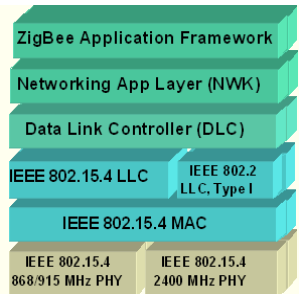
UNIVERSIDADE DA BEIRA INTERIOR
Departamento de Engenharia Electromecânica



Desenvolvimento, Simulação e Validação de Protocolos MAC e de Encaminhamento para Redes de Sensores sem Fios

Dissertação do Mestrado em Engenharia Electromecânica

Jorge Miguel da Silva Tavares

Covilhã, Agosto de 2009

À minha família.

Agradecimentos

Ao concluir este projecto quero agradecer a todos aqueles que contribuíram para o nascimento e evolução do mesmo. Agradeço desde já ao Prof. Fernando José Velez por ter aceitado orientar este projecto disponibilizando equipamentos, conhecimento científico, tempo, ideias, sugestões e criticas necessárias para o avanço deste.

Gratifico o Eng. Luís Borges pela ajuda na correcta instalação do Castalia e do OMNeT++ no Ubunto e VMWare Worksation, assim como por todas as dúvidas, ajuda na escolha de simuladores, partilha de conhecimentos e apoio prestadas.

Agradeço ao meu colega Norberto Barroca pela ajuda na resolução de erros de compilação, apoio na visualização de novas abordagens dos protocolos MAC, ajuda na escolha dos simuladores e também pelo apoio e convívio prestado.

Felicito o Eng. João Ferro pela cooperação e disponibilidade para ajudar na escolha do simulador assim como na correcta percepção dos parâmetros deste e por todo o apoio prestado.

Ao Eng. Daniel Robalo pela ajuda nas correcções da dissertação, disponibilidade, e apoio prestado.

Agradeço ao Mestre Orlando Cabral a ajuda na escolha do simulador, resolução de problemas em ambiente Linux, disponibilidade, convívio e apoio prestado.

Aos restantes membros do We-move@Covilhã assim como todos os que estiveram presentes no local de realização deste projecto, agradeço pela disponibilidade, convívio e apoio prestado.

Ao Eng. Pedro Martins pelas facilidades no horário laboral que foram possíveis facultar, e que permitiram um maior tempo investido nesta dissertação, prejudicando muitas vezes compromissos laborais já por si carenciados de tempo.

Resumo

A optimização de protocolos com elevada eficiência energética em redes de sensores sem fios constitui um desafio e a sua aplicação em contextos de automação de processos e recolha de dados apresentou-se como uma grande motivação. Simularam-se aplicações capazes de encaminhar os dados pela rede até ao nó *Sink* para vários protocolos com diferentes parâmetros ajustáveis, tais como o tempo de escuta; dutycycle e o período de contenção.

Consideraram-se sensores MICAz da Crossbow, a funcionar a 2,4GHz (IEEE 802.15.4) e suportados pelo sistema operativo TinyOS para instalação prática da aplicação nos nós sensores, ou suportada pelo simulador Castalia/OMNet++. Como cenário para as aplicações sem mobilidade, considera-se uma área para plantação onde os nós sensores a controlar a humidade do solo. Para as aplicações com mobilidade, considera-se que os nós sensores estão a monitorizar a localização de cada animal duma manada.

Verificou-se, através de simulação em Castalia, que quanto maior for o período de escuta maior será o consumo de energia. Nas aplicações com mobilidade, o protocolo S-MAC é o que apresenta uma eficiência energética superior. Nas aplicações sem mobilidade, num cenário com 36 nós dispostos numa grelha rectangular, conclui-se que o protocolo T-MAC tem um desempenho energético ligeiramente superior.

Caso se esteja a utilizar uma RSSF, numa aplicação prática em que todos os nós têm uma função crítica, deve-se escolher um valor para o período de contenção correspondente ao valor mais reduzido do desvio padrão da energia consumida. Caso a função de um nó sensor não seja crítica para o sistema, o valor óptimo para o período de contenção é dado pelo valor mínimo da média de consumo de energia da rede. No cenário considerado, os valores óptimos para o período de contenção são 40 e 50 ms, para os casos onde missão individual de cada nó é "crítica" ou "não crítica", respectivamente.

Palavras-chave

Redes de Sensores Sem Fios, *TinyOS*, OMNeT++, Castalia, Protocolos MAC, MICAz, IEEE 802.15.4.

Abstract

The optimization of highly energy efficient protocols for wireless sensors networks is a challenge and its application in contexts of process automation and data collection was faced as an endeavour. Applications able to route data through a network (towards the Sink node) were simulated for different adjustable parameters, such as the listening time; the dutycycle and the contention period.

MICAz sensors from Crossbow operating at 2.4 GHz (IEEE 802.15.4) and supported by TinyOS were assumed both for the practical installation of the sensor nodes application and for the Castalia/OMNet++ simulator. As a scenario for applications without mobility, an agriculture planting area was considered, whose soil humidity is monitored by the sensors nodes. In turn, for applications with mobility, one assumes that the sensors nodes are monitoring the location of each animal in a herd.

It was found through simulation in Castalia that the longer the listening period is the higher the energy consumption is. In applications with mobility, the S-MAC the protocol is the one with higher energy efficiency. However, for applications without mobility, in a scenario with 36 nodes deployed on a rectangular grid, the simulations showed that T-MAC energy efficiency is slightly better.

In a practical application in which all the WSN nodes have a “critical” function, one should choose a value for the contention period corresponding to the lowest value for the standard deviation of the energy spent. If the function of a sensor node is not “critical” for the system, the optimum value for the contention period is achieved for the minimum average energy spent. In the considered scenario, the values of the contention period to be chosen are 40 and 50 ms, for the cases where the individual node mission is “critical” or “non-critical”, respectively.

Keywords

Wireless Sensor Networks, *TinyOS*, OMNeT++, Castalia, MAC protocols, MICAz, IEEE 802.15.4.

Índice

AGRADECIMENTOS ..... **I**
RESUMO ..... **III**
PALAVRAS-CHAVE ..... **III**
ABSTRACT ..... **V**
KEYWORDS ..... **V**
ÍNDICE ..... **VII**
LISTA DE FIGURAS ..... **IX**
LISTA DE TABELAS ..... **XI**
LISTA DE SIGLAS ..... **XIII**
LISTA DE SÍMBOLOS ..... **XV**
LISTA DE PROGRAMAS ..... **XVII**
CAPÍTULO 1 INTRODUÇÃO **1**
1.1 INTRODUÇÃO 1
1.2 MOTIVAÇÃO 1
1.3 DESAFIOS E ABORDAGEM 3
1.4 ESTRUTURA DO TRABALHO 4
CAPÍTULO 2 CARACTERÍSTICAS E APLICAÇÕES **7**
2.1 INTRODUÇÃO 7
2.2 ALGUNS PROJECTOS EM CURSO 7
2.3 TINYOS 12
2.3.1 História do TinyOS *12*
2.3.2 Descrição do TinyOS *13*
2.3.3 Objectivos do TinyOS *14*
2.3.4 Eventos, Comandos e Tarefas *15*
2.3.5 O Projecto do Kernel TinyOS *18*
2.4 NESC 19
2.5 APLICAÇÕES 22
2.6 EFEITOS BIOLÓGICOS DA RADIAÇÃO ELECTROMAGNÉTICA 23
2.6.1 Mecanismos de interacção dos campos electromagnéticos *24*
2.6.2 Efeitos não térmicos *27*
2.7 SUMÁRIO E CONCLUSÕES 29
CAPÍTULO 3 IEEE 802.15.4 ..... **31**
3.1 INTRODUÇÃO 31
3.2 NORMA 32
3.2.1 Acesso ao meio *32*
3.2.2 Tipos de Dispositivos *33*
3.3 ESTRUTURA DAS TRAMAS 33
3.3.1 Data Frame Format *33*
3.3.2 Acknowledgement Frame Format *34*
3.3.3 MAC Command Frame Format *34*
3.3.4 Beacon Frame Format *35*
3.3.5 Superframe Format *35*
3.4 PROTOCOLOS DA CAMADA DE LIGAÇÃO 36
3.4.1 Protocolo Sensor – Medium Access Control *36*
3.4.2 Protocolo Adaptive Rate Control *38*
3.4.3 Protocolo Time-out-Medium Access Control *38*
3.4.4 Protocolo Backoff-Medium Access Control *40*

3.4.5 Protocolo Distributed Energy-Medium Access Control *41*
3.4.6 Protocolo Traffic adaptive Multiple Access *42*
3.5 ZIG BEE ALLIANCE 43
3.6 CARACTERÍSTICAS E CONSUMO DE ENERGIA 44
3.6.1 Definição da Norma *44*
3.6.2 Características Básicas da Rede *44*
3.6.3 Baterias *44*
3.7 FIABILIDADE E ROBUSTEZ 45
3.7.1 Fiabilidade *45*
3.7.2 Robustez *45*
3.8 ZIGBEE VS BLUETHOOTH 46
3.8.1 Tecnologias concorrentes ou que se complementam? *46*
3.8.2 Diferenças em relação à interface com o ar *46*
3.9 SUMÁRIO E CONCLUSÕES 47

CAPÍTULO 4 DESENVOLVIMENTO 49

4.1 INTRODUÇÃO 49
4.2 MULTIHOP 49
4.3 PERL 51
4.3.1 Características da linguagem *51*
4.3.2 Comando make *53*
4.4 HARDWARE 56
4.4.1 Módulo processador/rádio *56*
4.4.2 Placa de programação *57*
4.5 SIMULADOR 58
4.5.1 Introdução *58*
4.5.2 Selecção do simulador *58*
4.5.3 Castalia *63*
4.5.3.1 Diagrama de transição de estados do módulo MAC 64
4.5.3.2 Apresentação dos Parâmetros a Alterar 71
4.6 SUMÁRIO E CONCLUSÕES 77

CAPÍTULO 5 SIMULAÇÕES 79

5.1 INTRODUÇÃO 79
5.2 CENÁRIOS 79
5.3 RESULTADOS 82
5.4 SUMÁRIO E CONCLUSÕES 93

CAPÍTULO 6 CONCLUSÕES E TRABALHO FUTURO 95

6.1 CONCLUSÕES 95
6.2 PROPOSTA PARA TRABALHO FUTURO 96

A. ANEXO – MICROCONTROLADOR ATMEGA 128L 97

B. ANEXO – MENSAGENS DO CASTALIA 99

C. ANEXO –TABELA DE CONSUMO DE ENERGIA 111

D. ANEXO –TABELA DE CONSUMO DE ENERGIA 113

E. ANEXO – TABELA DE CONSUMO DE ENERGIA 115

F. ANEXO – GRÁFICOS DE CONSUMO DE ENERGIA 117

G. ANEXO – TABELA DE CONSUMO DE ENERGIA 121

H. ANEXO –TABELA DE CONSUMO DE ENERGIA 123

I. ANEXO –.CASTALIA-PRIMARY-OUTPUT.TXT 125

J. ANEXO – OMNETPP.INI 133

REFERÊNCIAS 137

Lista de Figuras

FIGURA 2.1 – UMA APLICAÇÃO PODE SER VISTA COMO UM GRUPO DE COMPONENTES [5]. 15
FIGURA 2.2 – GESTOR DE EVENTOS [5]. 17
FIGURA 2.3 – COMPONENTE [5]. 18
FIGURA 2.4 – FLUXO DE EXECUÇÃO [5]. 19
FIGURA 2.5 – ETAPAS ENVOLVIDAS NO DESENVOLVIMENTO DE UMA APLICAÇÃO NO *TINYOS* [5]. 20
FIGURA 2.6 – ETAPAS DE COMPILAÇÃO [5]. 20
FIGURA 2.7 – UM COMPONENTE E A SUA INTERFACE [5]. 21
FIGURA 2.8 – EXEMPLOS DE APLICAÇÃO DO PADRÃO ZIGBEE, EXTRAÍDO DE [8]. 22
FIGURA 2.9 – PROCESSO DE INTERACÇÃO DAS ONDAS ELECTROMAGNÉTICAS COM O CORPO HUMANO. 24
FIGURA 3.1 – CAMADAS DO PROTOCOLO IEEE 802.15.4, EXTRAÍDO DE [8]. 32
FIGURA 3.2 – ESTRUTURA *DATA FRAME FORMAT,* EXTRAÍDO DE [8]. 34
FIGURA 3.3 – ESTRUTURA *ACKNOWLEDGEMENT FRAME FORMAT,* EXTRAÍDO DE [8] 34
FIGURA 3.4 – ESTRUTURA MAC COMMAND FRAME FORMAT, EXTRAÍDO DE [8]. 35
FIGURA 3.5 – ESTRUTURA *BEACON FRAME FORMAT*, EXTRAÍDO DE [8]. 35
FIGURA 3.6 – ESTRUTURA *SUPERFRAME FORMAT*. 36
FIGURA 3.7 – PROTOCOLO S-MAC. 37
FIGURA 3.8 – PROTOCOLO T-MAC 39
FIGURA 3.9 – O NÓ D ENTRA EM REPOUSO ANTES DE C ENVIAR UM RTS (*EARLY SLEEPING)*. 40
FIGURA 3.10 – FUNDADORES. 43
FIGURA 3.11 – PARTICIPANTES. 43
FIGURA 3.12 – CAMADAS DEFINIDAS PELA ZIGBEE ALLIANCE E PELA IEEE 802.15.4, ADAPTADO DE [8]. . 44
FIGURA 4.1 – JANELA PARA VISUALIZAR E CONFIGURAR OS ALERTAS ENVIADOS PELOS NÓS SENSORES. 50
FIGURA 4.2 – EXEMPLO DOS PARÂMETROS PASSADOS AO COMANDO "NCC" PELO COMANDO "MAKE MICAZ". 54
FIGURA 4.3 – EXEMPLO DOS PARÂMETROS PASSADOS AO COMANDO "NCC" PELO COMANDO "MAKE MICAZ ROUTE,LP". 54
FIGURA 4.4 – MICAZ ZIGBEE SERIES (MPR2400) E RESPECTIVOS BLOCOS CONSTITUINTES [12]. 56
FIGURA 4.5 – PLACA PROGRAMADORA MIB510. 58
FIGURA 4.6 – DIAGRAMA DE ESTADOS DO MAC [14]. 66
FIGURA 4.7 – DIAGRAMA DE ESTADOS PARA O RÁDIO [14]. 67
FIGURA 5.1 – DISTRIBUIÇÃO DOS NÓS SENSORES 79
FIGURA 5.2 – CONSUMO DE ENERGIA EM CADA NÓ VARIANDO O DUTYCYCLE DO PROTOCOLO MAC TUNABLE 82
FIGURA 5.3 – CONSUMO DE ENERGIA EM CADA NÓ POR PROTOCOLO PARA A APLICAÇÃO NO CASTALIA 83
FIGURA 5.4 – CONSUMO DE ENERGIA EM CADA NÓ VARIANDO O TEMPO DE ESCUTA NO PROTOCOLO T-MAC DA APLICAÇÃO VALUEREPORTING NO CASTALIA 84
FIGURA 5.5 – CONSUMO DE ENERGIA PARA VÁRIOS PROTOCOLOS NA APLICAÇÃO SIMPLEAGREGATION DO CASTALIA. 84
FIGURA 5.6 – CONSUMO DE ENERGIA POR NÓ PARA VÁRIOS PROTOCOLOS NA APLICAÇÃO *MOBILITYTEST1* DO CASTALIA 85
FIGURA 5.7 – CONSUMO DE ENERGIA POR NÓ PARA VÁRIOS PROTOCOLOS NA APLICAÇÃO *MOBILITYTEST2* DO CASTALIA 86
FIGURA 5.8 – CONSUMO DE ENERGIA POR NÓ, PARA VÁRIOS PROTOCOLOS NA APLICAÇÃO MOBILITY3 DO CASTALIA 86
FIGURA 5.9 – CONSUMO DE ENERGIA EM CADA NÓ VARIANDO O *CONTENTIONPERIOD* DO PROTOCOLO T-MAC 87
FIGURA 5.10 – CONSUMO DE ENERGIA EM CADA NÓ NO PROTOCOLO T-MAC (*CONTENTIONPERIOD* = 1) 87
FIGURA 5.11 – CONSUMO DE ENERGIA EM CADA NÓ NO PROTOCOLO T-MAC (*CONTENTIONPERIOD* = 10) .. 88
FIGURA 5.12 – CONSUMO DE ENERGIA EM CADA NÓ NO PROTOCOLO T-MAC (*CONTENTIONPERIOD* = 20) .. 88
FIGURA 5.13 – CONSUMO DE ENERGIA EM CADA NÓ NO PROTOCOLO T-MAC (*CONTENTIONPERIOD* = 30) .. 89
FIGURA 5.14 – CONSUMO DE ENERGIA EM CADA NÓ NO PROTOCOLO T-MAC (*CONTENTIONPERIOD* = 40) .. 89
FIGURA 5.15 – CONSUMO DE ENERGIA EM CADA NÓ NO PROTOCOLO T-MAC (*CONTENTIONPERIOD* = 40) .. 90
FIGURA 5.16 – CONSUMO DE ENERGIA EM CADA NÓ NO PROTOCOLO T-MAC (*CONTENTIONPERIOD* = 50) .. 90
FIGURA 5.17 – CONSUMO DE ENERGIA EM CADA NÓ NO PROTOCOLO T-MAC (*CONTENTIONPERIOD* = 50) .. 91
FIGURA 5.18 – CONSUMO DE ENERGIA EM CADA NÓ NO PROTOCOLO T-MAC (*CONTENTIONPERIOD* = 100) 91

FIGURA 5.19 – DESVIO PADRÃO DO CONSUMO DE ENERGIA AO LONGO DA VARIAÇÃO DO PERÍODO DE CONTENÇÃO 92
FIGURA 5.20 – MÉDIA ARITMÉTICA DA ENERGIA CONSUMIDA AO LONGO DA VARIAÇÃO DO PERÍODO DE CONTENÇÃO 92
FIGURA F.1 – CONSUMO DE ENERGIA EM CADA NÓ NO PROTOCOLO T-MAC (*CONTENTIONPERIOD* = 1) ... 117
FIGURA F.2 – CONSUMO DE ENERGIA EM CADA NÓ NO PROTOCOLO T-MAC (*CONTENTIONPERIOD* = 10) . 117
FIGURA F.3 – CONSUMO DE ENERGIA EM CADA NÓ NO PROTOCOLO T-MAC (*CONTENTIONPERIOD* = 20) . 118
FIGURA F.4 – CONSUMO DE ENERGIA EM CADA NÓ NO PROTOCOLO T-MAC (*CONTENTIONPERIOD* = 30) . 118
FIGURA F.5 – CONSUMO DE ENERGIA EM CADA NÓ NO PROTOCOLO T-MAC (*CONTENTIONPERIOD* = 40) . 119
FIGURA F.6 – CONSUMO DE ENERGIA EM CADA NÓ NO PROTOCOLO T-MAC (*CONTENTIONPERIOD* = 50) . 119
FIGURA F.7 – CONSUMO DE ENERGIA EM CADA NÓ NO PROTOCOLO T-MAC (*CONTENTIONPERIOD* = 100) 120

Lista de Tabelas

TABELA 2.1 - EVENTOS *VERSUS* TAREFAS – RESUMO DAS ESTRUTURAS USADAS NA PILHA DO KERNEL [5].19
TABELA 2.2 - FREQUÊNCIAS E OS PRINCIPAIS EFEITOS BIOLÓGICOS....25
TABELA 3.1 – COOPERAÇÃO ENTRE ZIGBEE E O BLUETOOTH....47
TABELA 3.2 - CONSIDERAÇÕES DE TEMPORIZAÇÃO....47
TABELA 4.1 – DEFINIÇÃO DO COMANDO "MAKE … ROUTE,<STACK>"....54
TABELA 4.2 – SIMULADOR OMNET++. 59
TABELA 4.3 –SIMULADOR NETTOPO....60
TABELA 4.4 – SIMULADOR GTSNETS. 61
TABELA 5.1 – LOCALIZAÇÃO DOS NÓS EM COORDENADAS RECTANGULARES PARA A APLICAÇÃO VALUEREPORTING 80
TABELA 5.2 – LOCALIZAÇÃO DOS NÓS EM COORDENADAS RECTANGULARES PARA A APLICAÇÃO *VALUEPROPAGATION* E *SIMPLEAGREGATION*....81
TABELA 5.3 – LOCALIZAÇÃO DOS NÓS EM COORDENADAS RECTANGULARES PARA A APLICAÇÃO *MOBILITYTEST1* 81
TABELA 5.4 – LOCALIZAÇÃO DOS NÓS EM COORDENADAS RECTANGULARES PARA A APLICAÇÃO *MOBILITYTEST2* 81
TABELA 5.5 – LOCALIZAÇÃO DOS NÓS EM COORDENADAS RECTANGULARES PARA A APLICAÇÃO *MOBILITYTEST3* 82
TABELA B.1 – APP_NODE_STARTUP....99
TABELA B.2 – APP_DATA_PACKET 99
TABELA B.3 – MAC_SELF_CHECK_TX_BUFFER....101
TABELA B.4 – MAC_SELF_SET_RADIO_SLEEP 102
TABELA B.5 – MAC_SELF_WAKEUP_RADIO 103
TABELA B.6 – MAC_SELF_INITIATE_TX 103
TABELA B.7 – MAC_FRAME_SELF_PUSH_TX_BUFFER 105
TABELA B.8 – MAC_FRAME 105
TABELA B.9 – MAC_SELF_EXIT_EXPECTING_RX....106
TABELA B.10 – MAC_SELF_PERFORM_CARRIER_SENSE....107
TABELA B.11 – MAC_SELF_EXIT_CARRIER_SENSE 109
TABELA B.12 – RADIO_2_MAC_SENSED_CARRIER 109
TABELA B.13 – RADIO_2_MAC_FULL_BUFFER 110
TABELA B.14 – RESOURCE_MGR_OUT_OF_ENERGY....110

Lista de Siglas

ACK	*Acknowledge*
ADC	*Analog Digital Converter*
AEA	*Adaptive Election Algorithm*
ARC	*Adaptive Rate Control*
ARQ	*Acknowledgement request*
BAN	*Body Area Network*
BEB	*Binary Exponential Backoff*
CCA	*Clear Channel Assessment*
CPAN	*Comprehensive Perl Archive Network*
CPU	Unidade Central de Processamento
CSIRO	*Commonwealth Scientific and Industrial Research Organisation*
CSMA/CA	*Carrier Sense Multiple Access/Collision Avoidance*
CSMA/CD	*Carrier Sense Multiple Access/Collision Detection*
CTS	*Clear To Send*
DBI	Integração com base de dados
DCF	*Distributed Coordination Function*
DE-MAC	*Energy-Medium Access Control*
EB	Estação de Base
EEPROM	*Electrically Erasable Programmable Read-Only*
FCC	*Federal Communications Commission*
FCS	*Frame Check Sequence*
FDMA	*Frequency Division Multiple Access*
FFD	*Full Function Device*
ISM	*Industrial Scientific and Medical*
LEACH	*Low Energy Adaptive Clustering Hierarchy*
MANET	*Mobile Ad-hoc Network*
MAC	*Medium Access Control*
MeAC	*Message Authentication Code*
NAK	*Not Acknowledge*
MEF	Máquina de Estados Finitos
MEMS	*Micro-Electro-Mechanical Systems NesC*

NP	*Neighbor Protocol*
NSF	*National Science Foundation*
OSI	*Open Systems Interconnection*
PC	*Personal Computer*
PDA	*Personal Digital Assistants*
PEGASIS	*Power Efficient Gathering in Sensor Information System*
QoS	Qualidade de serviço (*Quality of Servive*)
RFD	*Reduced Function Device*
RSMSF	Redes de Sensores Multimédia Sem Fios
RSSF	Rede de Sensores Sem Fios
RSSI	*Received Signal Strenght Indicator*
RTOS	Sistemas operativos em tempo real
RTS	*Request To Send*
SAR	*Sequential Assignment Routing*
SAR	Taxa de absorção específica
SDMA	*Space Division Multiple Access*
SEP	*Schedule Exchange Protocol*
S-MAC	*Sensor Medium Access Control*
SMP	*Sensor Management Protocol*
SPIN-PP	*Sensor Protocol for Information via Negotiation*
SQDDP	*Sensor Query and Data Dissemination Protocol*
STORM /AD	*Self-organizing TOpology discoveRy and Maintenance Adaptive Diffusion/ Adaptive Diffusion*
SYNC	Pacote de sincronização
TCP/IP	*Transmission Control Protocol/ Internet Protocol*
TDMA	*Time Division Multiple Access*
TEEN	*Threshold-sensitive Energy Efficient Network*
T-MAC	*Timeout Medium Access Control*
TRAMA	*Traffic adaptive Multiple Access*
UNII	*Unlicensed National Information Infrastructure*
WBAN	*Wireless Body Area Network*
WEBS	Wireless Embedded System
WMSN's	Wireless Multimedia Sensor Networks
WSN	*Wireless Sensor Network*

Lista de Símbolos

a_n	Energia necessária para a transmissão do pacote passando pelo caminho n
D	Distância entre o emissor e o receptor
n	Expoente de propagação
G	Permutador
K	Subchave de encriptação
PA	Potência disponível
RTT_{RTS}	Intervalo de tempo de transmissão de um pacote RTS (ida e volta).
T_A	Período de tempo após o qual termina o tempo activo caso exista inactividade durante T_A
$t_{contenção}$	Intervalo de tempo de contenção.
t_{RTS}	Intervalo de tempo do pacote de RTS.
w_i	$Mensagem_i$

Lista de Programas

TABELA 4.1 – DEFINIÇÃO DO COMANDO "MAKE … ROUTE,<STACK>".....54

Capítulo 1
Introdução

1.1 Introdução

O objectivo da Investigação e Desenvolvimento (I&D) que motiva o aparecimento desta dissertação de Mestrado é a necessidade actual de desenvolver e validar protocolos de controlo de acesso ao meio eficientes do ponto de vista energético que permitam lidar com a complexidade de uma Rede de Sensores Sem Fios (RSSF) e suportar a mobilidade dos nós, podendo ser escaláveis com o aumento do número de nós.

