation: Assess the quality of the presentation in terms of English, organization and completeness
Readable (3)

> *** Technical Correctness: Assess the technical correctness of the work

Flaws but Easy to Correct (3)

> *** Relevance: How relevant is the paper to MSWiM?
Somewhat Relevant (3)

> *** Reviewer Familiarity: Please assess your familiarity with the subject matter of the paper
Working in this area of research (5)

> *** Recommendation: What is your overall recommendation for the paper?
Likely reject (Top 50%, but not top 30%) (2)

> *** Detailed Comments: Please provide detailed comments and suggestions to the authors for improving the paper.

This work highlights the need for solutions to support fast deployment that can be easily managed and that scale/adapt according to the evolution of the wildfire combating scenario, for instance, new teams, which require communication facilities, arriving at the scene.

The paper requires English improvement. There are several typos and grammatical errors.

You do not need a subsection in the Introduction section.

The research gap has not been clearly explained in the introduction section.

The authors claim that “The EvoMCS is tailored to IEEE 802.11ax and IEEE 802.11n technologies.” Why it cannot work with 802.11ac?

Some figures are not legible. Larger font size must be used.
The proposed work uses genetic algorithms to find the best configuration parameters in a given scenario.
The work, however, does not specify what is the novelty of the proposed solution. What is new in addition to engineering the GA for the application at hand with 802.11ax?

A very important parameter that is missing is the algorithm execution duration and its adaptability to network dynamics.