Irão propor-se soluções utilizando técnicas que permitam a poupança de energia e o aumento do tempo de vida das redes, de uma forma fiável, com a cobertura e a "conectividade" desejada, garantindo simultaneamente a qualidade de serviço. As aplicações visadas incluem a monitorização de equipas desportivas, aplicações médicas, *Wireless Body Area Networks* (WBAN) aplicadas a *Wearable Technologies* e aplicações militares de segurança.

1.2 Motivação

As RSSFs (WSNs - *Wireless Sensor Networks*, em terminologia Anglo-Saxónica) são redes compostas por vários nós sensores, muitas vezes sem uma localização pré-definida para cada sensor. Têm por base uma organização não infraestruturada, ou seja, sem estações de base. São portanto redes ad-hoc que conseguem detectar, converter e transmitir características físicas do meio envolvente. Tipicamente, as RSSFs possuem um nó ou terminal especial para o qual todas as

comunicações convergem, designado por nó destino (*sink node*) o que não acontece com as redes ad-hoc sem fios, *Mobile Ad-hoc Networks* (MANET). A informação contida nos nós sensores é encaminhada e poderá ser guardada numa base de dados central. Do ponto de vista de organização, as RSSFs e as MANETs são idênticas, já que possuem nós que comunicam directamente entre si através de sequências de saltos com comunicação sem fios. No entanto, as RSSFs têm especificidades bem diferentes das redes comuns. Por exemplo, as RSSFs possuem algumas características particulares tais como a utilização de recursos restritos de energia, a utilização de uma topologia de rede dinâmica e a utilização de uma grande quantidade de nós.

O consumo de energia em RSSFs depende da operação que é efectuada e do estado em que se encontra o processador: em carga, parado (*idle*), ou em repouso (*sleep*). Tipicamente, a emissão consome mais energia que a recepção, e a escrita em memória pode também apresentar valores do consumo de energia da mesma ordem de grandeza. O processamento central, ao nível do nó sensor, é realizado por um processador cujo consumo depende da frequência do relógio. Tipicamente este consumo de energia é proporcional à frequência do relógio. Para prolongar a vida útil das baterias é possível carregá-las utilizando alguma energia do meio envolvente.

Como as RSSFs devem executar tarefas colaborativas, onde é importante detectar e estimar o aparecimento de eventos (e não apenas fornecer mecanismos de comunicação), normalmente, os dados são resumidos ou tratados (em *clusters*) para melhorar o desempenho no processo de detecção de acontecimentos.

A poupança de energia é uma questão muito importante nas RSSFs na medida em que os nós sensores são frequentemente abastecidos por baterias sendo necessário que estes mesmos nós façam uma gestão inteligente da energia nas suas baterias. Os avanços da tecnologia a nível de hardware resultaram no aparecimento de nós sensores de baixo custo os quais já são compostos por memória, processador e emissor/receptor (*transceiver)*, embebido. As capacidades de baixa potência levaram a uma cobertura e alcance da comunicação limitada comparando com outros dispositivos. Ao contrário das outras redes sem fios, geralmente é difícil (ou impraticável) efectuar o carregamento/ substituição das baterias descarregadas. Por isso, o primeiro objectivo é a maximização do tempo de vida do nó/rede, deixando as restantes características de desempenho como objectivos secundários. Dado que a comunicação dos nós sensores consome mais energia do que o processamento dos dados dos nós sensores, tem-se como preocupação principal que a comunicação entre os diferentes nós sensores seja minimizada, para que

assim se obtenha uma rede de sensores sem fios funcional. Para resolver este aspecto são utilizados os protocolos de acesso ao meio, também conhecidos por protocolos *Medium Access Control* (MAC) de forma a gerir-se a comunicação entre os nós sensores que compõem a RSSFs.

Um tema que tem vindo a interessar investigadores nos últimos anos é a coexistência de outros sistemas de comunicação sem fios, com as redes de sensores sem fios, com ambos os sistemas a operar na mesma área de cobertura. Isto porque no mercado existem muitos sistemas que estão a optar por usar bandas de frequência não licenciadas. Exemplo disso são as bandas de frequência de 900 MHz, 2.4 GHz e 5.8 GHz, pertencentes à banda ISM (*Industrial Scientific and Medical*) e a banda UNII (*Unlicensed National Information Infrastructure*), que opera na frequência dos 5.2 GHz.

As regras aplicadas na utilização de bandas não licenciadas, tais como a banda ISM, são muito pouco restritivas. Logo, a utilização simultânea da mesma banda de nós pertencentes a sistemas diferentes poderá ser uma desvantagem, pois pode existir um elevado risco de existência de interferências, que poderão afectar os níveis de qualidade.

1.3 Desafios e Abordagem

Hoje em dia, a necessidade de recolher dados através de sensores, interpretá-los e actuar em dispositivos em tempo real está a ganhar um interesse crescente em várias áreas tais como na indústria, medicina, desporto, monitorização ambiental ou na agricultura. Em algumas destas situações, devido á complexidade dos sistemas, só será viável recolher dados através de RSSFs. Estas redes são compostas por vários nós sensores com base numa organização dinâmica e não infraestruturada, ou seja, sem estações de base (redes ad-hoc). Através dos sensores conseguem-se detectar características físicas do meio envolvente que serão depois convertidas e transmitidas.

Aborda-se uma utilização para os nós sensores baseada na recolha de dados num campo agrícola onde teremos sensores a monitorizar a humidade do terreno (comunicação sem mobilidade) ou onde teremos sensores instalados numa manada para recolha de dados tal como a sua localização (comunicação com mobilidade). Estabelecem-se as principais características da norma IEEE 802.15.4 tais como as camadas e protocolos MAC, o controlo de acesso ao meio, e a estrutura das tramas de dados, ZigBee. Exemplifica-se a forma de verificar as características de uma rede multi-hop (multi-salto). Descreve-se uma poderosa linguagem de programação (Perl) utilizada no compilador do TinyOS. Seleccionaram-se os nós sensores e exploram-se diversos

protocolos MAC que se pretendem optimizar do ponto de vista energético. Considera-se o simulador Castalia que funciona sobre o OMNeT++ e obtêm-se resultados de consumo de energia para redes de complexidade média.

Da investigação associada a esta dissertação resultou numa contribuição inovadora que foi publicada em revista:

Jorge Tavares, Fernando J. Velez, João M. Ferro, "Application of Wireless Sensor Networks to Automobiles," *MEASUREMENT SCIENCE REVIEW*, Vol. 8, No. 3, pp. 65-70, 2008.

1.4 Estrutura do trabalho

Apresenta-se de seguida a estrutura desta dissertação.

No Capítulo 2 é feita uma abordagem ao estado actual das RSSFs relativamente a projectos já desenvolvidos ou ainda em curso. Apresenta-se sistema operativo TinyOS direccionado para RSSFs, assim como a linguagem NesC utilizada por este. Identificam-se algumas aplicações das RSSFs, e identificam-se os possíveis efeitos das radiações radioeléctricas, com base em alguns estudos recentes.

No Capítulo 3 descrevem-se características da norma IEEE 802.15.4 nomeadamente formato da frame de dados, camadas de rede, gestão de energia, fiabilidade e robustez, protocolos da camada de ligação entre outras.

No Capítulo 4 identificam-se as características multi-hop (multi-salto) das RSSFs assim como será descrito um exemplo em que se tira partido dessa característica. Descreve-se a linguagem Perl, que é uma linguagem utilizada no desenvolvimento dos ficheiros do TinyOS que dizem respeito à compilação e geração do executável da aplicação. Também se apresenta algumas das carateristicas do hardware a utilizar nestas redes. Abordam-se as características de alguns simuladores, identificando-se as características que levaram a seleccionar o simulador Castalia (que corre sobre o OMNeT++).

No Capítulo 5 apresenta-se o cenário de simulação assim como os resultados das simulações efectuadas. Descrevem-se alguns parâmetros e quais as suas implicações e apresentam-se as melhorias conseguidas, através da sua variação. Extraem-se conclusões acerca do consumo de energia e tempo de vida das RSSFs para situações onde o funcionamento individual é crítico e para casos onde o seu funcionamento é crítico apenas do ponto de vista global da rede.

Por fim, no Capitulo 6, são apresentadas as conclusões deste trabalho assim como sugestões de melhorias que poderão ser implementadas no futuro.

Após a conclusão são apresentados, em anexo, alguns detalhes complementares dos diagramas, resultados mais detalhados e ficheiros do Castalia, assim como as referências consultadas na elaboração deste projecto.

Capítulo 2
Características e Aplicações

2.1 Introdução

As RSSFs encontram-se em constante evolução tanto a nível de software, como a nível de hardware e áreas de aplicação. Neste capítulo indica-se alguns dos projectos que estão a decorrer ou que tiveram lugar nos últimos anos. É feita também uma descrição do sistema operativo TinyOS

2.2 Alguns projectos em curso

A *Commonwealth Scientific and Industrial Research Organisation* (CSIRO), na Austrália, desenvolveu projectos desde Junho de 2006 a Julho de 2007, tais como [1]:

- ***Starin Gauge Interface*** – Pretende medir a tensão mecânica para monitorizar as condições de funcionamento de máquinas assim como para a análise da resistência á fadiga;
- **Interface para a qualidade da água** – Interface que liga sensores de medição da qualidade da água a um nó Flex;
- **Interface para estufas** – Interface para vários sensores que monitorizam dados relevantes para o crescimento das plantas, tais como, luz, humidade do ar, e temperatura e humidade do solo;

- ***Soil Moisture Interface: Gypsum Block*** – Interface para sensor de tensão da água que mede a facilidade com que as raízes de uma planta podem extrair humidade da terra;
- ***Soil Moisture Interface: Echo Probe*** – Interface padrão para sensor de humidade no solo que mede a concentração volumétrica de água na terra;
- ***Inertial Interface*** – Interface que mede a posição e orientação no espaço tridimensional sendo particularmente útil para o estudo do movimento humano e animal;
- ***GPS/MMC Interface*** – Interface que serve para medir a posição e a velocidade em ambientes exteriores;
- **Programação através da rede** – Como os nós são programados tipicamente ao nível do nó e não ao "nível de toda a rede" está-se a desenvolver uma nova linguagem de alto nível para que possam ser endereçados e programados através da rede. O paradigma da programação orientada por eventos limita o desenvolvimento de aplicações, estando-se por isso a desenvolver um novo sistema operativo baseado em processos;
- **Fontes de energia** – Para se prolongar as reservas de energia das baterias aposta-se habitualmente numa grande racionalização dos consumos do nó [1]. A CIRRO explora uma abordagem diferente, em que fornece-se energia absorvida do ambiente aos nós, permitindo assim um maior tempo de operação destes;
- **Energia solar** – É necessário que a energia armazenada durante o dia seja suficiente para manter o nó em funcionamento durante a noite [1]. A CSIRO conseguiu colocar nós a funcionar a energia solar durante 24 horas através do uso de supercondensadores, possibilitando assim viabiliza o funcionamento com este tipo de energia. Com esta tecnologia foi efectuado um teste com uma rede que se mantém em funcionamento, há pelo menos dois anos;
- **Antena robusta** – Foram desenvolvidas antenas com uma resistência adequada, que operam entre os 915MHz e os 928 MHz de frequência com ao objectivo de as montar em animais, tendo por isso mesmo que ser mais robustas que as antenas habituais, com uma impedância e um ganho insensível á sua montagem [1];
- ***Spatial Diversity System*** – Melhoria da qualidade das comunicações entre os nós de uma rede de sensores sem fios, possibilitando um aumento da distância

de comunicação [1]. Possui algoritmos de controlo e de troca de rádio frequência que poderão ser utilizados com duas ou mais antenas optimizadas para a banda de frequências em uso. Permite que se utilizem as antenas robustas pois estas teriam um baixo desempenho sem este sistema;

- **Autenticidade e confidencialidade** – Desenvolveu-se um protocolo para proteger a autenticidade e confidencialidade de uma mensagem difundida para toda a rede. A nível da localização dos nós focaram-se em facilitar a sua localização, nos casos em que não há linha de vista (LoS), o que permite melhorar a precisão do posicionamento do nó.
- ***Farm Environement*** – Actualmente a agricultura encontra vários desafios tais como a mudança de clima, escassez de água, diminuição da população que trabalha nesta área, e consciencialização sobre muitos problemas como bem-estar animal, segurança alimentar e impacto ambiental. Assim como em alguns países onde a sobre exploração das terras levou a uma maior erosão do solo e diminuição da qualidade da água. As Redes de Sensores são ferramentas importantes para medir e perceber a complexidade destes sistemas naturais. Está-se a desenvolver uma Horta Inteligente (*Smart Farm*) com a aplicação da tecnologia de RSSFs para a agricultura animal para tentar resolver estas exigências. Criou-se uma rede autoconfigurável penetrante, utilizando dispositivos simples e baratos, que recolhem dados sobre o meio ambiente em que se encontram inseridos, utilizando-os em benefício próprio.
- ***Sensor and Actuator Networks for Virtual Fencing*** – O controlo do gado através das redes de sensores e actuadores sem fios foi uma chave importante do programa de investigação realizado ate á data pela CSIRO. O objectivo deste trabalho foi o de desenvolver uma metodologia para combinar conhecimento científico sobre os animais com as ciências informáticas de modo a adquirir controlo espacial autónomo do gado através das linhas de uma cerca virtual. Em Abril de 2007 conseguiu-se de forma autónoma manter uma manada atrás de uma cerca virtual durante 13 horas, sendo que na maioria dos casos apenas foram utilizados estímulos audíveis para controlar o gado. Na próxima fase do trabalho pretende-se tratar dos assuntos de forma escalável assim como procurar informação sobre o estado de todo o rebanho, em vez de tomar decisões ao nível de apenas um animal

- ***Bull Separation*** – Separação de touros para prevenir danos na manada;
- ***A Sensor for Detecting Coliform*** – A poluição existente em águas recreativas é tradicionalmente avaliada pela recolha de amostras de água. Esse processo é algo demorado pois só quando a colónia de bactérias for suficientemente numerosa se consegue identificar a presença de bactérias. A maioria das técnicas de análise demoram mais de 24 horas para a obtenção do resultado. Existe um método de análise baseado na percepção de enzimas especificamente marcadas pelo crescimento de bactérias que promete resultados mais rápidos. As enzimas ficam florescentes e podem ser detectadas visualmente em aproximadamente 8 horas.
- **TinyOS 2.x** – Desenvolvido pela Barkley veio substituir o TinyOS 1.x. Este novo desenvolvimento ficou-se a dever ao facto de muitos aspectos do TinyOS 1.x estarem a ser solicitados para tarefas para as quais não foram inicialmente pensados e projectados. Enquanto essas limitações eram contornadas, foram criados componentes com interacções difíceis, e uma curva de aprendizagem muito íngreme para um recém programador de RSSFs. O novo TinyOS 2.x não é compatível com a versão anterior.
- **SyncWUF** – Protocolo MAC de muito baixo consumo projectado para RSSFs [2]. Este protocolo resulta de combinar o esquema Wake-Up-Frame e o WiseMAC que são dois esquemas eficientes baseados nos chamados preâmbulos de tecnologia de amostragem. Através da sincronização no momento de transmissão com o receptor, diminui-se o preâmbulo de wake-up com o objectivo de obter um protocolo MAC mais optimizado para o baixo tráfego nas RSSFs. As análises e simulações efectuadas comprovaram que se consegue um ganho significativo na vida das baterias comparativamente ao protocolo original WiseMAC e a norma IEEE 802.15.4. Manteve-se de forma similar a eficiência e o tempo de resposta comparativamente com ao norma IEEE 802.15.4, em diferentes aplicações e sem afectar negativamente outros parâmetros importantes tais como a capacidade e a latência do canal.
- ***Inner-Circle Consistency*** – A iteração local entre os nós é utilizada para neutralizar erros ou ataques impedindo-os de se propagarem pela rede e melhorando assim a fidelidade da informação enviada [3]. Para alcançar esse objectivo combinou-se informação estatística e técnicas de criptografia com as

aplicações para verificar que os dados que são naturalmente replicados pelas aplicações desenvolvidas para as redes de sensores sem fios.

- **QoS Surview** – A disponibilidade de hardware a baixo custo tal como os microfones e as câmeras CMOS (Complementary Metal-Oxide-Semiconductor) incentivou o desenvolvimento de Redes de Sensores Multimédia Sem Fios (RSMSF), do inglês *Wireless Multimedia Sensor Networks* (WMSN's).

Office Deployment – Projecto da CSIRO de gestão de energia baseada em agentes, para ambiente de escritório [1]. São utilizados os nós FleckTMs como instrumentos de medição nos seguintes locais:

1. lojas grandes e frias;
2. chão de uma superfície comercial.

Neste projecto acredita-se, que com um pequeno investimento se consegue um significativo retorno financeiro, ao se utilizar esta tecnologia com vista a obter uma resposta de demanda apropriada. Neste ambiente de escritório a tecnologia de RSSFs provou ser uma técnica confiável e apropriada na gestão da temperatura, da humidade e de dados ocupacionais.

James Reserve Experiment – foi instalado uma estação de campo pertencente ao sistema da Universidade da Califórnia numa remota reserva rodeada de floresta [1]. Os nós foram instalados durante mais de um ano para estudar a performance do rádio e da energia solar.

Burdekin Irrigation Network – A infiltração de sal em lençóis de água subterrâneos, tem sido uma preocupação contínua na gestão global da água nas zonas costeiras [1]. As principais decisões a ser tomadas em relação á exploração dos recursos hídricos em locais no litoral são: em que local extrair a água e que quantidade pode ser extraída de forma sustentável. Caso se retire demasiada água, poderá ocorrer uma infiltração da água salgada, nos lençóis de água, provocando danos cuja correcção é bastante dispendiosa.

De seguida descrevem-se alguns projectos com recurso a RSSFs em desenvolvimento na China:

- **Vigilância de minas de carvão** – As ameaças existentes em minas de carvão são o fogo, as explosões, as inundações e as quedas de paredes e tectos, [2]. Uma vigilância em tempo real das condições críticas envolventes, tais como gás, pó, oxigénio, e água é crucial.

- **Monitorização da poluição da água** – Vai-se desenvolver sensores para monitorizar sistemas de detecção da poluição da água, tentando encontrar a origem da poluição, para proceder ao aviso, com alertas em tempo real [3].
- **Monitorização e controlo de tráfego urbano** – será criada uma nova plataforma para o nó sensor, sendo esta caracterizada pelo baixo consumo de energia e por uma boa interface para sensores. Serão desenvolvidos sensores mais adequados aos objectivos [3].

2.3 TinyOS

O *TinyOS* foi desenvolvido pelo Departamento de Engenharia Electrotécnica e Ciências da Computação (EECS) da Universidade da Califórnia – Berkeley para trabalhar com o microprocessador ATMEL AT (©Atmel Corporation) [5]. Trata-se de um sistema operativo bastante simples e compacto, baseado em eventos, desenvolvido para atender alguns dos requisitos das RSSFs, tais como, possibilidade de com o mínimo de recursos de hardware, se ter grandes operações de interligação dos componentes; e baixo consumo de energia. O NesC, linguagem de programação utilizada no *TinyOS,* é uma linguagem de alto nível baseada na linguagem C, podendo-se dizer que se encontra a um nível mais elevado que esta, pois a compilação do NesC irá gerar os correspondentes ficheiros em linguagem C.

2.3.1 História do *TinyOS*

O *TinyOS* foi inicialmente desenvolvido por Jason Hill, no seu projecto de Mestrado – UC Berkeley – 2000 [5]. Foi orientado pelo Professor David Culler num projecto que envolveu a Universidade de Berkeley e a Intel. O *TinyOS* passou a fazer parte do Projecto Berkeley *Wireless Embedded System* (WEBS). Outros trabalhos de RSSF que estão a ser desenvolvidos pelo mesmo grupo em RSSFs são os seguintes:

- **TOSSIM** – simulador para o *TinyOS;*
- **Maté** – máquina virtual para o *TinyOS*;
- **TinyDB** – sistema de processamento de consultas desenvolvido para extrair informação de uma RSSF utilizando *TinyOS*;
- **NesC** – compilador personalizado específico para o *TinyOS*;
- **Calamari** – sistema de localização para redes de sensores;
- **Great Duck Island** – aplicação que emprega nós sensores chamados de

motes para monitorizar habitantes e vida selvagem;

- **TinySec** – camada de enlace com criptografia voltada para pequenos dispositivos.

Actualmente, as modificações no código e nos componentes do *TinyOS* são feitas pelo grupo *Intel-Berkeley Research Lab*.

2.3.2 Descrição do *TinyOS*

O *TinyOS* é composto por um sistema operativo simples, um ambiente de desenvolvimento em código aberto, um modelo e uma linguagem de programação [5]. O sistema operativo utiliza acontecimentos e um conjunto de serviços, como se descreve a seguir.

O *TinyOS* possui uma pilha de tarefas, que é uma fila "*First In First Out*" (FIFO), utilizando uma estrutura de dados de tamanho limitado. A pilha desenvolvida para concorrência intensiva não é multi-tarefa e não possui mecanismos sofisticados como fila de prioridades. Os nós sensores monitorizam acontecimentos do mundo real que são inerentemente concorrentes (pode ocorrer mais do que um evento no mesmo intervalo de tempo). Como podem ocorrer vários eventos, as tarefas que lhes podem atender têm que ser executadas eficientemente. O sistema operativo tem que ser projectado para que todos os eventos possam ser atendidos atempadamente. Para além disso, os recursos computacionais num nó sensor são limitados. Dessa forma, o *TinyOS* também foi construído para trabalhar com recursos limitados, facilitando o desenvolvimento de componentes de *software* orientados para a eficiência e modularidade. O *TinyOS* possui um conjunto de ferramentas que permitem desenvolver código para este sistema. O código é aberto, e está disponível em [6].

O *TinyOS* também é um modelo e uma linguagem de programação. Os programas são construídos a partir de um conjunto de componentes. A especificação do comportamento dos componentes é feita através de um conjunto de interfaces. Os componentes são ligados estaticamente uns aos outros via interfaces, o que aumenta a eficiência em tempo de execução porque as ligações entre componentes são verificadas durante a compilação.

O *TinyOS* possui um modelo de eventos que permite ter concorrência utilizando pouco espaço de memória. Um modelo baseado em troca de contexto e pilha iria requerer que o espaço da pilha fosse reservado para cada troca de contexto, para além de ser necessário despender processamento em cada troca. Para além disso, a energia é um

recurso precioso. O modelo baseado em eventos alcança um ganho de 12 vezes comparado com o modelo com troca de contexto. Os ciclos não utilizados da *Central Processing Unit* (CPU) são utilizados no estado "*sleep*", ao invés de procurar activamente um evento.

Finalmente, o *TinyOS* também constitui um conjunto de serviços. Assim o *TinyOS* disponibiliza interfaces e componentes que implementam os principais serviços de uma aplicação de RSSFs e estes serviços são:

- **Rádio, MAC, Mensagens, Encaminhamento** - componentes que implementam a pilha de protocolos do *TinyOS*.
- **Interface de Sensores** - conjunto de interfaces para os mais variados tipos de sensores que podem ser utilizados num nó sensor com *TinyOS*.
- **Gestão de Energia** - energia é o recurso mais crítico de uma RSSF.
- **Depuração** - fornece componentes e ferramentas que permitem a depuração (debuggers) de código.
- **Temporizadores** - fornece componentes de acesso aos relógios do nó sensor.

2.3.3 Objectivos do *TinyOS*

Os objectivos do *TinyOS* são os seguintes [5]:

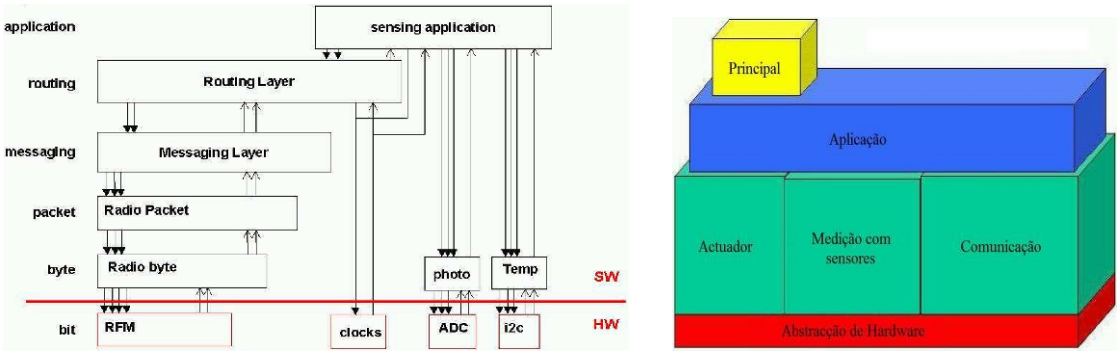
- **Atender sistemas embutidos em redes** – o sistema deveria "dormir", mas permanecer vigilante a estímulos. Quando ocorrem um ou mais eventos, estes devem ser atendidos.
- **Mica *hardware*** – o *TinyOS* é voltado especificamente para a arquitectura Mica Motes. Numa plataforma, o *TinyOS* implementa tarefas e componentes de energia, medição com sensores, computação e comunicação.
- **Acompanhar os avanços tecnológicos** – prevendo um desenvolvimento da tecnologia MEMS, o *TinyOS* deve acompanhar os avanços tecnológicos, para que não fique ultrapassado rapidamente. A tendência dos circuitos integrados é a de continuar a reduzir as dimensões, ficando mais barato, e incluindo tecnologias de baixo consumo de energia (*low power*).

As RSSFs não podiam utilizar os sistemas operativos em tempo real (RTOS) existentes. A arquitectura desses sistemas como o VxWorks, o QNX, o WinCE, o PalmOS e o μLinux são similares aos de *desktops*. Por outro lado, os sistemas operativos voltados para PDAs, telemóveis são sistemas embutidos baseados na arquitectura de PC sendo uma ordem de grandeza mais lentos e pesados, sem contar

com o consumo proibitivo de energia.

2.3.4 Eventos, Comandos e Tarefas

O *TinyOS* representa um novo paradigma de programação, incluindo conceitos de eventos, comandos e tarefas [5]. Uma configuração completa do sistema consiste num minúsculo programa composto por uma aplicação e pelos componentes do *TinyOS*. Uma aplicação é, na verdade, um conjunto de componentes agrupados, conforme apresentado na Figura 2.1 (a). Estes também podem ser agrupados em camadas, como ilustra a Figura 2.1 (b).



(a) (b)

Figura 2.1 – Uma aplicação pode ser vista como um grupo de componentes [5].

Um componente possui quatro partes relacionadas: um conjunto de comandos, um conjunto de gestores de eventos, uma *frame* (trama) encapsulada de tamanho fixo e um pacote de tarefas simples. As tarefas, os comandos, e os gestores de eventos executam-se no contexto da trama e operam sobre o seu estado. Para facilitar a modularidade, cada componente também declara os seus comandos e os eventos que sinaliza. Estas declarações são usadas para compor os componentes. O processo de composição cria camadas de componentes, onde os componentes de alto nível emitem comandos aos componentes de baixo nível e os componentes de baixo nível sinalizam eventos aos componentes de alto nível. A parte física de hardware representa o nível mais baixo dos componentes.

As tramas possuem tamanho fixo e são atribuídos estaticamente, permitindo conhecer os requisitos de memória de um componente em tempo de compilação, o que evita o custo adicional associado à utilização de atribuição dinâmica. A atribuição estática de memória também diminui o tempo de execução, uma vez que as posições das

variáveis podem ser definidas estaticamente no programa. Os comandos são pedidos assíncronos feitos aos componentes do nível inferior. Um comando pode colocar uma tarefa para executar futuramente. Ele pode também invocar alguns comandos de níveis inferiores, mas não deve esperar executar acções longas ou de latência indeterminadas. Um comando deve fornecer uma resposta a quem o chamou, indicando se foi bem sucedido ou não. Já os gerênciadores de eventos (*event handlers*) são invocados para tratar dos eventos do *hardware*, directamente ou indirectamente. Os componentes de baixo nível têm os gerenciadores ligados directamente às interrupções do *hardware*, que podem ser interrupções externas (eventos do temporizador) ou eventos contadores. Um gerênciador de eventos pode iniciar tarefas, sinalizar eventos de alto nível ou chamar comandos do nível inferior. Um evento de *hardware* provoca uma sequência de processamento que sobe de nível através dos eventos e pode descer através dos comandos, Figura 2.2. Com a finalidade de evitar ciclos em cadeia de comandos/eventos, os comandos não podem sinalizar eventos.

As tarefas executam trabalhos básicos, sendo atómicas no que diz respeito a outras tarefas e são executadas completamente até ao fim. As tarefas podem chamar comandos de nível inferior, sinalizar eventos de um nível superior e programar outras tarefas dentro de um componente. A execução completa das tarefas torna possível reservar uma única pilha, que é atribuída à tarefa que é processada no momento. Isto é essencial em sistemas com restrição de memória. As tarefas permitem a simulação de concorrência dentro de cada componente, desde que executadas de forma assíncrona em relação aos eventos. Entretanto, as tarefas nunca devem obstruir ou prolongar a espera, pois bloqueariam outros componentes.

A pilha de tarefas é uma fila FIFO, utilizando uma estrutura de dados de tamanho limitado. O processador entra no modo de repouso cada vez que a fila de tarefas está vazia, mas os periféricos continuam a operar, de modo a que alguns deles podem mesmo reactivar o sistema. Este comportamento permite o uso eficiente da energia. Uma vez que a fila está vazia, uma outra tarefa pode ser programada somente em consequência de um evento, assim não há a necessidade de "acordar" o empilhador da fila, até que um novo evento do hardware provoque uma actividade. A aplicação também é responsável pela gestão de energia, que é o recurso mais crítico em RSSF.

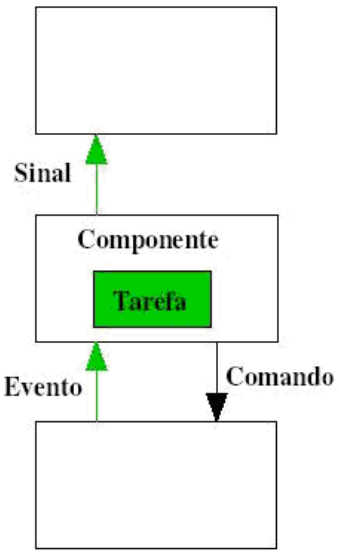


Figura 2.2 – Gestor de eventos [5].

A Figura 2.3 apresenta o conceito de componente. O componente possui uma interface com métodos definidos (representado na figura por setas), que o permite agrupar com outros componentes tanto no sentido para cima, como para baixo. Um componente típico inclui uma trama, gerênciadores de evento, comandos e tarefas para um componente de manipulação de mensagem. Como a maioria dos componentes, este exporta comandos de inicialização e gestão de energia; possui um comando para iniciar a transmissão de uma mensagem e sinaliza eventos ao concluir a transmissão ou quando chega uma mensagem. Os componentes de transmissão da mensagem editam comandos no pacote ao nível do componente e definem dois tipos de eventos. O primeiro indica que a mensagem foi transmitida e o segundo sinaliza que a mensagem foi recebida. A ligação entre os componentes é bastante simples, desde que os componentes descrevam os recursos que fornecem e os recursos que requerem. Assim, o programador combina simplesmente as assinaturas dos eventos e dos comandos requeridos por um componente com as assinaturas dos eventos e dos comandos fornecidos por um outro componente. A comunicação através dos componentes examina um formulário de chamada de função, que possui os dados necessários para a comunicação e fornece a verificação do tipo em tempo de compilação.

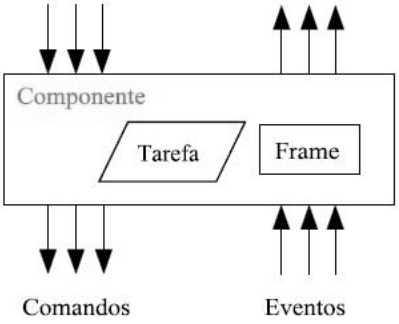


Figura 2.3 – Componente [5].

2.3.5 O Projecto do Kernel TinyOS

O projecto do *kernel* do *TinyOS* é baseado numa estrutura de dois níveis de escalonamento, Tabela 2.1 [5]:

- **Eventos** – utilizam uma pequena quantidade de processamento (interrupções de relógio, interrupções ADC) e podem interromper tarefas que estejam em execução.
- **Tarefas** – utilizam uma quantidade maior de processamento e não são críticas quanto ao tempo como eventos. Tarefas sempre executam até completarem. Esta propriedade é muito importante, pois permite que o *TinyOS* utilize apenas uma pilha de execução.

A Figura 2.4 apresenta o fluxo de execução de tarefas (rectângulos), eventos (setas para cima) e comandos (setas na diagonal). As interrupções são mapeadas como eventos. Um evento pode accionar comandos ou accionar a execução de uma tarefa. Uma tarefa é formada por um conjunto de comandos e não pode bloquear outra tarefa, mas um evento pode interromper a execução de uma tarefa. Na Figura 2.4, uma interrupção gera um evento. Este gera um evento para o nível superior, que por sua vez executa um comando e gera um evento para o nível superior. O gestor de eventos executa dois comandos, e cada um inicia uma tarefa, que executa um conjunto de comandos. Os eventos gerados por interrupção podem interromper as tarefas, enquanto que as tarefas não podem interromper tarefas. As tarefas e os gestores de eventos executam um grupo de comandos.

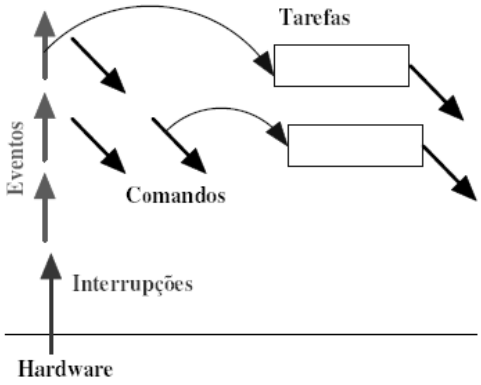


Figura 2.4 – Fluxo de execução [5].

Tabela 2.1 - Eventos *versus* Tarefas – Resumo das estruturas usadas na pilha do kernel [5].

Item	Eventos	Tarefas
Processamento	Pequenas quantidades de processamento	Não são críticas em relação ao tempo
Exemplos	Relógios e Interrupções; ADC (conversão analógico digital)	Calcular a média de um vector
Propriedades	Podem interromper tarefas em execução	Executam até terminar

2.4 NesC

De seguida apresenta-se uma breve descrição da linguagem de programação NesC, que é utilizada nas aplicações para os motes, criadas em ambiente *TinyOS* [5]. As etapas que envolvem o desenvolvimento de uma aplicação no *TinyOS* são apresentadas na Figura 2.5.

O código fonte desta aplicação, junto com o kernel do *TinyOS* e as bibliotecas do *TinyOS* são compiladas por um compilador NesC, resultando no código de uma aplicação com *TinyOS* em linguagem C. Este código é compilado por um compilador C, gerando o executável de uma aplicação. Os tipos de arquivos em cada uma das etapas do desenvolvimento da aplicação possuem extensões diferentes, que servem para identificar o formato de cada um dos arquivos.

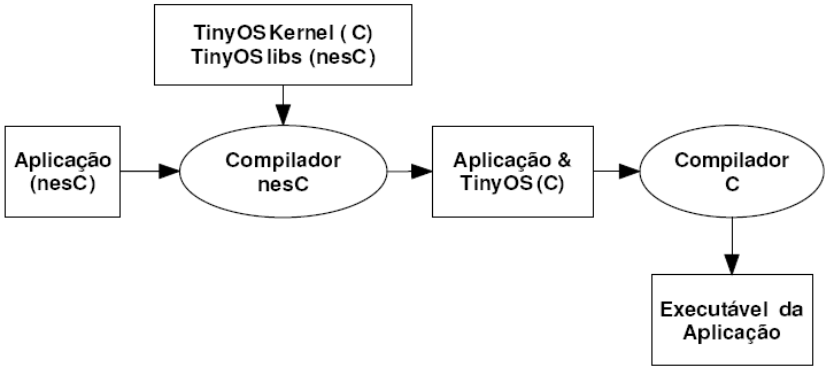


Figura 2.5 – Etapas envolvidas no desenvolvimento de uma aplicação no *TinyOS* [5].

A Figura 2.6 apresenta o fluxo de compilação. Os ficheiros escritos em NesC possuem extensão .nc e quando compilados através do compilador da linguagem NesC, o ncc, geram ficheiros em linguagem C, que possuem extensão .c. Um compilador de linguagem C (no caso o avr-gcc) gera código de máquina (extensão .s) que serve de entrada para um construtor, *builder*, (no caso o avr-as). O construtor irá gerar o código objecto (extensão .o). Por último, o compilador avr- gcc é utilizado para gerar o código executável (extensão .exe). Este executável pode ser descarregado para o *hardware* do Mica Motes no formato S-Records (arquivos com extensão .srec). A conversão do executável para o formato S-Records é feita pelo avr-objcopy.

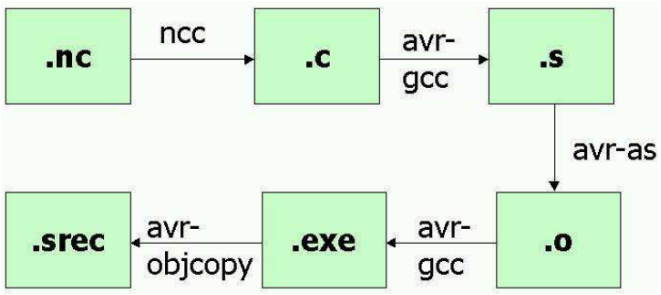


Figura 2.6 – Etapas de compilação [5].

Esclarecido o processo de geração de código, é preciso compreender os conceitos da linguagem NesC, para poder desenvolver aplicações.

A linguagem NesC é uma extensão da linguagem de programação C projectada para incluir os conceitos estruturais e modelos de execução do *TinyOS* [5]. O *TinyOS* é um sistema operativo orientado a eventos e vocacionado para RSSFs cujos nós possuem recursos limitados (por exemplo, 8 kbytes de memória de programa, 512 bytes de RAM). O manual da linguagem, assim como o código fonte, são livres e podem ser encontrados em [7].

Conceitos Básicos do NesC

O NesC permite separar construção de composição [5]. As aplicações escritas em NesC são compostas por componentes, que podem ser construídos e combinados para formar uma aplicação, aumentando a modularidade e reutilização de código. Em NesC, o comportamento de um componente é especificado em termos de um conjunto de interfaces. As interfaces são bi-direcionais, e informam sobre o que um componente usa e sobre o que ele fornece. A Figura 2.7 ilustra um componente (*Timer*) e a sua interface.

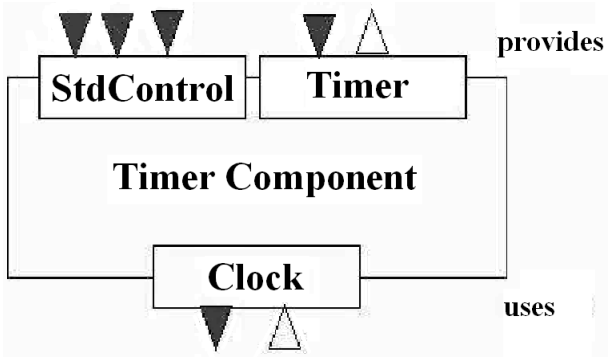


Figura 2.7 – Um componente e a sua interface [5].

Os componentes são ligados estaticamente uns aos outros via interfaces. O fluxo de informação pode ocorrer com camadas inferiores (via comandos) ou com camadas superiores (via eventos), conforme se apresenta na Figura 2.2. Os conceitos do *TinyOS*, tais como eventos, tarefas e comandos, estão embutidos no NesC. Relembrando os pontos importantes destes conceitos são os seguintes:

- **Tarefas**
  - Realizam computação (conjunto de comandos);
  - Não são críticas com relação ao tempo.
- **Eventos**
  - Críticos em relação ao tempo;
  - Sinalizam interrupções externas;
  - Geram um "*Signal*";
  - Receptor recebe/aceita um "*Event*".
- **Comandos**
  - Funções de procedimentos para outros componentes;
  - Não podem sinalizar eventos.

O *TinyOS* fornece outras ferramentas para facilitar o desenvolvimento de aplicações, como simuladores e depuradores (*debuggers*).

2.5 Aplicações

A norma ZigBee foi concebida para aplicações que não necessitam de muita banda mas que sejam económicas, tanto do ponto de vista de energia como do preço dos equipamentos, sendo ideal para controladores e sensores, para a automação de prédios, processos industriais e demais aplicações relacionadas conforme indicado na Figura 2.8 [8].

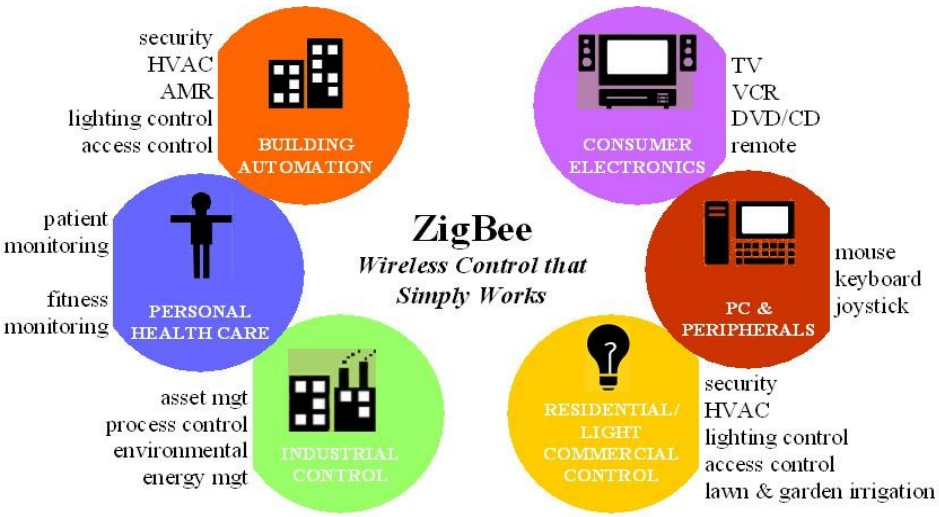


Figura 2.8 – Exemplos de aplicação do padrão ZigBee, extraído de [8].

2.6 Efeitos biológicos da radiação electromagnética

Os telemóveis têm contribuído bastante para a preocupação sobre os efeitos da radiação sobre a saúde do ser humano [9]. Um efeito biológico pode ser caracterizado quando ocorre uma mudança que pode ser percebida por um sistema biológico, após a introdução de um determinado estímulo. As ondas electromagnéticas quando interagem com o corpo humano, dependendo da frequência e da potência, podem produzir algum tipo de efeito biológico. Tal efeito, por si só, não significa necessariamente a existência de um perigo. Um efeito biológico tonar-se-á um risco à segurança quando ocorrer uma problema na saúde da pessoa ou dos seus descendentes.

A radiação das ondas electromagnéticas é classificada em duas categorias: radiação ionizante e não-ionizante. A radiação ionizante corresponde a campos com frequências mais elevadas que as das emissões de luz, como por exemplo, os raios X e os raios gama, cujos comprimentos de ondas são nanométricos Este tipo de radiação possuí energia suficiente para quebrar ligações químicas por ionização Desta forma o material genético das células pode ser danificado, originando doenças como, por exemplo, o cancro.

As células, quando expostas à radiação, sofrem acção de fenómenos físicos, químicos e biológicos. A radiação causa a ionização de átomos e moléculas, que poderão afectar células, que por sua vez podem afectar os tecidos, que poderão afectar os órgãos e, portanto, poderão afectar o corpo todo.

Um organismo complexo como o corpo humano, constituído por cerca de 5 triliões de células, quando exposto a radiações, sofre determinados efeitos somáticos que são restritos ao próprio corpo, mas também efeitos genéticos que são transmissíveis às gerações seguintes.

Os fenómenos físicos responsáveis pela partilha da energia entre as células devido à irradiação são a Ionização e a excitação dos átomos.

Os fenómenos químicos surgem em seguida e provocam a ruptura das ligações entre os átomos ou moléculas, formando-se radicais livres num curto intervalo de tempo que por sua vez são responsáveis por alguns fenómenos biológicos que possam surgir. Estes últimos alteram as funções específicas das células e são responsáveis pela diminuição da actividade da substância viva do organismo. Poderíamos especificar os músculos que perderiam algumas propriedades características como sendo as primeiras reacções do organismo às radiações.

Os efeitos biológicos caracterizam-se, também, pelas variações morfológicas, que são alterações em certas funções essenciais da célula ou então a morte imediata da célula.

Nem todas as células vivas têm a mesma sensibilidade à radiação. As células que têm mais actividade são mais sensíveis, pois a divisão celular requer que o DNA seja correctamente reproduzido para que a nova célula possa sobreviver.

Em frequências mais baixas que as de emissão de luz, incluindo as faixas de macro-ondas, os campos electromagnéticos não possuem energia suficiente (10eV) para provocar a quebra das ligações químicas como no caso anterior e, portanto, esta irradiação é chamada de não-ionizante.

2.6.1 Mecanismos de interacção dos campos electromagnéticos

A maioria dos efeitos biológicos conhecidos está associada aos efeitos térmicos como resultado da interacção dos campos electromagnéticos [9]. Este processo de interacção é mostrado na Figura 2.9.

Os efeitos térmicos são aqueles cujas alterações são cansadas pelo aquecimento do organismo como consequência da absorção de parte da onda incidente. Os efeitos não térmicos são devidos à interacção directa do campo electromagnético com o organismo.

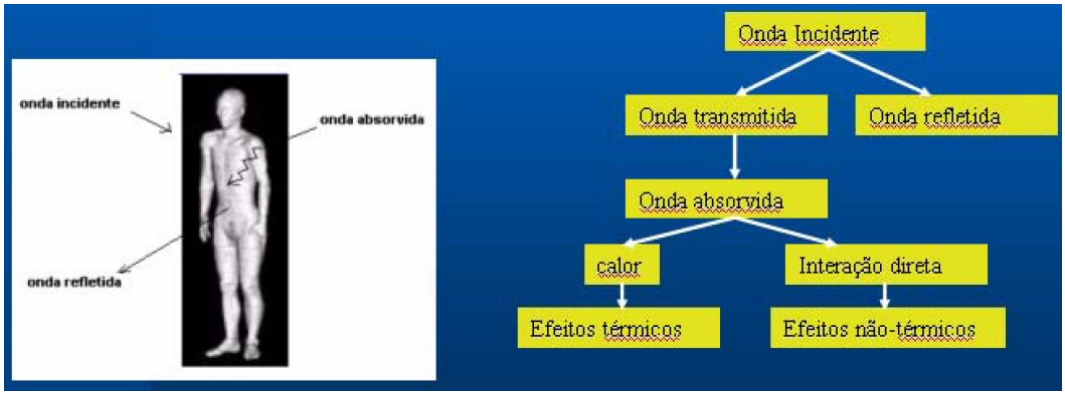


Figura 2.9 – Processo de interacção das ondas electromagnéticas com o corpo humano.

A Tabela 2.2 apresenta a relação entre a frequência das ondas e os principais efeitos biológicos, em função da penetração das ondas electromagnéticas no tecido humano.

A partir das características eléctricas e geométricas do corpo irradiado e das condições de exposição é possível, em princípio, calcular o campo resultante internamente e a taxa através da qual a energia é absorvida pelo tecido que recebe a radiação (Taxa de Absorção Especifica — SAR).

Tabela 2.2 - Frequências e os principais efeitos biológicos

Frequência [MHz]	**Comprimento de Onda [cm]**	**Local de maior efeito**	**Principal Efeito Biológico**
> 10 000	< 3	Pele	A superfície da pele age como reflector ou como absorvente, permitindo que surja o efeito de aquecimento.
10 000	3	Pele	Aquecimento da pele com sensação de calor
10 000 a 3 000	3 a 30	Camadas superficiais, a pele e lentes dos olhos	Lentes dos olhos e testículos são particularmente sensíveis
10 000 a 1 000		Lentes dos olhos	Formação de cataratas e danos nos testículos
1 200 a 150	25 a 200	Órgãos internos	Prejuízos aos órgãos internos por sobreaquecimento
< 150	> 200		É indiferente para o corpo.

Particularmente em altas-frequências (na faixa das microondas), a condutividade é alta, já que o corpo possui grande quantidade de água, que apresenta ressonância naquela faixa.

O aquecimento resulta da absorção do campo electromagnético em um meio dissipado, onde parte da potência é reflectida pela pele e parte penetra dissipando-se rapidamente com a profundidade. A absorção acontece principalmente em função do movimento dos dipólos da água e dos iões dissolvidos. A proporção de água é, portanto, um parâmetro importante quando se determinam as propriedades dieléctricas do tecido. Os valores da condutividade para partes ou para densidades específicas do corpo são dadas para diferentes frequências.

A taxa com que o corpo humano absorve energia electromagnética varia com a frequência sendo relativamente baixa para a gama de frequências dos telemóveis.

Entretanto, a resposta térmica do corpo humano á radiação depende de alguns factores tais como:

- Taxa de absorção especifica (SAR), definida como sendo a taxa de absorção de energia por unidade de massa do objecto exposto. É uma medida de energia absorvida que pode ou não ser dissipada em forma de calor;
- Cobertura do corpo (roupas);
- Sistema termo-regulatório;
- Condição fisiológica;
- Meio ambiente;
- Parte específica do corpo e a quantidade de circulação sanguínea existente nessa região.

Em situações normais, os vasos sanguíneos dilatam-se e o aquecimento é reduzido e ou removido pela corrente sanguínea Desta forma o principal risco de dano térmico concentra-se nas áreas de baixa circulação sanguínea, como por exemplo, os olhos e a têmpora. Os olhos são considerados órgãos bastante críticos com relação ao efeito das radiações não-ionizantes, sendo bastante susceptíveis ao efeito térmico. Quantidades relativamente pequenas de energia electromagnética podem elevar a temperatura das lentes oculares, pelo fato destas não possuírem sistema vascular adequado para as trocas térmicas, o que reduz sua capacidade de dissipação de calor. Por isso, a possibilidade de danos nos olhos constitui um aspecto muito sério na utilização de radiações com microondas ou radiofrequência. O cristalino (lente interna do olho) está sujeito a alterações provocadas por radiações electromagnéticas, pois apresenta algumas características que o tornam particularmente sensível a esse tipo de energia, tais como:

- Posição superficial em relação ao corpo;
- Envolvimento em meio aquoso;
- Circulação sanguínea reduzida;
- Células germinárias estão essencialmente situadas na região superficial equatorial.

É portanto mais provável danificar um tecido, nas áreas onde ocorre um maior aumento da temperatura relativamente ao aumento proporcional da radiação. Essa elevação térmica do cristalino pode levar à sua opacificação, conhecida como catarata.

Efeito como este é de particular interesse para os operadores e utilizadores de transmissores portáteis em que as antenas normalmente ficam próximas aos olhos. Alguns médicos investigadores concordam que a catarata poderia surgir devido ao aquecimento se o olho estiver sujeito continuamente a uma taxa de absorção específica (SAR) de 100 W/kg durante um período mínimo de 100 minutos.

Outros efeitos térmicos pesquisados e conhecidos, tais como queimaduras externas ou internas, exaustão e choque térmico, ocorrem apenas sob exposições de alta intensidade.

Os utilizadores de telemóveis são submetidos a uma SAR cujos valores estão abaixo do especificado pelas normas de segurança, seja quanto à radiação emitida pelos aparelhos ou pelas Estações de Base (EBs). A radiação emitida em ambos os casos tem provocado inúmeras questões quanto aos possíveis efeitos sobre a saúde da população. Primeiro em relação aos efeitos locais da radiação sobre a cabeça de um utilizador com o seu terminal portátil em conversação e em seguida os efeitos sobre o corpo todo de qualquer pessoa que esteja no raio de cobertura da EB, principalmente na direcção da radiação do lóbulo principal da antena.

Está comprovado que, na grande maioria dos casos, a taxa de absorção para os utilizadores de telemóveis é bem maior que a taxa para as pessoas que se situam próximas das EB. Isto explica o porquê de existir um número maior de estudos em relação aos utilizadores de telemóveis.

2.6.2 Efeitos não térmicos

Além dos efeitos térmicos existem ainda outros efeitos que podem ser caracterizados como fisiológicos e comportamentais [9]. Existem uma série de estudos, muitos deles controversos, que destacam o risco efectivo dos níveis de radiação emitidos pelos telemóveis.

O ponto central de tais pesquisas é o seguinte: a exposição a microondas, mesmo com baixas intensidades, poder resultar em distúrbios nervosos, dos quais podemos exemplificar as dores de cabeça, fadiga, tontura, perda de memória e insónias. Embora não existam evidências científicas que relacionem estes efeitos com as radiações emitidas pelas EBs, cujos níveis radiados são bastante baixos, poderá existir risco para a saúde humana [9]. Entretanto, surgiram estudos que direccionam a dor de cabeça como sendo um efeito da utilização de telemóveis, sugerindo uma mudança comportamental ou fisiológica, muito embora ainda não esteja confirmada cientificamente. Segundo

Moulder existe a probabilidade, mesmo que pequena, de surgirem alguns efeitos tais como:

- Aumento da pressão sanguínea;
- Alteração da actividade eléctrica do cérebro;
- Pequenas mudanças no padrão de sono;
- Pequenas alterações em exames de eletroencéfalogramas.

Recentemente, um estudo feito por N. Edelstyn e A. Oldershaw mostrou que as pessoas melhoraram o desempenho em avaliações de concentração quando submetidas à radiação de 900MHz. De forma controversa, e sob as mesmas condições, observou-se que nos ratos o efeito foi o contrário, ou seja, ocorreu uma perda de memória a longo prazo e também uma redução no tempo de reacção. Segundo J. Gonzalez não houve qualquer indicativo do efeito sobre esse processo para a frequência em questão e com uma SAR a variar entre 1 e 3,5 W/kg por um período de exposição diário de 45 minutos sobre a cabeça dos mesmos.

O investigador norte-americano W. Ross Adlev tem dedicado os seus estudos, há mais de 20 anos, à relação entre os efeitos da modulação e os efeitos biológicos. Numa das suas pesquisas ele verificou que a radiação em amplitude modulada (AM) nas faixas de frequências de 150 e 450 MHz poderia produzir mudanças na percepção de cálcio em células do sistema nervoso. Trabalhos posteriores, de outros autores, refutaram tais conclusões.

O biofísico G.J. Hyland é um especialista sobre o mecanismo de interacção da radiação não-ionizante com organismos vivos e é um critico severo em relação aos níveis recomendados nas normas de segurança adoptadas mundialmente, considerando que estas se encontram demasiado altas. Critica também a filosofia de desenvolvimento dessas normas, que são principalmente baseadas nos efeitos térmicos e na premissa de que os efeitos não térmicos, indicados anteriormente, ainda carecem de fundamentação científica. É ainda defensor, da teoria de que a modulação seria por si só responsável por alguns efeitos fisiológicos e comportamentais podendo ocasionar danos ao funcionamento dos sistemas nervosos e imunitário dos organismos vivos. As considerações sobre tais efeitos da modulação empregada nos telemóveis têm sido mencionadas como uma das evidências do perigo das EB.

Entretanto H. Lai, que há já algum tempo é defensor da importância dos efeitos não térmicos, de acordo com as suas pesquisas com ratos e células, mostra uma relação

entre estes efeitos e o sistema nervoso, quando exposto a radiofrequências nos níveis limites das directrizes de segurança. Tais efeitos são contestados por outros especialistas.

Recentemente, noutro estudo, os especialistas tentaram perceber quais os efeitos da radiação nos materiais utilizados em implantes intracranianos para tratamento do aneurisma. Foi utilizado um filamento de tungsténio e um de platina para reprodução da experiência, tendo sido ambos expostos a ondas electromagnéticas de 900MHz por um período de 99 horas com vista a reproduzir a exposição real de uma pessoa que utiliza regularmente um telemóvel com tecnologia GSM, durante alguns meses. Após as observações e medidas, concluiu-se que o filamento de platina não sofreu qualquer dano, sendo portanto mais adequado aos implantes deste tipo.

Outro efeito bastante discutido é o efeito da radiação nos testículos, já que estes são considerados órgãos críticos no que diz respeito aos efeitos das radiações electromagnéticas visto serem extremamente sensíveis ás elevações de temperatura, ficando mais sujeitos à radiação devido à:

- Localização superficial em relação ao corpo;
- Grande sensibilidade ao calor por parte das células germinativas que se encontram numa faixa de temperatura inferior à temperatura corporal (aproximadamente 33°C), apresentando uma velocidade de redução celular, já a temperaturas de 37°C.

Pesquisas com cães, coelhos e ratos, para determinar o limite a partir do qual surgem efeitos prejudiciais, mostraram que a 10 mW/cm^2 de densidade de potência, os efeitos patológicos nos testículos incluem degeneração do epitélio que reveste os tubos seminíferos e uma acentuada redução da maturação dos espermacitócitos. Essa redução da função testicular é devida ao aquecimento, e tudo indica que seja temporária e provavelmente reversível.

2.7 Sumário e Conclusões

As RSSFs possuem um vasto campo de aplicações existindo cada vez mais programas que facilitam a sua utilização, tanto para instalação nos nós sensores como para simulação das RSSFs, ou mesmo para conversão entre ambos, existindo um vasto leque de projectos em curso a nível mundial.

Neste Capítulo começou-se por abordar brevemente o TinyOS e o NesC.

Em relação aos efeitos da radiação no corpo humano é de notar que a radiação

acima de certos valores tem efeitos negativos no corpo humano, tal como o aquecimento de certas zonas do corpo, não existindo porém estudos que comprovem, sem margem para dúvidas, alguns dos efeitos secundários nefastos que se acreditam serem causados pela proximidade do corpo humano com fontes emissoras de radiação electromagnética.

Capítulo 3
IEEE 802.15.4

3.1 Introdução

Estando as RSSFs no seu inicio devem-se fazer apostas acertadas na escolha do rumo e das características que se pretende atingir. Foi efectuada uma pesquisa em 2002 acerca das características que seriam mais importantes para os utilizadores de sensores sem fios [8]. Deste estudo resultou que essas características são as seguintes, em ordem decrescente de importância:

1. Fiabilidade dos dados;
2. Vida útil da bateria;
3. Custo;
4. Alcance da transmissão;
5. Taxa de transferência dos dados;
6. Latência dos dados;
7. Tamanho Físico;
8. Segurança dos dados.

É de prever que no futuro as máquinas possam comunicar umas com as outras, por exemplo, com o telemóvel poder-se-á abrir a porta da garagem ou acender uma lâmpada ou com um sensor de leitura instalado numa carrinha poder-se-á registar o consumo de energia eléctrica das casas, à medida que a carrinha avança pela respectiva rua.

Para que esta comunicação seja possível os instrumentos terão de ser compatíveis, ou seja, devem utilizar uma linguagem própria (protocolo) de forma a poderem compreender-se entre si.

A norma IEEE 802.15.4 foi divulgada em Maio de 2003 e tem em vista aplicações sem fios para equipamentos que não necessitem de um elevado ritmo de transmissão, mas que necessitam de uma latência reduzida e de um baixo consumo de energia reduzido, Figura 3.1.

A norma IEEE 802.15.4 estabelece um protocolo simples de transmissão de dados por pacotes para redes sem fios. O canal de acesso ao meio é via *Carrier Sense Multiple Access collision avoidance* (CSMA-CA) e com *time slotting* opcional. Possui reconhecimento de mensagem e uma estrutura sinalizadora, chamada *beacon*. A segurança é feita por multi-camadas.

É utilizado em dispositivos que necessitam de baterias com longa duração, baixa latência para controladores, sensores, monitorização remota e dispositivos electrónicos portáteis.

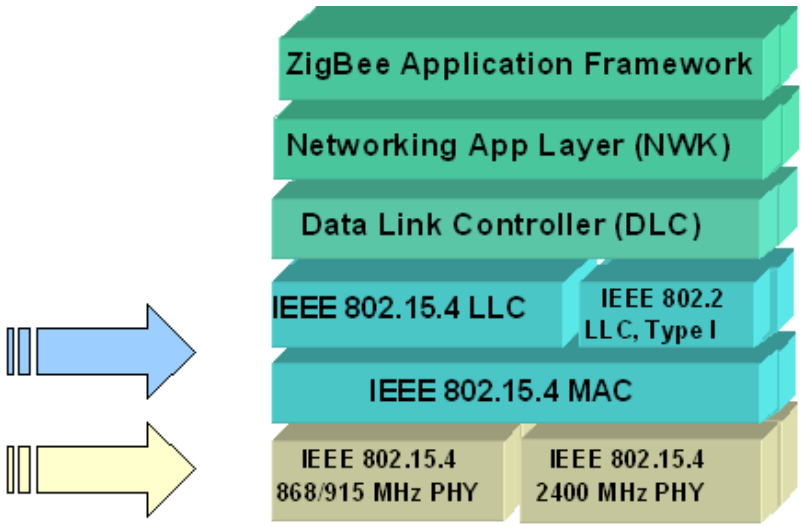


Figura 3.1 – Camadas do protocolo IEEE 802.15.4, extraído de [8].

3.2 Norma

3.2.1 Acesso ao meio

O acesso ao meio pode ser feito de duas formas [8]:

Non-beacon network - Para pacotes recebidos com sucesso o CSMA-CA contem mensagens positivas de *acknowledgement* (ack). O *time sloting* é opcional.

Beacon-enable network - Neste caso, utiliza-se o conceito de *superframe*. Desta forma, tem-se uma banda passante dedicada para um determinado dispositivo. O *Beacon-enable network* é alterado pelo *network coodinator*, descrito na secção seguinte, para envios de *beacon* em intervalos predeterminados. Existem 16 intervalos entre os *beacons*.

3.2.2 Tipos de Dispositivos

Existem 3 tipos de dispositivos numa rede IEEE 802.14.5:

Network Coordinator - Mantém um conhecimento completo da rede. É o mais sofisticado entre os três tipos, sendo por isso o que ocupa mais memória e processamento. Permite coordenar a rede, escolher a forma de acesso ao meio (se é com ou sem *beacon*) e a entrada de novos nós na rede.

Full Function Device **(FFD)** - Contém toda a funcionalidade do 802.15.4. Possui memória adicional, e devido ao seu processamento computacional é ideal para realizar funções de mapeamento da rede. Pode também ser utilizado nas fronteiras da rede ligando-as ao mundo real.

Reduced Function Device **(RFD)** - Possui funções limitadas para controlar custos e complexidades do tráfego da rede. Não faz mapeamento sendo por isso utilizados nas fronteiras das redes.

3.3 Estrutura das Tramas

Nesta secção apresentam-se algumas das estruturas das tramas da camada MAC.

3.3.1 Data Frame Format

Esta estrutura é uma das mais básicas e importantes da norma IEEE 802.15.4; Figura 3.2. Suporta uma capacidade de transferência até 104 bytes por pacote, sendo esse valor suficiente para transferência de informações entre sensores. Para garantir a fiabilidade na entrega dos dados contém um campo com uma numeração sequencial dos dados e um campo de *Frame Check Sequence* (FCS).

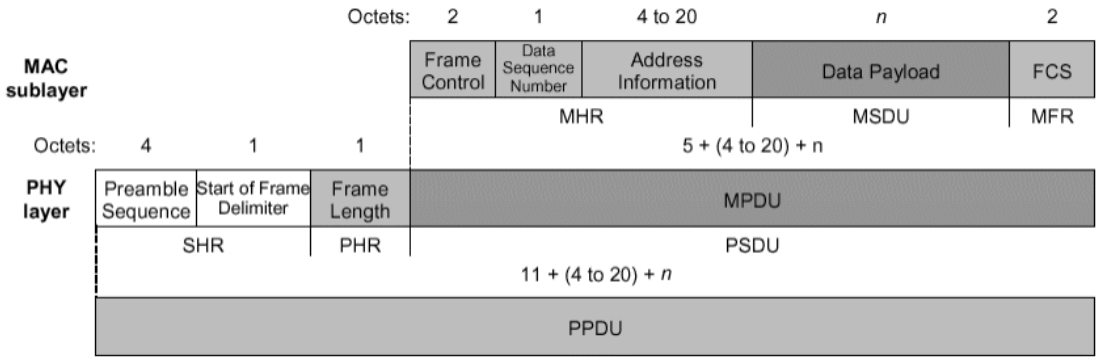


Figura 3.2 – Estrutura *Data Frame Format,* extraído de [8].

3.3.2 Acknowledgement Frame Format

É enviado uma mensagem de retorno (ACK) do receptor para o emissor informando-o de que o pacote foi recebido sem erros, Figura 3.3.

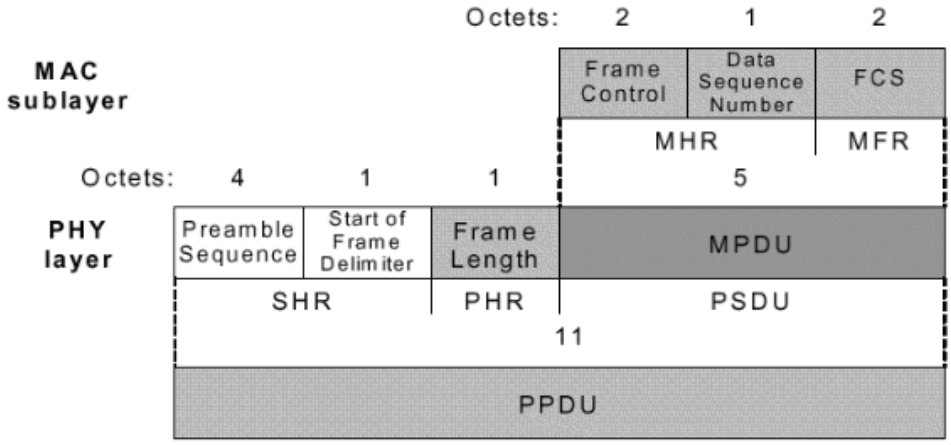


Figura 3.3 – Estrutura *Acknowledgement Frame Format,* extraído de [8] .

3.3.3 MAC Command Frame Format

Representa um mecanismo para controlo e configuração remota dos nós da rede, permitindo que uma rede centralizada configure clientes individualmente não importando a extensão da rede, Figura 3.4.

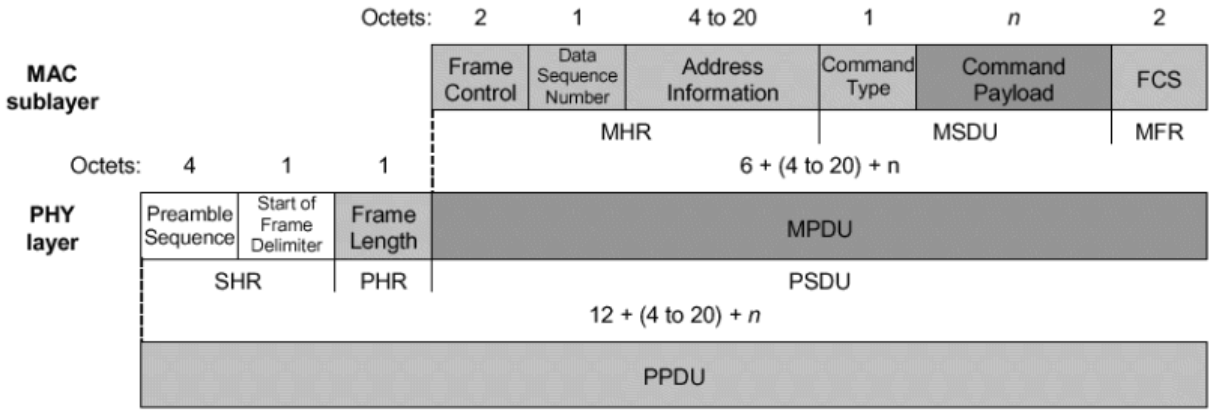


Figura 3.4 – Estrutura MAC Command Frame Format, extraído de [8].

3.3.4 Beacon Frame Format

É uma estrutura opcional de sinalização. Os dispositivos podem acordar somente quando este sinal é transmitido; caso contrário, retornam ao estado de repouso. É utilizado nas topologias *mesh* e estrela estendida para manter os nós sincronizados sem a necessidade deles consumirem energia por longos períodos de tempo, Figura 3.5.

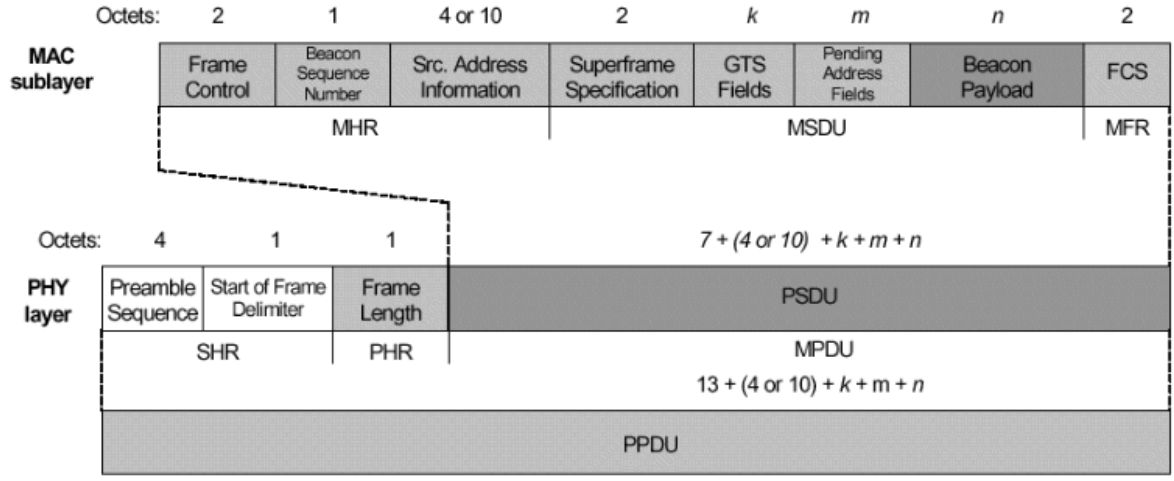


Figura 3.5 – Estrutura *Beacon Frame Format*, extraído de [8].

3.3.5 Superframe Format

Outra estrutura opcional da camada MAC é a *superframe.* É definida pelo nó coordenador, Figura 3.6, utiliza *beacons* para sinalização, *slotting time* e *Garanty Time Slotting* (GTS) para permitir que um nó continue utilizando o meio de acesso caso esteja mandando uma mensagem com prioridade alta.

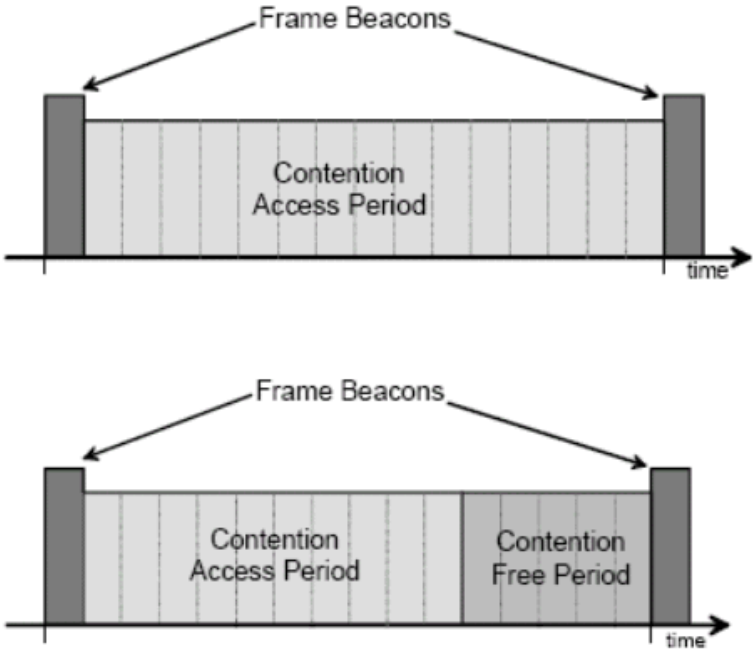


Figura 3.6 – Estrutura *Superframe Format*.

3.4 Protocolos da camada de ligação

3.4.1 Protocolo Sensor – Medium Access Control

O *Sensor–Medium Access Control* (S-MAC) é um protocolo de controlo de acesso ao meio baseado na atribuição dinâmica do canal, mas que utiliza sincronização para coordenação dos modos de operação do rádio [10]. É destinado a redes com aplicações orientadas a eventos, com recolha periódica de dados, insensível à latência e com baixa taxa de envio de mensagens. A comunicação entre os nós segue um fluxo de difusão ou um fluxo *unicast* para troca de mensagens. Considera-se os requisitos de uma rede densa e homogénea para ser eficiente em energia e permitir a auto-configuração dos nós da rede. O protocolo S-MAC procura ser eficiente em termos energéticos reduzindo o consumo dos principais eventos responsáveis pelo desperdício de energia, descritos a seguir:

- **Colisões** – os nós desejam transmitir ao mesmo tempo para um mesmo destino. Para resolver o problema de colisão o S-MAC emprega a mesma técnica utilizada no IEEE 802.11 – *Distributed Coordination Function* (DCF), usando um diálogo de comunicação *Request-To-Send-Clear-To-Send-Data-A*cknowledge (RTS-CTS-DATA-ACK). Este diálogo de comunicação evita colisões, problemas de terminal escondido e o problema

da estação exposta. Caso ocorra a colisão utiliza um algoritmo para aguardar um tempo aleatório, o *Binary Exponential Backoff* (BEB).

- ***Overhearing*** – os nós escutam transmissões de pacotes destinados a outros nós. A técnica empregada pelo S-MAC é desligar o rádio do nó (modo *sleep*) quando verifica que o pacote não é destinado a ele, Figura 3.7.
- ***Overhead*** – os pacotes de controlo são utilizados para reserva do canal de comunicação, reconhecimento de pacotes de dados, sincronização e outros. Estes pacotes de controlo aumentam o tráfego da rede, mas não transportam dados úteis. O S-MAC reduz o tamanho dos pacotes de controlo para diminuir o *overhead*.
- ***Idle listening*** – o nó escuta o meio mesmo quando não existe tráfego na rede. O S-MAC utiliza um ciclo de operação reduzida, com tempos fixos de actividade (*listen*) e de repouso (*sleep*), Figura 3.7. O tempo de actividade é menor que o tempo de repouso (será cerca de 10%).

A sinalização para os pacotes de controlo e de sincronização é feita dentro do canal, difundindo um pacote de sincronização (SYNC) para todos os seus vizinhos. O S-MAC aplica a técnica de *message passing* para reduzir a latência durante a contenção em aplicações que requerem armazenamento de informações para processamento na rede (*in-network*). Esta técnica permite a transmissão de mensagens longas, que são divididas em pequenos fragmentos e enviadas em rajada. Este protocolo obtém considerável redução do consumo de energia, prolonga o tempo de vida da rede e encontra-se já implementado na plataforma Mica Motes da *Crossbow*.

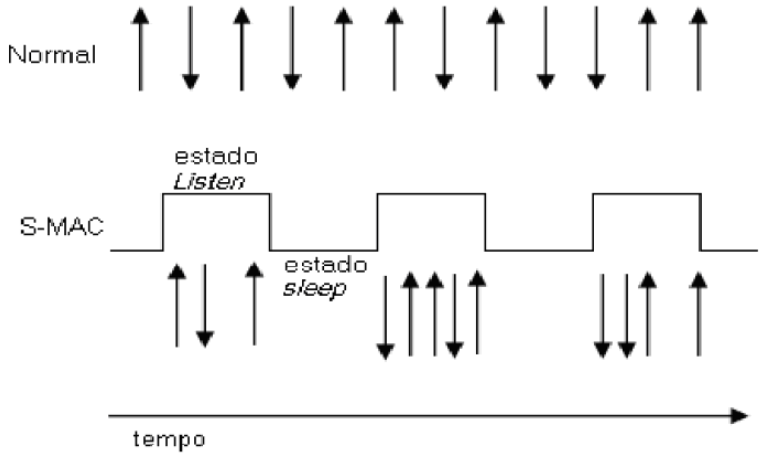


Figura 3.7 – Protocolo S-MAC.

3.4.2 Protocolo Adaptive Rate Control

O *Adaptive Rate Control* (ARC) tem como metas atribuir largura de banda e ser justo e eficiente energeticamente para condições de tráfego na rede tanto elevadas como baixas. O protocolo propõe um mecanismo que passivamente adapta a taxa de transmissão aos dois tipos de tráfego: transmissão de dados passagem e transmissão de dados originados localmente no dispositivo [10]. Este mecanismo usa um incremento linear e um decremento multiplicativo para controlar a taxa de transmissão de dados. A capacidade computacional exigida aos nós por este esquema é pequena e está dentro das limitações de *hardware*. No contexto de ARC propõe-se um novo esquema CSMA, adicionando-se um atraso aleatório antes do tempo de escuta, para evitar colisões repetidas devido ao comportamento sincronizado do nó na ocorrência de um evento. Este esquema CSMA é composto pelas seguintes fases: atraso inicial aleatório, tempo de escuta – intervalo de tempo fixo, e mecanismo de *backoff* – tempo de atraso gerado com janela fixa, com incremento binário exponencial ou com decremento binário exponencial.

O ARC em conjunto com este novo mecanismo CSMA fornece controlo efectivo de acesso ao meio sem a utilização de pacotes explícitos de controlo. Este esquema é justo e consegue uma largura de banda razoável, sendo eficiente do ponto de vista energético para situações de baixo tráfego.

3.4.3 Protocolo Time-out-Medium Access Control

O protocolo *Time-out-Medium Access Control* (T-MAC) é baseado em contenção para o controlo de acesso ao meio em RSSFs [10]. O T-MAC foi desenvolvido para aplicações dirigidas a eventos que possuem baixa taxa de entrega de mensagens, insensíveis a latência e com transmissão contínua ou periódica de dados. A meta do T-MAC é ser eficiente do ponto de vista energético, considerando as limitações do *hardware* do nó e os padrões de comunicação de troca de mensagens entre os seus vizinhos e entre os nós e a estação base.

O ciclo de operação é reduzido e possui tempos de actividade (*listen*) e de repouso (*sleep*) variáveis que se adaptam à carga da rede, Figura 3.8. A variação dinâmica do tempo activo é obtida pela implementação de um temporizador que desliga o rádio do nó ao verificar que não existe transmissão durante um intervalo de tempo.

O T-MAC pretende reduzir o tempo de *idle listening* para diminuir o consumo de energia do nó. As mensagens recebidas durante o tempo de repouso são armazenadas e transferidas em rajadas no início do tempo activo, Figura 3.8.

O nó escuta a rede, transmite e recebe dados durante o tempo activo. O temporizador determina o final do tempo activo quando não ocorrem eventos durante um tempo T_A, Figura 3.8. A activação por eventos ocorre por: início periódico de quadro, recepção de dados no rádio, final da transmissão dos seus vizinhos, final da transmissão do seu próprio pacote de dados ou recepção de ACK, ou por detecção de sinal no rádio *Received Signal Strenght Indicator* (RSSI). Os nós comunicam-se com o diálogo RTS-CTS-DATA-ACK para evitar colisões e obter uma transmissão de confiança. De maneira semelhante ao S-MAC, o T-MAC utiliza agrupamentos virtuais que seguem escalas para sincronizar o seu ciclo de operação. Os nós transmitem as suas escalas para os seus nós vizinhos através de pacotes SYNC.

A recepção de pacotes *Request-To-Send* (RTS) ou *Clear To Send* (CTS) é suficiente para renovar o tempo T_A, indicado na Figura 3.8. O intervalo de tempo T_A deve ser suficiente para receber pelo menos o início de um pacote CTS, sendo obtido por:

$$T_A > t_{contenção} + t_{RTS} + RTT_{RTS} \tag{3.1}$$

Em que $t_{contenção}$ é o intervalo do tempo de contenção, t_{RTS} é o tempo do pacote de RTS e RTT_{RTS} é o tempo de transmissão de um pacote RTS (ida e volta).

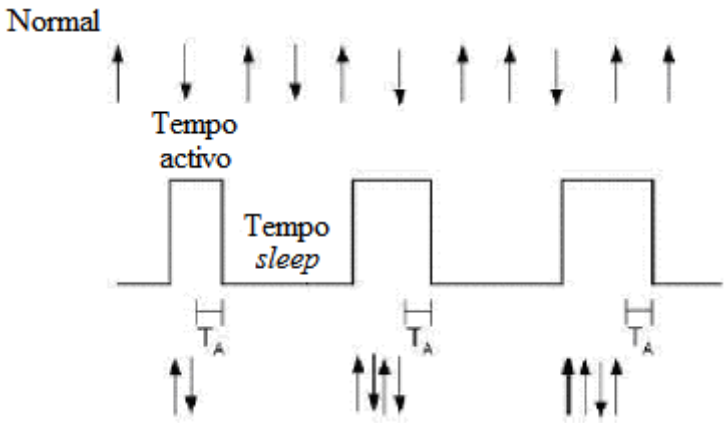


Figura 3.8 – Protocolo T-MAC

O mecanismo de *backoff* é baseado num número aleatório de intervalos fixos, calculados em função da carga máxima. Indiferentemente de sucesso ou falha na comunicação, a janela de contenção não é incrementada. O T-MAC tem o problema de um nó poder entrar no estado de repouso enquanto outro nó ainda tem uma mensagem para ele. Este é conhecido como sendo o problema de adormecer cedo demais (*early sleeping*), Figura 3.9 (onde o nó D entra no estado de repouso antes de C enviar um RTS).

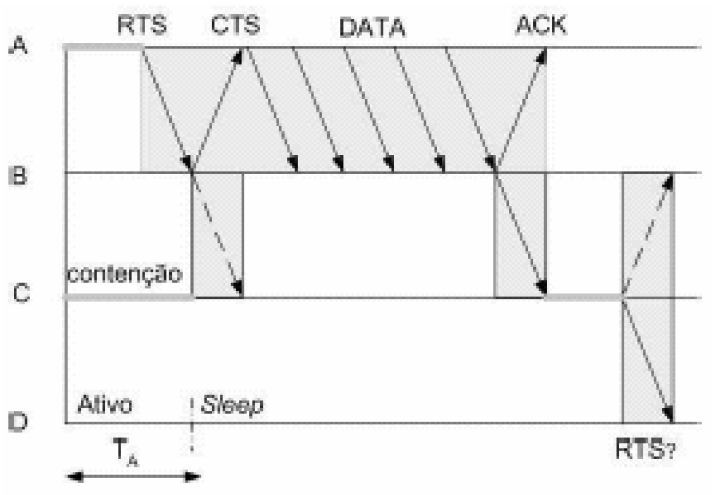


Figura 3.9 – O nó D entra em repouso antes de C enviar um RTS (*early sleeping).*

Este problema pode ser resolvido de duas maneiras:

- Um nó ao escutar um pacote CTS destinado a outro nó envia imediatamente aos seus vizinhos um pacote designado de *Future RTS* (FRTS);
- Um nó pode usar um esquema que dê prioridade ao esvaziar do *buffer* quando este estiver perto do seu limite. Quando um nó recebe um RTS em vez de responder com um CTS, transmite as mensagens armazenadas no seu *buffer* para o nó de destino. O protocolo T-MAC consegue ser mais eficiente do ponto de vista energético que o S-MAC, mas é extremamente limitado em largura de banda e o seu algoritmo não é aplicável depois de parte da largura da banda do canal ser utilizada.

3.4.4 Protocolo *Backoff-Medium Access Control*

Este protocolo B*ackoff-Medium Access Control* (B-MAC) foi projectado especificamente para RSSF e utiliza como plataforma de desenvolvimento os Mica2

Motes [10]. Encontra-se implementado na versão do *TinyOS* 1.1.3 como um novo método de CSMA/CA para RSSF.

O princípio do protocolo B-MAC é o seguinte: em vez de inserir o algoritmo de *backoff* inicial dentro da camada MAC, é estabelecida uma política de gestão em que a aplicação controla o *backoff* inicial antes de submeter um pacote para transmissão. Neste caso o algoritmo de *backoff* binário exponencial não é usado para o controlo de colisões, em vez disso verifica-se o estado do canal. O B-MAC utiliza a heurística chamada *Clear Channel Assessment* (CCA) para verificar se existe actividade no canal e para retornar a informação para a aplicação. O CCA utiliza a técnica de julgamento do canal baseado-se numa estimativa de ruído do canal obtida pela potência do sinal recebido *Received Signal Strenght Indicator* (RSSI). Na implementação do B-MAC (*TinyOS* versão 1.1.3), o tamanho do preâmbulo das mensagens foi reduzido e o limite teórico do canal foi aumentado de 42 pacotes/segundo para 53 pacotes/segundo, para o caso de o pacote ser de 36 bytes. O B-MAC é um novo método CSMA para RSSF que encontra razoável aceitação em comparação aos métodos tradicionais e proporciona uma boa taxa de utilização do canal, flexível para diferentes aplicações.

3.4.5 Protocolo *Distributed Energy-Medium Access Control*

O protocolo *Distributed Energy-Medium Access Control* (DE-MAC) trata da gestão e balanço de energia em RSSF [10]. É um protocolo que emprega atribuição estática de canal TDMA, portanto livre de colisões e de *overhead* de pacotes de controlo.

Utiliza o conceito de ciclo de operação reduzido com tempos de actividade e de repouso para evitar o desperdício de energia com a escuta de pacotes destinados a outros nós (*overhearing*) e com a escuta do meio sem tráfego (*idle listening*).

O DE-MAC utiliza um algoritmo distribuído para balanço da carga na rede. Este algoritmo estabelece que os nós com baixa energia devem ser usados com menor frequência no encaminhamento e para isso realiza um procedimento local de eleição. A eleição é usada para escolher o nó com energia mais baixa de todos os nós da rede. O nó eleito ficará mais tempo em repouso (modo *sleep)* em relação aos nós vizinhos. Como o protocolo é baseado em TDMA e a eleição é totalmente integrada com o tempo atribuído para cada nó *(slot* TDMA) o protocolo não diminui o escoamento da rede. O

DE-MAC assume a sincronização dos pacotes TDMA, e o método de "eleição" dos nós com a energia mínima garante uma distribuição equilibrada da energia na rede.

3.4.6 Protocolo *Traffic adaptive Multiple Access*

O protocolo *Traffic adaptive Multiple Access* (TRAMA) é baseado em atribuição estática de canal TDMA [10]. É projectado para aplicações dirigidas a eventos com recolha contínua ou periódica de dados em RSSFs. A meta principal deste protocolo é ser eficiente em termos energéticos e o método de acesso ao canal garante que não existirão colisões em comunicações *unicast*, *broadcast* (difusão) ou *multicast*.

O TRAMA adapta-se ao tipo de tráfego e emprega um algoritmo distribuído de eleição. Este algoritmo determina qual o nó que pode transmitir em determinado intervalo de tempo (*time-slot*), e não faz reserva para os nós sem tráfego de envio. O algoritmo de eleição é baseado em informações de tráfego de cada nó e selecciona receptores baseados em escalas anunciadas pelos emissores. As escalas são obtidas pela troca de informações locais da sua vizinhança de dois *hops* e são transmitidas para especificar quais os nós que serão os respectivos receptores do seu tráfego por ordem cronológica. O TRAMA alterna entre acessos aleatórios e escalonados para acomodar mudanças de topologia, permitindo a adição de nós na rede e tolerância a falhas.

Consiste em três componentes:

- ***Neighbor Protocol* (NP)** – é responsável pela propagação e actualização de informações sobre os vizinhos de um salto (*hop*). As actualizações são incrementais e permitem determinar o conjunto de vizinhos que serão adicionados ou removidos.
- ***Schedule Exchange Protocol* (SEP)** – permite que os nós troquem informações e escalas da vizinhança de dois *hops*.
- ***Adaptive Election Algorithm* (AEA)** – utiliza as informações da vizinhança e das suas escalas para seleccionar emissores e receptores para o intervalo de tempo actual, enquanto os outros nós seleccionam o modo de repouso.

Apesar da troca de informações entre vizinhos tentar criar uma visão global da rede e o protocolo assumir a sincronização de pacotes TDMA, o TRAMA mostra-se adequada para aplicações insensíveis à latência que necessitam de eficiência energética e uma elevada taxa de entrega.

3.5 Zig Bee Alliance

Os membros da ZigBee Alliance encontram-se a definir os padrões globais de confiabilidade, custos para aplicações sem fios de potência reduzida [8]. A ZigBee Alliance encontra-se com um crescimento rápido, sendo uma empresa sem fins lucrativos que é formada por um consórcio de empresas líderes nos ramos de tecnologia.

A organização foca-se na automação e controlo de residências e de prédios para empresas, em dispositivos electrónicos, periféricos de computador, monitorização médica, brinquedos e na normalização industrial de aplicações que funcionam através do protocolo IEEE 802.15.4.

O conselho de direcção é constituído pelos membros fundadores, existindo actualmente 5 promotores e 1 uma responsável por gerir essa direcção. Também existem empresas que por desejarem fazer contribuições têm acesso prévio ás especificações e contribuem para moldar o Zig Bee, Figura 3.11.

Figura 3.10 – Fundadores.

Figura 3.11 – Participantes.

3.6 Características e Consumo de Energia

3.6.1 Definição da Norma

A ZigBee Alliance define os protocolos das camadas superiores, da camada de rede à camada de aplicação, enquanto que a norma IEEE 802.15.4 define as camadas inferiores: MAC e física [8].

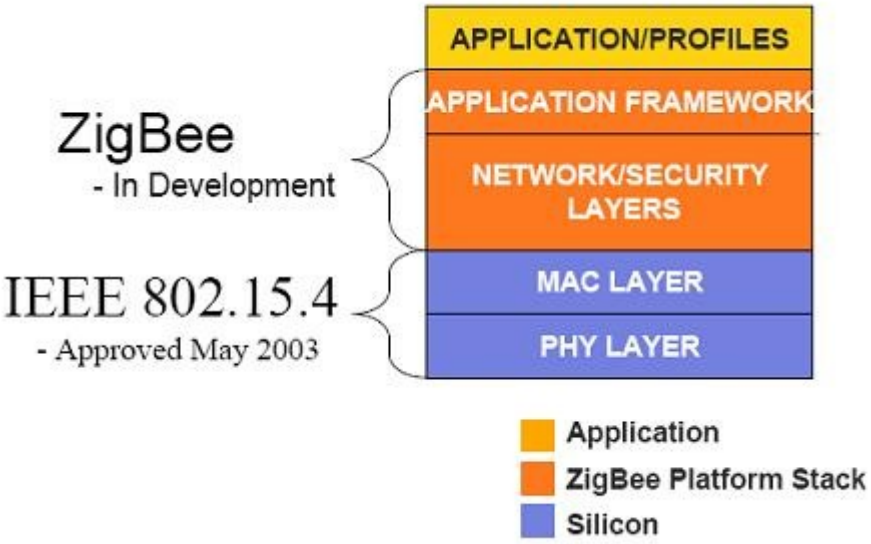


Figura 3.12 – Camadas definidas pela ZigBee Alliance e pela IEEE 802.15.4, adaptado de [8].

3.6.2 Características Básicas da Rede

A rede pode possuir até 64 mil nós, sendo um o nó coordenador [8]. Se existir um novo nó para se juntar à rede, irá demorar a activar-se 30ms. Para um nó adormecido se tornar activo, o tempo de activação será de 15ms.

A entrada destes nós na rede são coordenadas pelo coordenador da PAN que entra em modo de enumeração ou aprendizagem, permitindo que novos nós entrem na rede por um determinado curto período de tempo. Se durante este tempo houver uma requisição para se unir à rede, então o novo nó será capaz de aderir à rede.

A topologia da rede é variada. Pode ser estrela, estrela estendida, *peer-to-peer*. Os nós RDFs não executam mapeamento, portanto ficam nas margens da rede.

3.6.3 Baterias

O protocolo ZigBee foi projectado para suportar aplicações que necessitam de baterias durante um longo período de vida [8]. A vida útil da bateria varia em função da

carga eléctrica e da aplicação em que é utilizada. Como o ZigBee está a ser projectado para controladores e sensores, é de esperar que estes sejam aplicados em ambientes com temperaturas pouco recomendáveis.

3.7 Fiabilidade e Robustez

3.7.1 Fiabilidade

A atmosfera não é um meio de transmissão protegido pois encontra-se susceptível a degradações, dispersões, caminhos múltiplos, interferências, dependência de frequências e outros aspectos de segurança [8].

Em cada camada do modelo ISO há mecanismos de combate a essas degradações para optimizar a transmissão de dados.

- **Camada Física** - *Direct Sequence* com *Frequency Agility* (DS/FA). O *Direct Sequence* usa uma sequência especial de um "chip". Quanto maior a quantidade de chips por símbolos, maior a capacidade de rejeitar caminhos múltiplos ou interferência. A *Frequency Agility*, não confundir com o *frequency hopping* do *bluethooth*, consiste na habilidade de trocar de frequências para evitar interferência de alguma fonte de sinal.
- **Camada de Ligação -** ARQ (*acknowledgement request*) e *Coordinator buffering*. Uma transmissão com sucesso é verificada retornando um acknowledge (ACK). Se o ACK não é recebido, o pacote é retransmitido. Já com Coordinator buffering, o nó coordenador da rede armazena as mensagens para nós adormecidos até eles acordarem novamente.
- **Camada de Rede** - Permite que dados sejam transmitidos por diferentes caminhos.
- **Camada de Aplicação** - *Application Support Sub-layer* (APS) contém uma segurança que mantém afastados outros dispositivos que possam corromper os sinais da rede.

3.7.2 Robustez

O conceito de robustez é muito semelhante à confiabilidade e pode ser definido como a tolerância a um fenómeno degradante no meio físico ao qual um canal está sujeito, como por exemplo os múltiplos caminhos e a interferência [8]. O ZigBee também possui mecanismos de combate a essas degradações tais como o *Frequency hopping* e o DS/FA.

Frequency hopping é um método que permite que periodicamente se troque de canal para evitar utilizar um canal com mau fucionamento. Apesar de esta técnica ser muito eficiente em algumas circunstâncias, ela cria outros problemas tais como latência, incerteza para canais em estado de repouso, perda no resultado do produto de banda x tempo.

Já o *Direct Sequence with Frequency Agility* (DS/FA) combina os melhores recursos do DS e FA sem a maioria dos problemas causados pela interferência porque a troca de frequências não é necessária durante a maior parte do tempo.

3.8 ZigBee vs BlueThooth

3.8.1 Tecnologias concorrentes ou que se complementam?

O Zig Bee e o Bluethooth complementam-se pois cada uma tem um determinado alvo de aplicações [8], Tabela 3.1.

O ZigBee é indicado para a transferência de pequenos pacotes através de grandes redes, que sejam mais estáticas e com pouca frequência de utilização. Permite conectar até 64 mil dispositivos num raio de 30 metros. As suas aplicações estão relacionadas à automação de residências, controlos remotos, monitorização de sensores.

O Bluetooth é indicado para transferência de grandes pacotes através de uma pequena rede do tipo *ad-hoc*, ou seja transferência de arquivos, imagens, áudios, interconectando dispositivos como telemóveis, *headsets*, PDAs. São capazes de ligar até 8 dispositivos numa área de alcance de 10 metros.

3.8.2 Diferenças em relação à interface com o ar

O ZigBee é optimizado para aplicações com tempos críticos, os tempos de acesso são muito pequenos e reflectem a necessidade de se obter respostas rápidas, já que são aplicados em sensores de monitorizam variáveis, contribuindo assim para a optimização e para prolongar a vida das baterias [8].

O tempo de acesso de um nó activo no Bluetooth é bem inferior ao ZigBee uma vez que esse protocolo tem que ser capaz de transferir uma maior quantidade de dados a uma taxa de transferência que não exija muito tempo, Tabela 3.2.

Tabela 3.1 – Cooperação entre Zigbee e o Bluetooth

	ZigBee	Bluetooth
Codificação	DSSS	FSSS
Taxa de transferência	250 Kbits/s	1 Mbits/s
Modulação	O-QPSK	FSK

Tabela 3.2 - Considerações de temporização

Tempo de	ZigBee	Bluetooth
União à rede	30 ms	3 s
Activação de um nó adormecido	15 ms	3 s
Acesso de um nó activo	15 ms	2 ms

3.9 Sumário e Conclusões

A norma IEEE 802.15.4 é bastante robusta e funcional para ser utilizada em RSSFs.

A aliança ZigBee adoptou um sub-conjunto de especificações da norma IEEE 802.15.4 e reprenta actualmente uma boa solução para dispositivos que não necessitam de banda larga, mas apenas de baixo consumo de energia como, por exemplo, sensores e controladores [8].

O ZigBee tem como alvo aplicações ainda não utilizadas pelo Bluetooth ou por outra norma sem fios, ou seja, estas duas tecnologias poderão ser o complemento uma da outra para uma solução em larga escala.

Capítulo 4
Desenvolvimento

4.1 Introdução

Neste Capitulo serão abordadas algumas experiências, métodos, ou novos conhecimentos adquiridos durante as várias abordagens dadas para percepção dos protocolos MAC e da sua extracção de dados da rede.

4.2 Multihop

Uma das vantagens das RSSFs reside no facto de as ligações entre os nós se adaptarem automaticamente as mudanças de local dos respectivos nós. Desta forma consegue-se enviar dados, desde o nó fonte até ao nó de destino, utilizando os nós entre eles que se encontram em melhor localização, dando preferência ao trajecto mais curto, ou por exemplo, o mais económico. A este tipo de comunicação dá-se o nome de *multihoop*.

Utilizando o programa AntiTheft e considerando três nós Micaz, podemos verificar o funcionamento da comunicação *multi-hoop* na rede, já que cada nó deste tipo tem um alcance de comunicação de cerca de 30m em linha de vista (LoS) sendo necessário utilizar o *multi-hoop* para que o mesmo nó consiga comunicar a distâncias superiores.

O programa AntiTheft tem como objectivo avisar sobre o possível roubo de um objecto, tendo para isso de se ter previamente colocado um nó sensor unido ao objecto, o que permitirá a recolha das acelerações desse mesmo objecto. Sempre que a sua aceleração ultrapasse um certo valor, será enviado um sinal de aviso para o computador

através do nó base e mesmo de nós intermédios caso seja necessário. Para programar os nós deve-se ligar o nó base à placa programadora MIB510. Executa-se o seguinte comando:

cd /opt/tinyos-2.x/apps/AntiTheft/Root; SENSORBOARD=mts300 make micaz install,1 mib510,/dev/ttyS0

Em seguida, ligam-se, um a um, os restantes nós á placa programadora e executa-se o seguinte comando:

cd /opt/tinyos-2.x/apps/AntiTheft/Node; SENSORBOARD=mts300 make micaz install,<ID number node> mib510,/dev/ttyS0

sendo <lD number node> o número que identifica, de forma única, os restantes nós (2, 3, 4, etc). Para se poderem visualizar os alertas enviados por cada nó, executa-se os seguintes comandos em diferentes janelas:

cd /opt/tinyos-2.x/apps/AntiTheft/java; java net.tinyos.sf.SerialForwarder -comm serial@/dev/ttyS0:micaz; #na janela de comandos A

cd /opt/tinyos-2.x/apps/AntiTheft/java; ./run #numa outra janela de comandos B

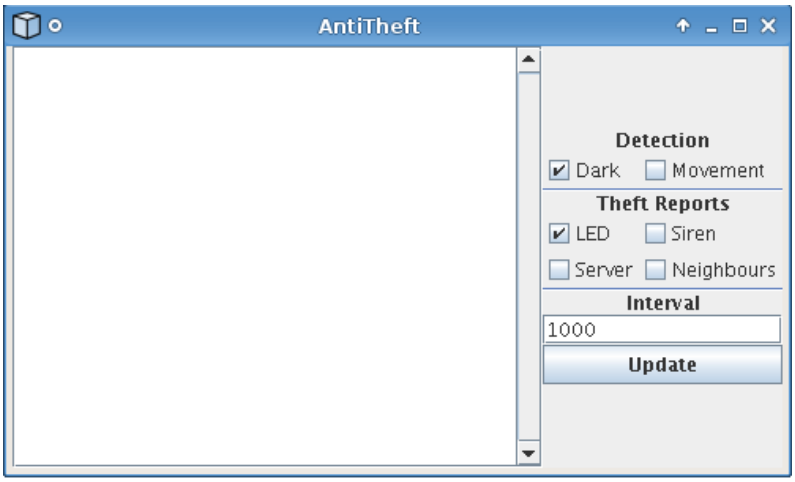


Figura 4.1 – Janela para visualizar e configurar os alertas enviados pelos nós sensores.

Podemos verificar que, ao mover-se um nó, irá aparecer no computador um alerta com a indicação do número desse mesmo nó. Para verificarmos o funcionamento *multi-hoop* da rede, afastou-se um nó sensor (por exemplo, o nó 2) do nó base, sem que existam nós intermédios entre eles, parando quando o alerta de nó em movimento deixar de aparecer na janela do computador. Leva-se então um outro nó (por exemplo, o nó 3) de encontro ao nó 2 que afastámos, e agitamos um pouco o nó 2 para existirem acelerações não nulas. Pode-se verificar que as mensagens de alerta do nó 2, que se

encontra afastado, irão aparecer novamente na janela visto que agora existe um nó intermédio (nó 3) que é utilizado para reencaminhar a mensagem vinda do nó 2.

4.3 Perl

O Perl é uma linguagem de programação estável e multi-plataforma, usada em aplicações de missão crítica em todos os sectores. Destaca-se a sua utilização no desenvolvimento de aplicações web de qualquer tipo [11]. Foi criada por Larry Wall em Dezembro de 1987. A origem do Perl provém do *shell scripting*, Awk e da linguagem C. Encontra-se disponível praticamente para todos os sistemas operativos, embora seja utilizado habitualmente em sistemas Unix. O nome foi prposto originalmente por Larry Wall em referência à Parábola da Pérola, de Mateus 13, tendo-se alterado a sua grafia de "*Pearl*" para "*Perl*" visto que o primeiro nome já estava registado noutra linguagem de programação. Algumas expansões possíveis foram propostas posteriormente, tais como *Practical Extraction and Report Language* e *Pathologically Eclectic Rubbish Lister*, este último tendo sido proposto pelo próprio *Larry Wall*, conhecido por sua personalidade sarcástica e criativa.

A linguagem Perl é uma das preferidas dos administradores de sistemas ou de autores para aplicações *web* pois é bastante versátil no processamento de cadeias de texto (*strings*), manipulação de texto ou *pattern matching* através de expressões regulares, para além disso, permite que o período de desenvolvimento do programa seja bastante curto. A linguagem Perl já foi incorporada em mais de 100 plataformas diferentes, sendo bastante utilizada no desenvolvimento de aplicações para a *web*, finanças e bioinformática.

4.3.1 Características da linguagem

A sintaxe da linguagem Perl é, de uma forma geral, semelhante à linguagem C em vários aspectos, tais como nas suas variáveis, expressões, atribuições, delimitação de blocos de código, estruturas de controlo e sub-rotinas.

Além disso, a linguagem Perl recebeu influências das linguagens de *shell script* tendo as suas variáveis escalares sido precedidas por um cifrão ($). Esta marcação permite identificar perfeitamente quais são as variáveis do programa, onde quer que elas estejam. Um dos melhores exemplos da utilidade desse recurso é a inserção de variáveis directamente no conteúdo das *strings*. Esta linguagem possui também muitas funções já integradas e que permitem realizar tarefas habituais como, por exemplo, a ordenação e o

acesso a arquivos no disco.

O Perl possui também semelhanças com as listas de Lisp, com os *arrays* associativos (tabelas *hash*) de Awk e com as expressões regulares de sed. Todas estas características simplificam e facilitam qualquer forma de interpretação e tratamentos de textos e dados em geral.

A linguagem suporta estruturas de dados complexas, possui recursos da programação funcional, como a passagem de valores retornados da execução de uma função e possui um modelo de programação orientado a objectos. O Perl também possui variáveis orientadas para o léxico, que tornam mais fácil a escrita de código mais robusto e modularizado.

Todas as versões de Perl possuem uma gestão da memória automática e atribuição do tipo das variáveis dinâmica. Os tipos e necessidades de cada objecto de dados no programa são determinados automaticamente; a memória é reservada ou libertada de acordo com as necessidades. A conversão entre tipos de variáveis é feita automaticamente durante a execução resultando em erros fatais quando as conversões são ilegais.

Em resumo constata-se que:

- O Perl aproveita as melhores características das linguagens C, awk, sed, sh, e BASIC, entre outras;
- A interface de integração com base de dados (DBI) suporta muitas base de dados, incluindo Oracle, Sybase, PostgreSQL, MySQL e outros;
- O Perl tem módulos para trabalhar com HTML, XML, e outras linguagens de *markup*;
- O Perl suporta Unicode;
- O Perl permite programação processual orientada a objectos.;
- O Perl pode aceder a bibliotecas externas em C/C++ através de XS ou SWIG;
- O Perl é extensível, existindo milhares de módulos disponíveis no *Comprehensive Perl Archive Network* (CPAN);
- O compilador Perl pode ser incorporado noutros sistemas.

Também é importante resolver os seguintes pontos:

- Perl é uma linguagem estável e multiplataforma;

- É usada em aplicações críticas, qualquer que seja o sector;
- Perl é um software livre, disponível sob a Licença Artística ou a GNU *General Public Licens*e (GPL);
- Perl foi criada por Larry Wall;
- Perl 1.0 foi lançada na usenet alt.comp.sources em 1987;
- Possui mais de 5 mil módulos que abrangem as mais variadas finalidades;
- O lema desta linguagem é: "*There's More Than One Way To Do It*" (TMTOWTDI);

4.3.2 Comando *make*

O comando "ncc" é o responsável pela compilação do código nesC para linguagem C, gerando-se um ficheiro "*.c" que fica alojado na directoria <aplicação>/build.

O comando "make" encontra-se escrito em linguagem Perl, na directoria "/opt/MoteWorks/make" do TinyOS, e serve basicamente para simplificar a compilação do programa evitando a utilização directa do comando "ncc" pois seria necessário fornecer muitos parâmetros a este comando. O comando "make" não requer tanta informação na própria linha de comandos, pois ele atribui a esses parâmetros valores por defeito, caso não tenham sido passados como parâmetro.

Nas Figura 4.2 e 4.3 podemos verificar a quantidade e variedade de parâmetros fornecidos ao comando "ncc". Em ambas as figuras, foi executado o comando "make" para a mesma aplicação, tendo-se na Figura 4.3 especificado qual o protocolo a utilizar através do parâmetro *route*. Na Tabela 4.1 está indicado algum do código Perl do ficheiro "/opt/MoteWorks/make/avr/route.extra", que define a utilização deste parâmetro assim como os protocolos que podem ser passados através do mesmo. Conforme se pode ver nas figuras e na tabela a definição de *route*, irá alterar o valor do parâmetro -DROUT_PROTOCOL passado para o compilador "ncc". O valor definido neste parâmetro ficará alojado na variável ROUT_PROTOCOL (sem o -D) que pode ser acedida tanto na linguagem nesC (aplicação para o mote) como na linguagem Perl (código em que foi escrito o comando make).

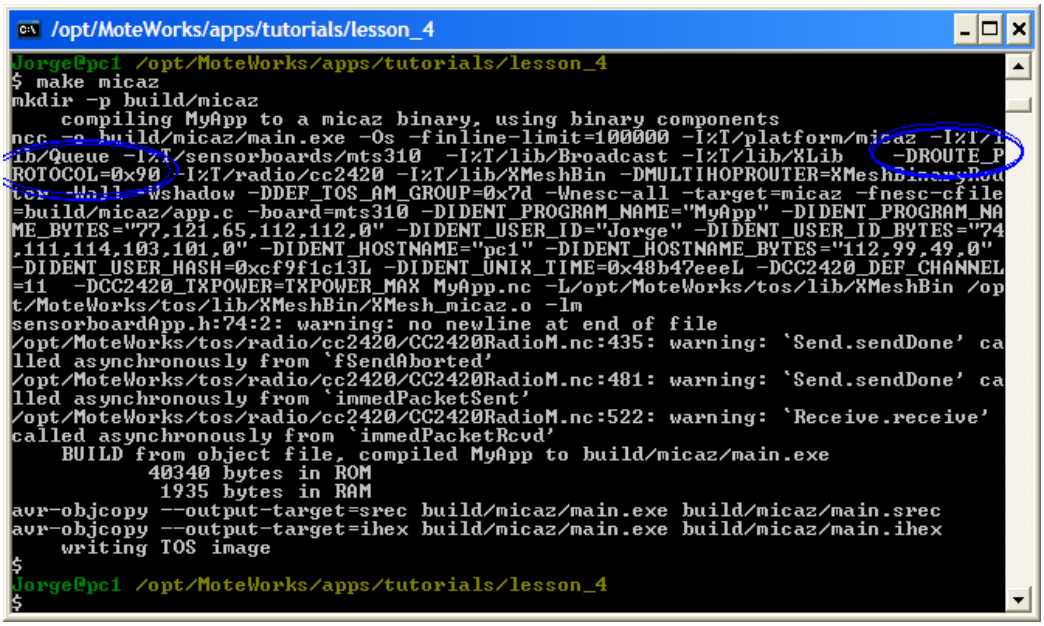


Figura 4.2 – Exemplo dos parâmetros passados ao comando "ncc" pelo comando "make micaz".

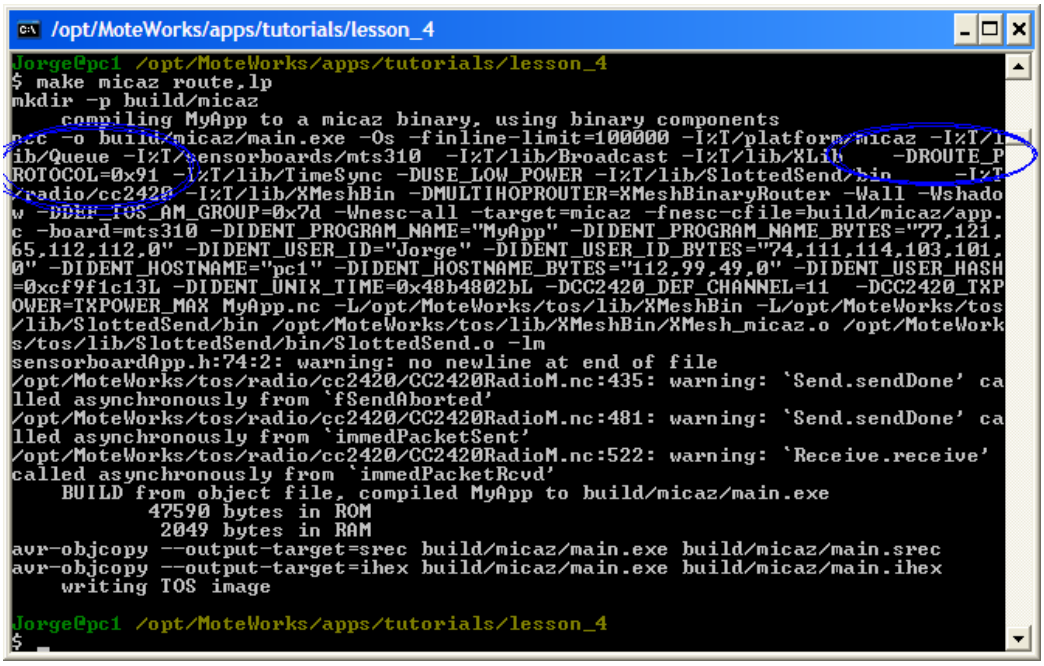


Figura 4.3 – Exemplo dos parâmetros passados ao comando "ncc" pelo comando "make micaz route,lp".

Tabela 4.1 – Definição do comando "make … route,<stack>"

# make micaz ... route,<stack>	where stack is one of:	
#		
# hp	XMesh High Power	ROUTE_PROTOCOL=XMESH_HP=0x90
# lp	XMesh Low Power	ROUTE_PROTOCOL=XMESH_LP=0x91
# elp	XMesh Extended Low Power	ROUTE_PROTOCOL=XMESH_ELP=0x92

```
# VLP       very low power (time synced low power) ROUTE_PROTOCOL=XMESH_VLP=0x93
#
# surge       Surge Reliable                          ROUTE_PROTOCOL=SURGE=0x70
# surge_lpl  Surge Low Power Listening                ROUTE_PROTOCOL=SURGE_LPL=0x71
# surge_ts   Surge Low Power with Time Sync           ROUTE_PROTOCOL=SURGE_TS=0x72
# mint        Mint Route                               ROUTE_PROTOCOL=SURGE_MINT=0x73
#

#default always assigned.  This is done also in binlink.extra
ifeq ($(ROUTE),)
   ROUTE=hp   #valor por defeito, caso no comando make não esteja definido um valor para ROUTE
endif

#
# HP -- high power mesh backbone
#
ifeq ($(ROUTE), hp)  #verifica se ROUTE é igual a hp
    PFLAGS += -DROUTE_PROTOCOL=0x90

   #path for cc1000 radio
   ifeq ($(RADIO_CHIP), CC1000)
      PFLAGS += -I$(XBOWROOT)/radio/cc1000hp
   endif

   #path for cc2420 radio
   ifeq ($(RADIO_CHIP), CC2420)  # verifica se RADIO_CHIP é igual a CC2420, no caso do mote
                                 # MicaZ esta condição é verdadeira
      PFLAGS += -I$(XBOWROOT)/radio/cc2420 # indica em que directoria se encontram os
                                           #componentes que controlam o rádio
   endif

(…)

endif

#
# LP == low power (low power sampling)
#
ifeq ($(ROUTE), lp)
          PFLAGS += -DROUTE_PROTOCOL=0x91
```

```
PFLAGS += -I$(XBOWROOT)/lib/TimeSync
PFLAGS += -DUSE_LOW_POWER

#CC2420
ifeq ($(RADIO_CHIP), CC2420)
        #Binary
        ifeq ($(RADIO_BUILD_MODE),binary)
                PFLAGS += -I$(XBOWROOT)/lib/SlottedSend/bin
        Endif

(…)
```

4.4 Hardware

4.4.1 Módulo processador/rádio

Para a emissão/recepção e processamento do sinal seleccionou-se a plataforma MICAz ZigBee Series (MPR2400), disponibilizada pela Crossbow, Figura 4.4 [12]. Este módulo utiliza um microcontrolador ATMega 128L, apresentado no Anexo A.

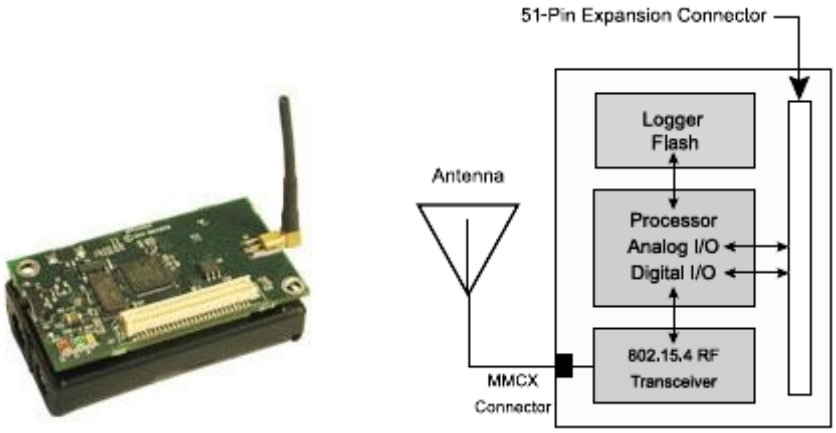


Figura 4.4 – MICAz ZigBee Series (MPR2400) e respectivos blocos constituintes [12].

Esta plataforma é caracterizada por**:**

- Emissão/recepção a 2,4 GHz, norma IEEE 802.15.4;
- *Tiny* (dimensões minúsculas) do sistema sem fios;
- Certificação pela *Federal Communications Commission* (FCC);
- Ter sido projectado especificamente para integrar redes de sensores sem fios;
- Elevado ritmo de transmissão via interface rádio, 250 kbps;

- Comunicação sem fios com nós da rede com capacidade de encaminhamento;
- Conector de expansão que permite a entrada e saída de sinais. Através deste conector, recebem-se os dados dos sensores de luz, temperatura, humidade, pressão barométrica, aceleração/sísmica, microfone, magnetómetro, etc.

4.4.2 Placa de programação

A placa de programação seleccionada para ligar o mote ao computador permite a recolha de dados da rede ou a "descarga" do programa em *TinyOS* para o mote. Trata-se de uma MIB510 – *Serial Gateway* e é do mesmo fabricante do mote MICAz, tendo sido projectada especificamente para estes motes. A placa possui as seguintes características:

- Pode-se ligar à plataforma MICA, MICA2 e MICAz (através do conector branco de 51 pinos que é visível na Figura 4.5) e MICA2DOT (através do conector circular);
- Possui uma porta RS-232 para comunicação série com um computador a um ritmo superior a 57.6kbps se estiver ligado a um MICA2;
- Tem uma ligação para uma fonte externa de +5V DC;
- Inclui um circuito para detecção de subtensão;
- Permite ver o estado dos leds do MICA, MICA2, MICAz e MICA2DOT na própria placa para uma depuração mais fácil;
- Para programar não é necessário a porta paralela;
- Endereça e corrige problemas relacionados com a programação e erros de memória;
- Permite o *download* rápido do programa para o mote, através da porta série a uma taxa de 115kbps.

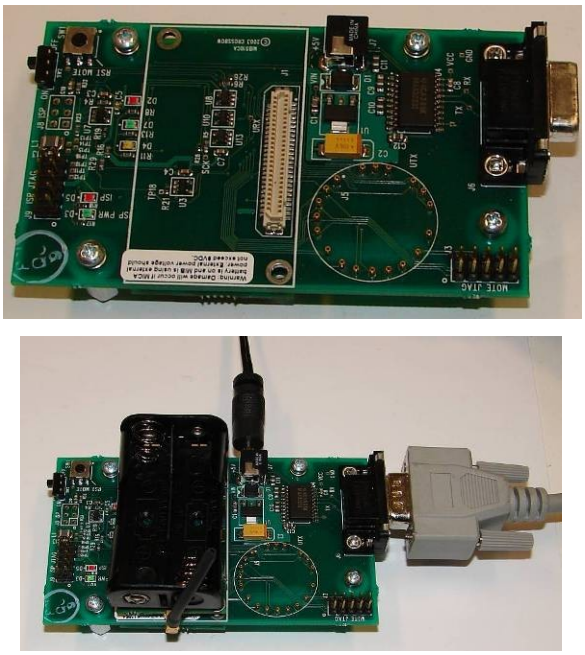


Figura 4.5 – Placa programadora MIB510.

4.5 Simulador

4.5.1 Introdução

Para optimização e estudo do protocolo MAC tentou-se inicialmente actuar directamente no código NesC do TinyOS de forma a poderem-se obter resultados reais utilizando o código alterado nos nós sensores. Devido a alguma complexidade em extrair os dados dos nós (pois a própria troca desses dados irá interferir nos resultados extraídos), e à dificuldade em se obter o diagrama de estados completo, abandonou-se temporariamente esta iniciativa, passando-se a focar a atenção nos simuladores existentes.

4.5.2 Selecção do simulador

Ao analisar-se um vasto leque de simuladores escolheram-se os três principais como sendo o OMNeT++, o NetTopo e o GTNetS (Tabelas 4.2-4.4). Estes simuladores foram seleccionados por terem sido desenvolvidos propositadamente para Redes de Sensores sem Fios (RSSFs), por terem o nível de detalhe apropriado, por já incluírem os protocolos IEEE 802.15.4 PHY e MAC, pelo facto de o código fonte ser aberto, e ainda por possuírem uma interface amigável com o utilizador.

Tabela 4.2 – Simulador OMNeT++.

Nome:	OMNeT++ http://www.omnetpp.org/
Licença:	Freeware
Descrição:	O OMNeT++ tem código fonte público, é baseado em componentes, é modular e tem um ambiente de simulação de arquitectura aberta, com um forte suporte por parte da Interface Gráfica de Utilizador e um núcleo de simulação embebido. A sua principal área de aplicação é a simulação de redes de comunicação, devido á sua arquitectura flexível e genérica, que tem sido utilizada com sucesso em outras áreas como a simulação de sistemas IT, redes de filas de espera (*queuing networks*), arquitecturas de hardware, bem como processos de negócio.
Plataforma:	Windows XP, Linux
Vantagens:	• Uma das maiores vantagens do OMNeT++ é o facto de se escrever em C ++. • O OMNeT++ é gratuito para fins académicos e a sua utilização não tem fins lucrativos. • Tem uma interface gráfica de utilizador poderosa; para detecção, depuração e animação das simulações. • A ferramenta de documentação gera documentação de alta qualidade, a partir do modelo de código fonte comentado, com gráficos, tabelas e referências cruzadas. Integra-se ainda com a ferramenta de documentação Doxygen C++. • Tem uma vasta gama de aplicações. • É de referir a possível utilização do nesCT e suas vantagens.
Desvantagens:	-

Tabela 4.3 –Simulador NetTopo.

Nome:	NetTopo http://www.semanticreality.org/nettopo/index.htm
Licensa:	O código fonte do NetTopo é Freeware na versão Java
Descripção:	• NetTopo é um programa de simulação e a visualização em código fonte aberto orientado para investigação, projectado para testar e validar algoritmos para RSSFs • A finalidade do NetTopo é a de construir uma ferramenta de simulação e visualização de RSSFs que fornece ao utilizador uma extraordinária flexibilidade para simular os seus próprios algoritmos e está a possibilitar a substituição de simuladores comerciais nas aplicações a RSSFs. • Usado para auxiliar a investigação de algoritmos de encaminhamento RSSFs.
Plataforma:	• Windows e Linux
Vantagens:	• Topologia de saída da rede de sensores personalizada. • Nós sensores personalizáveis com atributos definidos pelo utilizador. • Fácil extensibilidade para simular algoritmos definidos pelo utilizador. • Simulação passo-a-passo específica para comparação entre GPSR e TPGF. • Funções baseadas em ficheiros facilitam (ao utilizador) salvar e recuperar o processo simulado. • Interface gráfica do utilizador com o sistema operativo local "look and feel". • Suporte para importação e exportação XML. • Plataforma independente (escrita em Java). • Integração e virtualização das RSSFs reais testadas. • Suporte para 2D e 3D. • O resultado da simulação é formulado com um só formato que permite que o sentido da simulação do utilizador seja formulado num formato unificado que também possibilita a

	exportação para o Microsoft Office Excel para elaborar os gráficos. • Os utilizadores podem facilmente definir um elevado número de parâmetros iniciais no arranque dos nós sensores, por exemplo, energia residual, largura de banda e raio de cobertura. • Os utilizadores também podem definir e ampliar o comportamento do processamento interno dos nós sensores, tais como o consumo de energia e gestão de banda. • Permite aos utilizadores simular redes homogéneas de escala extremamente elevada. • O módulo de visualização funciona como um componente de ligação, a fim de se visualizar a conexão dos estados, a topologia e dados medidos das RSSFs. • Os dados medidos capturados a partir da rede de nós sensores reais podem conduzir a simulação a uma RSSFs virtual pré destacada. • Topologia de traçados e algoritmos de RSSFs virtuais são personalizáveis e trabalham como plug-ins definidos pelo utilizador.]
Desvantagens:	--------

Tabela 4.4 – Simulador GTSNetS.

Nome:	GTSNetS http://perso.citi.insa-yon.fr/twatteyn/documents/doxy_gtsnets/
Licença:	Livre
Descrição:	O Georgia Tech Sensor Network Simulator (GTSNetS) é um simulador de redes de sensores sem fios em grande escala. É caracterizado pela sua escalabilidade, adaptabilidade e extensibilidade. Pode ser usado para simular RSSFs composta por várias centenas de milhares de nós. A adaptabilidade provém dos diferentes métodos incluídos na implementação base dos modelos de consumo de energia, modelos de precisão de leitura,

	aplicações e protocolos de encaminhamento e opções de localização. A extensibilidade provém de implementações modulares utilizando a linguagem C++ (de programação orientada a objectos).
Plataforma:	Linux (apenas GTNetS em Windows)
Vantagens:	• Desenvolvido para RSSFs. Simula também sistemas de controlo • Este simulador permite ao utilizador escolher entre diferentes alternativas implementadas: diferentes protocolos de rede, diferentes tipos de aplicações, diferentes sensores de energia e de diferentes modelos e precisão. Se necessário, pode-se facilmente adicionar novos modelos. • Pode ser utilizado para recolher estatísticas detalhadas de um sensor específico na rede, tanto ao nível da unidade funcional, bem como ao nível do nó ou ao nível da rede.
Desvantagens:	O Modelo de simulação de alto nível funcional do processo de comunicação de rádio, em termos das interacções dos acontecimentos. Como resultado, obtém-se um excelente desempenho na simulação, mas normalmente não se obtêm informações precisas de relógio, sendo esse um factor crítico para realizar a depuração e a optimização da potência. Apenas foi indexado um *paper* pelo IEEE

Depois de se analisar as características dos três simuladores individualmente, concluiu-se que o melhor simulador que responde às necessidades de simulação para RSSFs é o OMNeT++ porque tem código fonte público, é baseado em componentes, é modular e tem um ambiente de simulação de arquitectura aberta com um forte suporte à Interface Gráfica de Utilizador e com um núcleo de simulação incorporado.

Outra vantagem na utilização do OMNeT++, comparativamente com os outros dois simuladores, é o facto de o OMNeT++ se ter tornado bastante popular na comunidade científica e industrial, tendo-se publicado vários modelos. Isso levou a que se a sua comunidade tenha crescido e que esteja cada vez mais forte e dinâmica, com um fórum onde se podem trocar informações com pessoas que trabalham em RSSFs, o

que é bastante útil para a depuração e resolução de problemas relacionados com a simulação de RSSFs. Além disso, relativamente aos outros dois simuladores, conclui-se que o OMNeT++ é mais rápido em pelo menos uma ordem de grandeza e que gere a memória de forma mais eficiente.

O OMNeT++ tem a vantagem de suportar dois tipos de modelos de simulação, uma baseada em eventos e outra orientada a processos únicos, enquanto GTNetS apenas suporta processamento de eventos discretos.

O NetTopo foi desenvolvido sobre a plataforma Java™ enquanto o OMNeT++ utiliza a plataforma C++. Este facto leva a que o NetTopo não disponibilize um acesso total aos seus recursos e que o seu desempenho seja inferior ao do OMNeT++. Por outro lado as aplicações em linguagem C++ superam muitas vezes o desempenho das mesmas aplicações escritas em linguagem Java em operações aritméticas e trigonométricas.

Por fim, escolheu-se o OMNeT++ por ser modular, totalmente programável e por ter sido projectado para suportar modelos de redes com muitos nós, baseando-se em modelos de componentes reutilizáveis.

4.5.3 Castalia

Instalou-se o módulo Castalia a correr sobre o OMNeT++. O Castalia é um simulador para RSSFs, *Body Area Networks* (BAN), e de um modo geral para sistemas embebidos de potência reduzida. É baseado na plataforma OMNeT++, sendo utilizado por investigadores e programadores para testar os seus algoritmos distribuídos e/ou protocolos, num modelo de rádio e canal de Redes Sem Fios realista, com um comportamento realista dos nós especialmente no que diz respeito ao acesso ao rádio. O simulador Castalia utiliza o modelo de sombreamento log-normal (*lognormal shadowing model*) para simular a atenuação média de sinal ao longo do trajecto do pacote, sendo utilizado para modelar os dados empíricos na RSSFs. Simula também variações temporais da atenuação de sinal no trajecto, considerarando fenómenos de desvanecimento durante a mudança de ambientes (isto é, quando existe movimento dos nós ou de partes do ambiente). O modelo de variação temporal do Castalia foi projectado para filtrar os dados medidos em vez de assumir hipoteses específicas sobre a criação de *fast fading* (desvanecimento rápido). O Castalia possui outras características como a modelação de processos físicos, a medição de dispositivos de direcção e ruído, a deriva do relógio no nó, e a implementação de vários protocolos MAC e de mapeamento.

O simulador Castalia encontra-se em desenvolvimento no National TIC Austrália desde 2006. Em 2007, ficou com o código fonte acessível ao público através de uma licença disponibilizada para fins académicos. Actualmente, encontra-se na versão 2.1b.

Deve-se ter em atenção a versão do Castalia e do OMNeT++ para que as mesmas sejam compatíveis. Neste projecto, optou-se pelas versões Castalia 2.1b e OMNeT++ 3.3. Depois de correctamente instalados, deve-se abrir a linha de comandos e utilizar o seguinte conjunto de comandos para executar a simulação do programa valueReporting:

rm "/home/jorge/omnetpp33/Castalia-2.0/Simulations/valueReporting/Castalia-Debug.txt"; rm "/home/jorge/omnetpp33/Castalia-2.1b/Simulations/valueReporting /Castalia-Primary-Output.txt"; cd "/home/jorge/omnetpp33/Castalia-2.1b "; make clean; ./makemake; make all; cd "/home/jorge/omnetpp33/Castalia-2.1b /Simulations/valueReporting"; ./run*

Como o objectivo de se uniformizar o comando de forma a poder ser executando dentro de qualquer directoria, conjugaram-se os vários comandos separados por ponto e vírgula. Começa-se por eliminar os ficheiros de saída dos dados com o comando "rm" de seguida muda-se para a directoria do Castalia, executa-se uma limpeza de alguns ficheiros resultantes da compilação, e compila-se novamente o código. De seguida muda-se para a directoria do programa e executa-se o script *runValueReporting*. A saída dos dados do programa irá ficar nos ficheiros "Castalia-Debug.txt" e "Castalia-Primary-Output.txt".

4.5.3.1 Diagrama de transição de estados do módulo MAC

O módulo MAC tem quatro estados [13]:

- MAC_STATE_DEFAULT;
- MAC_STATE_TX;
- MAC_STATE_CARRIER_SENSING;
- MAC_STATE_EXPECTING_RX;

Cada um destes estados está estreitamente ligado a um comportamento específico do módulo. Isto significa que cada vez que o módulo MAC recebe uma mensagem a partir do módulo de aplicação ou do rádio, ele irá processar a mensagem de acordo com o seu estado actual.

É de salientar que nem todos os tipos de mensagens podem alterar o estado do

nível MAC. Para além disso, as mensagens do mesmo tipo podem afectar o módulo do nível MAC de várias maneiras nas diferentes etapas da simulação, porque o estado actual e outras variáveis pertencentes ao objecto do módulo desempenham um papel principal na escolha do estado de transição. O diagrama de transição de estados na Figura 4.6 apresenta os estados do módulo MAC. O diagrama de transição de estados na Figura 4.7 apresenta os estados do módulo MAC.

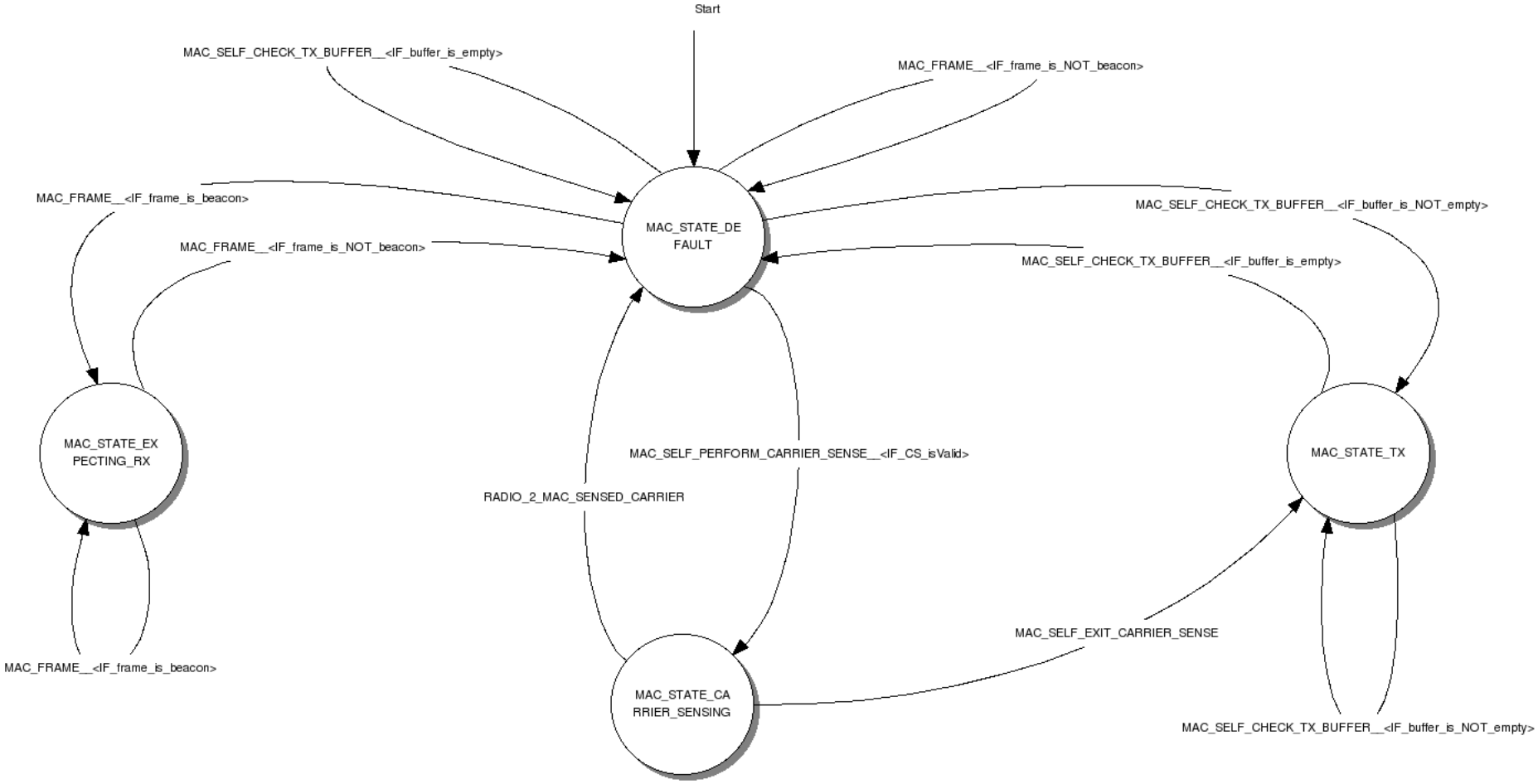


Figura 4.6 – Diagrama de estados do MAC [14].

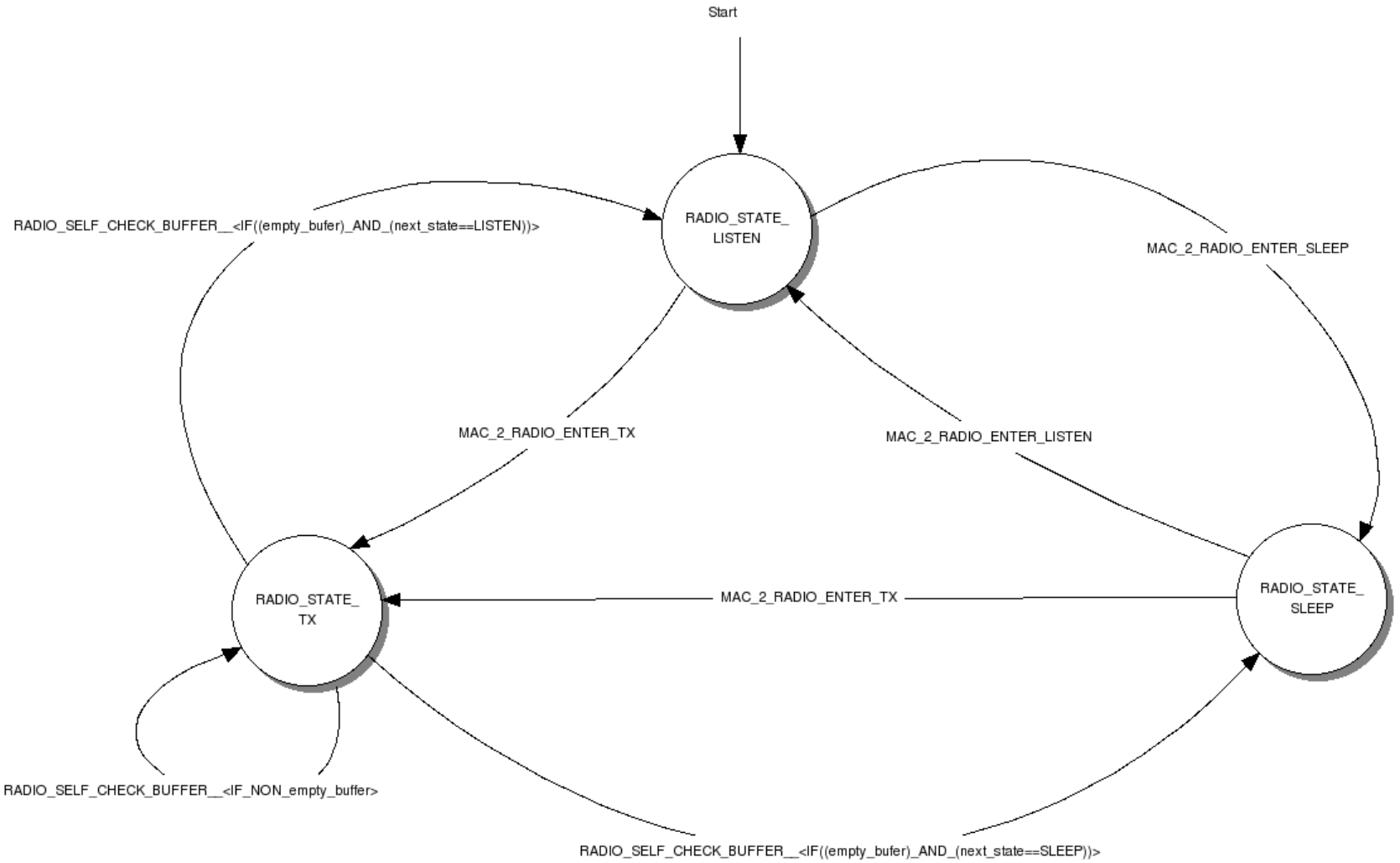


Figura 4.7 – Diagrama de estados para o Rádio [14].

As Transições entre estados ocorrem de uma forma bem definida, sendo:

- O estado actual do Módulo representado por círculos;
- O tipo/natureza da mensagem recebida representadas pelas setas de transição;
- Algumas condições actuais baseadas nos valores de certas variáveis pertencentes ao módulo MAC seguem o tipo da mensagem e têm a forma de <IF …>.

Para entender o funcionamento do módulo MAC não é suficiente descrever a máquina de estados finitos (MEF) da Figura 4.6; existem muitos tipos de mensagens extra que são trocadas entre MAC e a Aplicação, ou entre MAC e módulos do Rádio, ou mesmo entre MAC e MAC (auto-mensagens para si próprio) e que nada têm a ver com a transição de estados.

Nas linhas seguintes serão explicados todos os tipos de mensagens "automáticas" que o módulo MAC pode receber e manusear. Estão assinalados os tipos de mensagens incluídas no MEF da Figura 4.6.

O significado de cada campo das mensagens "automáticas" é apresentado na Tabela 4.5.

Tabela 4.5 – Significado de cada campo das mensagens "automáticas"

MSG_KIND_FIELD	Fornece um apontador (msg) para o objecto CMessage, MSG_KIND_FIELD é o tipo de retorno de msg -> kind(). Tem o mesmo tipo do que é recebido pelo módulo da mensagem, sendo esta informação preciosa para a determinação da forma como a mensagem é processada pelo módulo MAC.
Senders	São os módulos que possibilitam o envio de mensagens do tipo MSG_KIND_FIELD ao módulo MAC. De referir que nem todos os módulos estão preparados para enviar todos os tipos de mensagem.
Triggered messages	Depois do módulo MAC receber a mensagem, ela continua a ser processada de forma apropriada baseando-se no MSG_KIND_FIELD. Enquanto ocorre o processamento ou mesmo no final o módulo pode criar e enviar novas mensagens para outros módulos (Aplicação ou Rádio). Neste campo da Tabela, estão contidas as mensagens

	"disparadas" enviadas pelo MAC.
Direct State transitions	Se o tipo de mensagem afectar directamente o estado do MAC, a possível transição será aqui apresentada. *Direct* significa que o estado é alterado antes do processamento terminar.
Indirect State transitions	Muitas vezes, alguns tipos de mensagens não alteram directamente o estado do módulo MAC, mas este processo produz novas mensagens "automáticas" que são capazes de alterar um estado num futuro próximo (relativamente ao tempo da simulação), ou seja calendarizar novos acontecimentos
Handling Description	Aqui é apresentada uma explicação em linguagem natural, sobre como o MAC processa a mensagem recebida com a MSG_KIND_FIELD dada.

Podem-se notar algumas diferenças entre MEF e as tabelas seguintes C.1 a C.14 apresentadas no anexo B [13]. Estas mudanças foram realizadas na Figura 4.6 principalmente porque alguns tipos de mensagens "disparam" a execução de outras mensagens, que irão realmente mudar o estado do MAC.

No entanto, por vezes é mais importante perceber qual o tipo de mensagem que desencadeia inicialmente o processo da mudança de estado. Por exemplo, quando o MAC está no estado MAC_STATE_EXPECTING_RX e recebe a MAC_FRAME, que não é *beacon*, o estado não é alterado directamente para MAC_DEFAULT_STATE, mas existe uma mensagem intermédia que irá fazer essa transição (MAC_SELF_EXIT_EXPECTING_RX). Se no diagrama se utilizar MAC_SELF_EXIT_EXPECTING_RX para a transição, então ficará claro qual o motivo dessa mesma transição.

Os termos Mensagens, Pacotes, *Frames (Tramas) e Beacons têm o seguinte significado* [13]:

- **Mensagens** - são os objectos que são trocados entre cada módulo simples e que contêm apenas dados de controlo;
- **Pacotes** - são as mensagens que são enviadas pelo módulo de Aplicação para

ao módulo de Comunicação (mais especificamente para o módulo MAC), com vista a serem difundidas/transmitidas para a rede. O módulo MAC também envia de volta pacotes com dados para a camada de Aplicação;

- ***Frames*** (Tramas) - são as mensagens que contêm dados e que são trocadas somente entre o módulo MAC e o módulo de Rádio. Estas são respectivamente as MacFrames e as PhyLayerFrames;
- ***Beacons*** - são um caso especial de MacFrames que não contêm dados reais. São utilizadas (e criadas) pelo módulo MAC quando o *duty cycle* está em uso, a fim de despertar (ou melhor, a fim de manter acordados), os nós vizinhos/receptores que possivelmente possam estar a dormir.

4.5.3.2 Apresentação dos Parâmetros a Alterar

O Protocolo Medium Access Control (MAC) é uma parte importante do comportamento do nó, por conseguinte é definido num módulo separado. No simulador Castalia estão implementados dois módulos principais MAC, sendo eles o TunableMAC que fornece vários parâmetros ao utilizador e uma aplicação para afinação. A partir do TunableMAC também se tem o comportamento de um simples protocolo MAC CSMA/CA (definiu-se um ficheiro "ini" para incluir na simulação). O outro módulo MAC implementa o T-MAC conseguindo-se através deste obter o S-MAC alterando apenas um parâmetro (T-MAC é um reforço a capacidade do S-MAC permitindo estender o tempo de actividade).

O MAC Just Carrier Sense é um protocolo simples de MAC CSMA, que é implementado como o Tunable MAC mas com os seus parâmetros definidos para apenas garantir a função CSMA.

O BypassMAC é essencialmente um "não" MAC, tal como o como o nome sugere. Como o Castalia espera ter um módulo MAC para ser definido e ligado a outros módulos, existe este módulo para quando o não é necessária a camada MAC.

Tunable MAC

O projecto do simulador Castalia teve como grande motivação dos seus autores, testar um protocolo MAC altamente ajustável numa situação com condições no canal de rádio reais. Castalia aproxima-se de vários protocolos de duty-cycling existentes (ex: B-MAC, LPL). Todavia é de referir que, este protocolo foi construído a pensar na difusão das comunicações transmitidas, assim não suporta acknowledgments

ou pacotes de controlo RTS e CTS. A principal função deste protocolo é a de gerar um ciclo de rádio e a de transmitir um conjunto apropriado de beacons antes de cada dado transmitido (visto que as transmissões dos nós não se encontram temporalmente ordenadas). Também fornece diversas outras funções parametrizadas como as retransmissões, transmissões probabilísticas e máquinas recuadas. Pode-se observar o ficheiro:

Simulations/Parameter_Include_Files/MAC_Tunable.ini

para uma regulação completa dos vários parâmetros dos módulos.

Com o primeiro define-se o nome do módulo (macModuleName), ficando assim o simulador Castalia a saber qual o módulo a utilizar. Os três seguintes parâmetros foram implementados para procura e análise de erros de programação (printDebugInfo, printPotentiallyDroppedPacketsStatistics, printStateTransitions). De seguida descrevem-se 10 parâmetros relevantes com interferência no comportamento do MAC.

- ***dutyCycle*** - É a fracção de tempo que o nó permanece á escuta do canal. Consequentemente (1-*dutyCycle*) será a fracção de tempo que o nó permanece no modo de repouso. Este parâmetro será provavelmente o mais importante no que diz respeito ao consumo de energia e pode reduzir drasticamente o tempo de escuta.
- **listenInterval -** Este é o período de tempo que o nó permanece á escuta. Conhecendo o *duty cycle* pode-se definir o momento em que o nó entra em modo de repouso. Depois de se ter um intervalo de escuta e de repouso começa-se novamente o ciclo. É preferível que o período de escuta seja pequeno ficando assim minimizada a latência da entrega dos dados, num dado *duty cycle*. Se tivermos um intervalo de escuta demasiado reduzido, não será possível receber os pacotes na totalidade (*beacons* ou pacotes de dados). No caso dos rádios de 250kbps que são bastante utilizados, um bom momento para iniciar a escuta será entre os 5ms e os 10ms para um rádio de 250kbps, normalmente utilizado. Poderá escolher outros de modo a encontrar o melhor para o seu cenário.
- **BeaconIntervalFraction** - Quando temos um *duty cycle* necessitamos de uma forma de receber a atenção do nó adormecido antes de transmitir os dados. O método mais comum consiste em utilizar um conjunto de *beacons* antes da transmissão dos dados. Quando existe a necessidade de se garantir que o nó receptor irá permanecer acordado, deve-se enviar o conjunto de *beacons* pelo

menos durante todo o intervalo de repouso. Mas imagine-se que se têm vários nós vizinhos e que não é necessário acordá-los todos. Neste caso, assume-se por exemplo que (em termos estatísticos) se pretende acordar metade dos nós. Para isso o intervalo de *beacon* deverá ser metade do intervalo de repouso. Este parâmetro expressa a razão entre o intervalo de *beacon* actual e o intervalo máximo do *beacon* (igual ao intervalo de repouso). Quanto mais pequeno este parâmetro for menos energia se gasta, porém menos hipótese existem de se conseguir acordar um nó vizinho.

- **numTx -** Este número de transmissões indica a quantidade de vezes que se deve tentar transmitir um pacote. Quanto maior for este parâmetro mais energia se gasta, contudo, melhor desempenho pode ser alcançado (consegue-se melhor alcance dos nós).
- **reTxInterval** - Intervalo de retransmissão, ou seja, entre retransmissões. Com este parâmetro pode-se efectivamente difundir a transmissão esperada.
- **probTx -** Probabilidade de transmissão. Quando forem transmitidos pacotes serão transmitidos com uma certa probabilidade; probabilidade essa que é assumida por muitos protocolos como sendo 1 (acontecimento certo). Este valor combinado com o número de retransmissões pode criar um número de transmissões esperado por nó. Por exemplo, se tivermos 6 retransmissões (ou seja, um total de 7 transmissões) com uma probabilidade de transmissão de 0,5, então teremos uma probabilidade de transmissão de 3,5 por nó por cada valor de dados que se pretende transmitir.
- **RandomTxOffset -** Intervalo de Transmissão aleatório. Quando se decide transmitir algo isto não é feito de imediato, mas espera-se por um tempo aleatório distribuído entre [0..Random Transmission Offset]. Essa aleatoriedade ajuda a evitar a colisões que é muito comum num cenário do tipo *broadcast*. Se um nó transmitir os seus dados e quatro dos nós vizinhos os receberem e decidirem transmiti-lo de imediato teremos condições para haver colisões. Juntando a aleatoriedade numa transmissão podemos evitar estas colisões. Este valor não necessita de ser elevado caso os nós façam a escuta do canal antes de transmitirem (por exemplo, utilizam o CSMA).
- **backoffType** - Este parâmetro é inerente à escuta do canal. Como mencionado anteriormente, a escuta do canal é efectuada normalmente pelo MAC (parâmetro

"carrier sensing" declarado como "verdadeiro"). Isto significa que quando o MAC necessita de transmitir um pacote, e antes começar a transmitir potências *beacons*, verifica pelo rádio se o canal está livre. Se não estiver, a transmissão é adiada (*backoff*) algum tempo e o MAC coloca o rádio em *sleep mode* para poupar energia. O pârametro do tipo de *backoff* especifica o modo como o tempo de *backoff* é determinado. Se for colocado a 1 o *backoffMaxInterval* é um tempo constante definido em *backoffBaseValue*. Se colocado a 2, o backoffMaxInterval vai depender do número de vezes consecutivas em que se encontra o canal ocupado. Esta é a relação *backoffMaxInterval = (backoffBaseValue)x(times)*. Se o parâmetro for colocado a 3 novamente depende do número de vezes que encontrarmos o canal ocupado. *backoffMaxInterval = (backoffBaseValue)^times.*

- **backoffBaseValue** - É o parâmetro usado pelo *backoff type* para definir o tempo de espera por cada tipo de *backoff*.
- **randomBackoff -** Trata-se de um parâmetro verdadeiro/falso, que especifica se o tempo de *backoff* é aleatório. Este deve ser seleccionado "verdadeiro" na maioria dos casos, dado que é a forma usual de *backoff*. Se "verdadeiro", o *backoff* é seleccionado aleatoriamente do intervalo [0.. backoffMaxInterval]. Se "falso" tempo de *backoff* = *backoffMaxInterval*. Os restantes parâmetros definem o número de bytes da trama de dados, qual o *overload*, tamanho do *beacon* e da trama de ACK (embora não seja utilizada actualmente). Também se define o tamanho da fila MAC e, finalmente, se o MAC executa a escuta do canal

T-MAC e S-MAC

O T-MAC é muito popular como MAC para as RSSFs dado que usa muitas técnicas para manter baixa a energia consumida (usando taxas agressivas de *sleep*/funcionamento e sincronizações) enquanto se tenta manter o desempenho (por exemplo taxa de entrega de pacotes) elevado recorrendo a ciclos de *sleep*/funcionamento de acordo com a necessidade do tráfego. O S-MAC pode ser visto como antecedente do T-MAC, dado que introduz muitas das suas técnicas, mas usa ciclos de sleep/funcionamento mais rígidas. Um módulo Castalia (T-MAC) apresenta a funcionalidade dos dois protocolos. A implementação do T-MAC no Castalia foi complicada, tornando-se um processo bastante moroso porque muitas das questões

práticas definidas no protocolo não se encontravam bem descritas nem clarificadas no artigo inicial. Por exemplo, todos os detalhes sobre o tempo de sincronização foram escondidos por outros trabalhos. Mais importante quando os detalhes das referidas técnicas foram encontrados, nem sempre faziam sentido aplicalos como faziam com o T-MAC, havia então que tomar a decisão para manter o bom desempenho do T-MAC. Para além disto, adicionaram-se funcionalidades extra que também não estavam descritas no protocolo inicial tais como o número fixo de retransmissões falhadas de pacotes unicast. Os pormenores da implementação do T-MAC merecem ser explicados noutro documento. O parâmetro do módulo T-MAC pode ser encontrado no ficheiro : *Simulations/Parameter_Include_Files/T_MAC.ini*.

Começa-se com o nome do módulo e os 3 parâmetros para controlar a sida de debug.. Seguidamente define-se o tamanho dos diferentes pacotes de controlo do T-MAC (sync,ack, rts, cts) bem como o tamanho máximo da treama de dados e do overhead. Os onze parâmetros seguintes definem o comportamento do protocolo (na descrição seguinte o valor depois de "=" é o valor usado por defeito):

- **macBufferSize = 32** - O número máximo de pacotes de comandos superiores que podem ser armazenados para transmissão futura. Se a fila estiver cheia e um pacote for recebido da camada superior, o T-MAC vai eliminar o pacote e enviar uma mensagem a MAC_2_NETWORK_FULL_BUFFER.
- **maxTxRetries = 1 -** É o número de tentativas de transmissão de um único pacote unicast que o TMAC vai realizar. Uma transmissão é considerada bem sucedida só se um pacote de ACK for recebido do nó de destino. O envio de um pacote RTS é também considerada uma tentativa de transmissão. De notar que este parâmetro não se aplica a pacotes de broadcast.
- **frameTime = 610** - É a duração de cada trama para todos os nós (em milisegundos). Os nós tentam sincronizar o início e fim de cada trama com o esquema global (com a possibilidade de haver mais esquemas). De notar que isto se refere à duração de toda a trama, as partes activas e inactivas de cada trama são determinadas dinamicamente e individualmente para cada nó.
- **contentionPeriod = 10 -** A duração do intervalo de contenção, em milissegundos, para cada tentativa de transmissão. O maior efeito deste parâmetro é no evitar interferências de transmissão por parte de nós vizinhos.

- **listenTimeout = 15 -** A duração do período de escuta, em milisegundos. Este parâmetro define a quantidade de tempo que é necessário passar sem actividade no canal sem fios para que o nó entre no modo sleep na actual trama.
- **waitTimeout = 5 -** Define o tempo de expiração de espera de uma resposta de outro nó (em milisegundos). Esta resposta pode ser um pacote CTS ou ACK. Se nenhuma resposta for recebida após este intervalo de tempo, então a tentativa de transmissão é considerada falhada e o respectivo contador é decrementado.
- **resyncTime = 40 -** O intervalo entre o envio de pacotes de sincronização (em segundos). O valor deste parâmetro está directamente relacionado com a deriva do relógio dos nós na rede simulada. O valor adequado para o modelo actual da deriva do relógio do Castalia é de 40 s. De notar que os valores originais do artigo do S-MAC (que o T-MAC herda) é 180s.
- **allowSinkSync = 1 -** Se o valor for definido para 1 (verdadeiro), o T-MAC vai tentar extrair informação das camadas superiores (aplicação) de forma a determinar se o nó actual é um sink. Este parâmetro permite aos sink evitar o intervalo de contenção quando criam um esuqema de sincronização para a rede, permitindo sincronização mais rápida e, consequentemente, melhor débito (especialmente que os pacotes precisam de ser enviados cedo na simulação).
- **useFrts = 0 -** Este parâmetro refere-se ao uso do Future Request to Send (FRTS) como definido pelos criadores do algoritmo T-MAC. Na actual implementação do TMAC este parâmetro não é suportado.
- **useRtsCts = 0 -** Este parâmetro é uma característica adicional que não foi proposta originalmente no T-MAC. Permite desligar o mecanismo RTS/CTS limitando as trocas a pacotes entre os nós a dados e ACK. Útil para situações em que apenas pacotes pequenos são trocados, tornando desnecessário reservar o canal para a transmissão (dado que a reserva do canal tomaria mais tempo que a própria transmissão).
- **disableTAextension = 0 -** Outro parâmetro extra que permite desactivar a extensão de expiração do tempo. Dado que a flexibilidade do período activo é a grande melhoria do T-MAC face ao S-MAC, desabilitando-o e definindo o intervalo de escuta apropriado (10% da trama completa) implementa-se o S-MAC.

4.6 Sumário e Conclusões

Neste Capítulo discutiram-se as características de alguns simuladores, identificando-se a razão da escolha do Castalia. Especificaram-se os protocolos recorrendo a diagramas de transição de estados (MEF). Por norma, inicialmente deve-se estudar o desempenho dos protocolos através de um simulador adequado, passando depois à sua análise e optimização recorrendo a implementações concretas numa RSSF real.

Capítulo 5
Simulações

5.1 Introdução

5.2 Cenários

Considerou-se que os nós sensores se encontram a medir a humidade de um terreno com uma área de 1 ha para plantação. Os nós sensores estão distanciados de 20m entre eles, conforme representado na Figura 5.1. A Tabela 5.1 representa as coordenadas dos nós. Este cenário foi utilizado com a aplicação *valueReporting*.

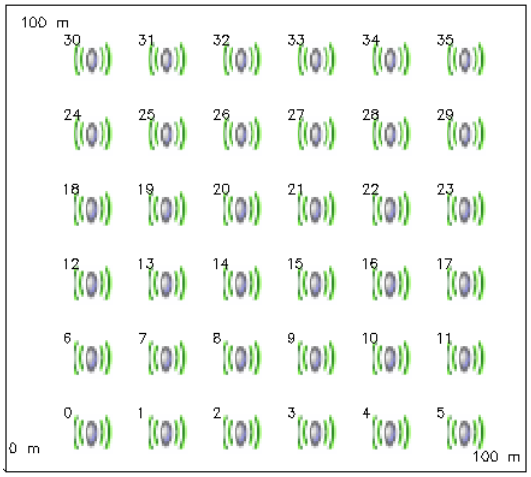


Figura 5.1 – Distribuição dos nós sensores

Apresenta-se na Tabela I.1 em Anexo o ficheiro gerado ao compilar a aplicação *valueReporting* onde se pode ver todos os ficheiros e valores de variáveis utilizados nesta simulação. Na Tabela I.2 em Anexo apresenta-se os parâmetros definidos pelo utilizador para a aplicação valueReporting no Castalia. Os principais parâmetros utilizados são:

- Tempo máximo de Simulação (sim-time-limit) = 600s;
- Canal com interferência segundo o modelo realístico (ficheiro WChanel_Realistic.ini);
- Rádio CC2420 (ficheiro CC2420.ini.);
- Encaminhamento de árvore simples (ficheiro Routing_simpleTree.ini;
- Tamanho máximo do pacote (maxAppPacketSize) = 30;
- Nível de potência de transmissão (txPowerLevelUsed) = 2.

Tabela 5.1 – Localização dos nós em coordenadas rectangulares para a aplicação valueReporting

Nó ID	*x*	*y*
0	0	0
1	20	0
2	40	0
3	60	0
4	80	0
5	100	0
6	0	20
7	20	20
8	40	20
9	60	20
10	80	20
11	100	20
12	0	40
13	20	40
14	40	40
15	60	40
16	80	40
17	100	40
18	0	60
19	20	60
20	40	60
21	60	60
22	80	60
23	100	60
24	0	80
25	20	80
26	40	80
27	60	80
28	80	80
29	100	80
30	0	100
31	20	100

Nó ID	*x*	*y*
32	40	100
33	60	100
34	80	100
35	100	100

Para as aplicações *valuePropagation* e *simpleAgregation* as posições dos nós são as indicadas na Tabela 5.2.

Tabela 5.2 – Localização dos nós em coordenadas rectangulares para a aplicação *valuePropagation* e *simpleAgregation*

Nó ID	*x*	*y*
0	0	0
1	0	25
2	0	50
3	20	0
4	20	25
5	20	50
6	40	0
7	40	25
8	40	50

Para as aplicações com mobilidade (mobilityTest1, mobilityTest2, mobilityTest3) as posições iniciais dos nós são apresentadas nas Tabelas 5.3, 5.4 e 5.5.

Tabela 5.3 – Localização dos nós em coordenadas rectangulares para a aplicação *mobilityTest1*

Nó ID	*x*	*y*
0	0	0
1	50	50
2	150	150

Tabela 5.4 – Localização dos nós em coordenadas rectangulares para a aplicação *mobilityTest2*

Nó ID	*x*	*y*
0	10	50
1	0	50
2	5	0

Tabela 5.5 – Localização dos nós em coordenadas rectangulares para a aplicação *mobilityTest3*

Nó ID	*x*	*y*
0	20	50
1	0	50
2	22	0

5.3 Resultados

A Figura 5.2 apresenta a variação do consumo de energia em cada nó com o *dutycycle* para o protocolo MAC Tunable.

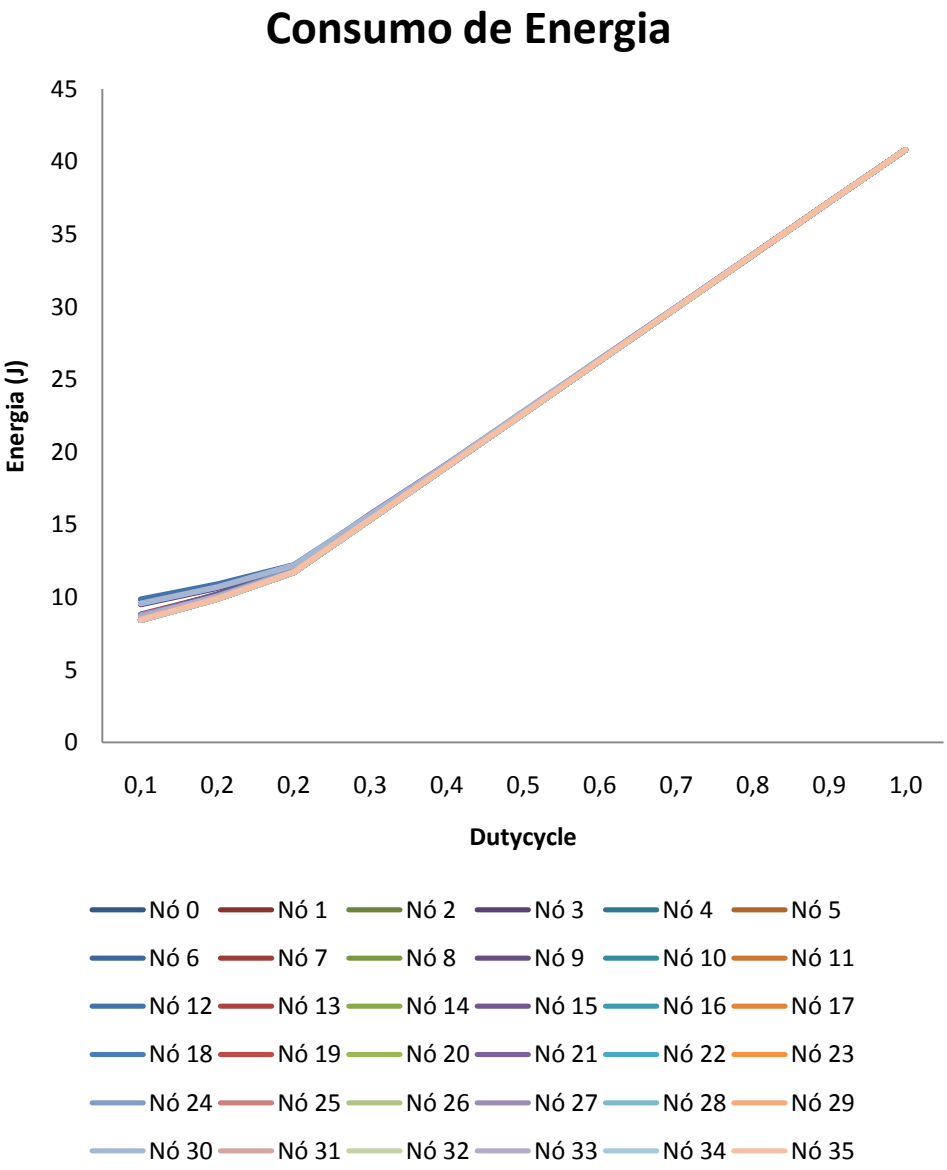


Figura 5.2 – Consumo de energia em cada nó variando o dutycycle do protocolo MAC Tunable

Interessa comparar o desempenho energético do MAC Tunable com os protocolos S-MAC, T-MAC, MAC Bypass e MAC Just Carrier Sense. Verifica-se que os protocolos S-MAC e T-MAC são mais eficientes do ponto de vista energético que o MAC Tunable.

Exceptuando picos para alguns nós nos protocolos MAC Tunable e T-MAC o consumo de energia é regular para todos os nós sensores.

Conforme se pode verificar na Tabela C. o consumo de energia é igual para os protocolos MAC Bypass e MAC Just Carrier Sense.

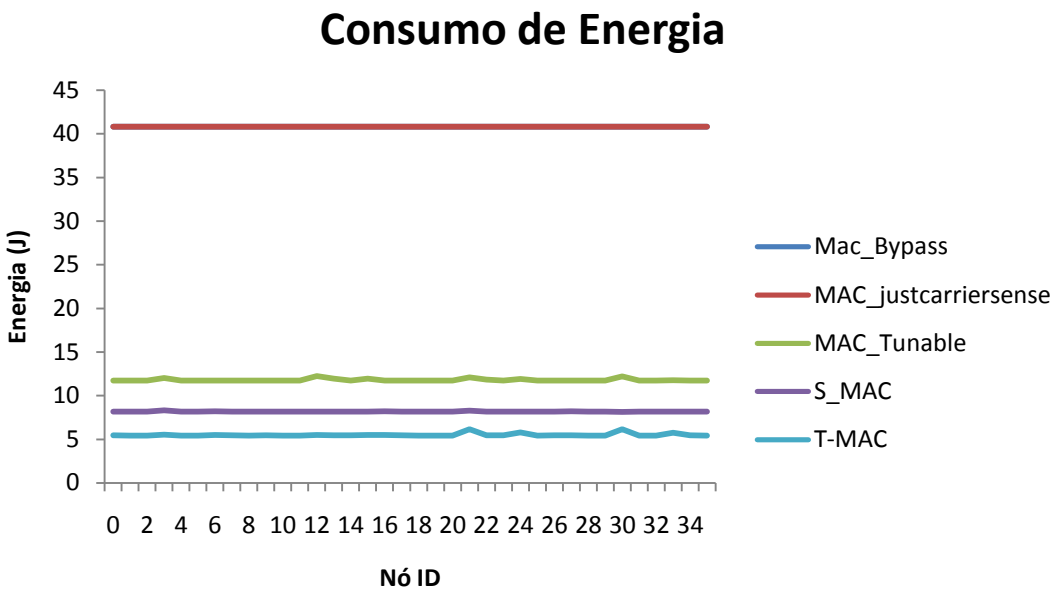


Figura 5.3 – Consumo de energia em cada nó por protocolo para a aplicação no Castalia

Fazendo variar *ListenTimeout* (entre 1, 15, 30 e 300 ms) no protocolo T-MAC obtém-se valores diferentes de consumo de energia, Figura 5.4.

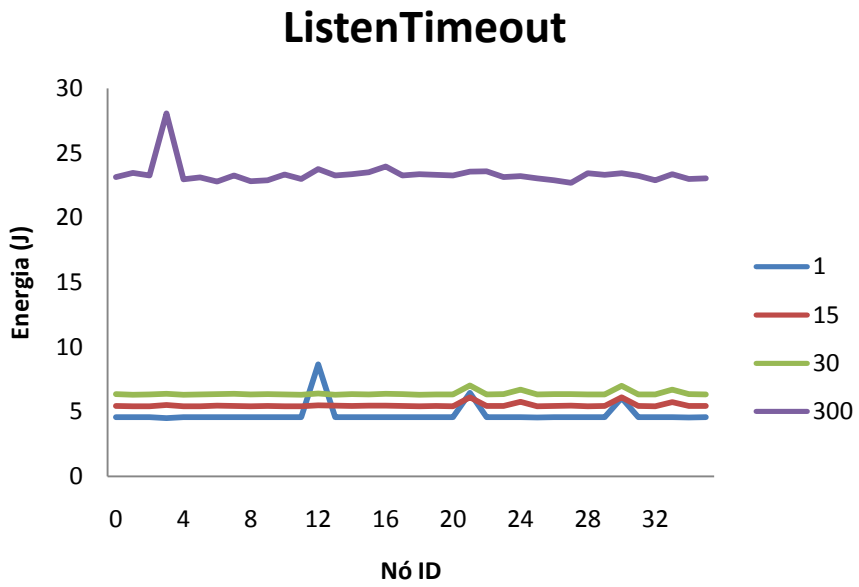


Figura 5.4 – Consumo de energia em cada nó variando o tempo de escuta no protocolo T-MAC da aplicação valueReporting no Castalia

A Tabela correspondente de valores do consumo de energia é apresentada no Anexo G.

Quanto se reduz o período de escuta (*ListenTimeout*) reduz-se o consumo de energia. No entanto, note-se que a diminuição extrema do período de escuta pode causar o aumento das mensagens não entregues.

A aplicação simpleAgregation do Castalia transmite os valores recolhidos pelo nó sensor para os *cluster heads* que por sua vez reencaminham para o nó *sink*.

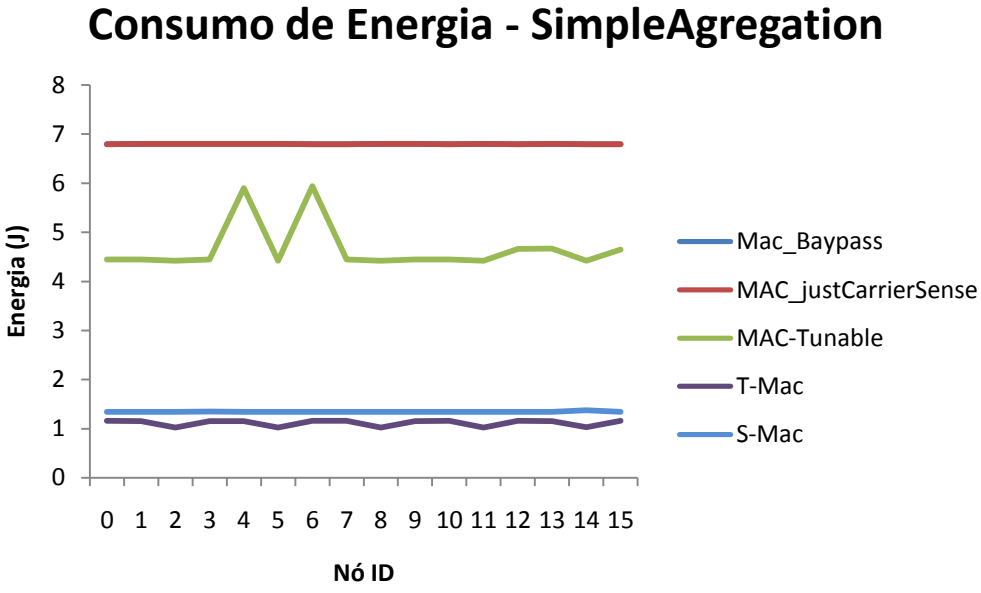


Figura 5.5 – Consumo de energia para vários protocolos na aplicação SimpleAgregation do Castalia.

Conclui-se que o protocolo T-MAC tem um desempenho energético ligeiramente mais favorável relativamente aos outros protocolos simulados (nomeadamente o S-MAC) para as várias aplicações sem mobilidade.

As aplicações com mobilidade no Castalia começam inicialmente com os nós sensores em posições iniciais deslocando-se depois esses nós sensores segundo uma função do tempo definida no ficheiro node_locations.ini. No *MobilitiTest1* o nó 0 é movido linearmente a partir da sua posição inicial indicada na Secção 5.2. O consumo de energia é igual para os protocolos MAC Bypass e MAC Just Carrier Sense ficando as suas rectas sobrepostas. Verifica-se que, com mobilidade, o resultados mais favoráveis em termos energéticos ocorrem com o protocolo S-MAC (cerca de 1 J/nó)

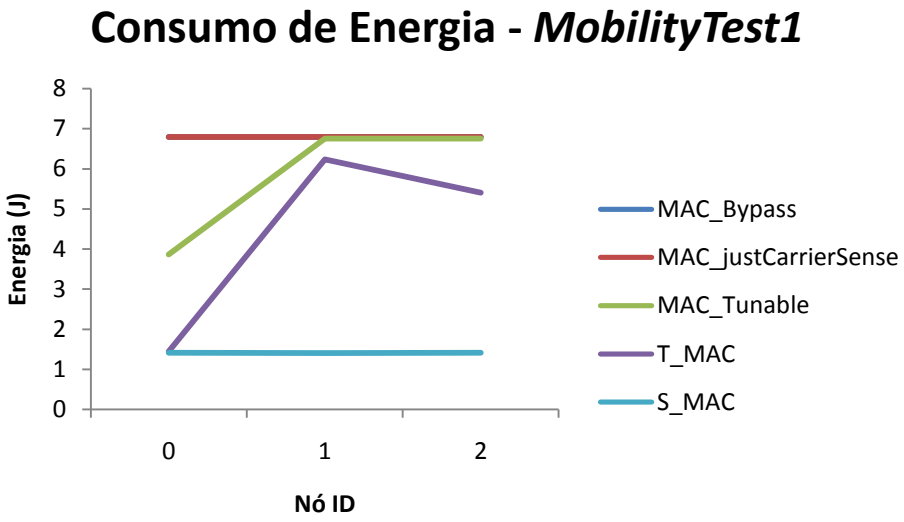


Figura 5.6 – Consumo de energia por nó para vários protocolos na aplicação *MobilityTest1* do Castalia

No *MobilityTest2* e no *MobilityTest3* o nó 2 é movido linearmente a partir da sua posição inicial indicada em Secção 5.2. Na aplicação MobilityTest2 não foi possível efectuar a simulação para os protocolos T-MAC e S-MAC. O MAC Bypass tem um comportamento energético muito parecido com o MAC Just Carrier Sense sendo estes os protocolos com um comportamento mais favorável em termos energéticos.

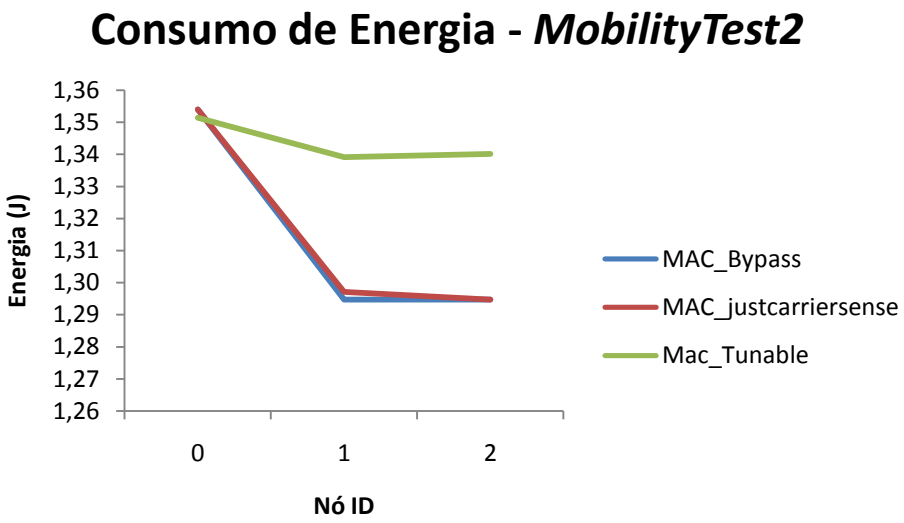


Figura 5.7 – Consumo de energia por nó para vários protocolos na aplicação *MobilityTest2* do Castalia

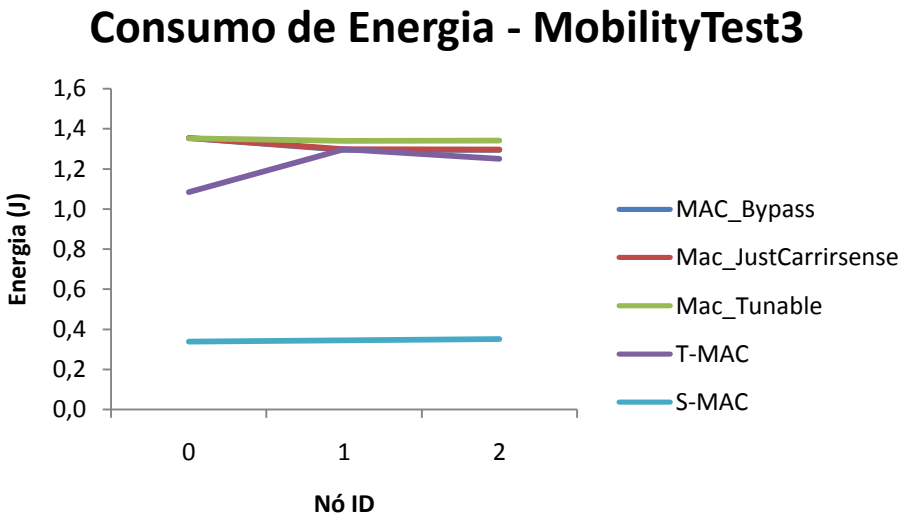


Figura 5.8 – Consumo de energia por nó, para vários protocolos na aplicação Mobility3 do Castalia

Na aplicação *MobilityTest3* os protocolos *MAC Just Carrier Sense* e o *MAC Bypass* encontram-se *novamente sobrepostos. O S*-MAC apresenta um consumo de energia mais favorável com cerca de 0,34 J/nó, sendo um valor bastante reduzido. Interessa também efectuar experiências menos elaboradas, onde se varia um parâmetro importante do protocolo T-MAC: *contentionPeriod.* A Figura 5.9 representa a variação do consumo de energia com o *contentionPeriod*.

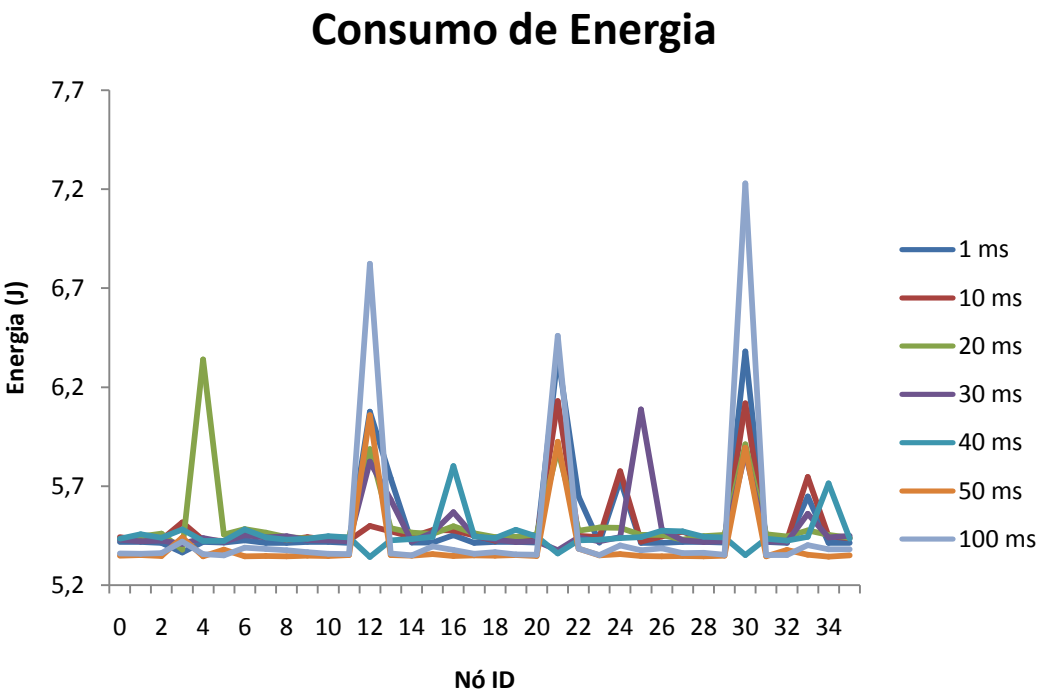


Figura 5.9 – Consumo de energia em cada nó variando o *contentionPeriod* do protocolo T-MAC

O mesmo gráfico é apresentado nas Figuras 5.10 a 5.18 a 3 dimensões e para cada valor do período de contenção. A Tabela com os valores de energia é apresentado no anexo H.

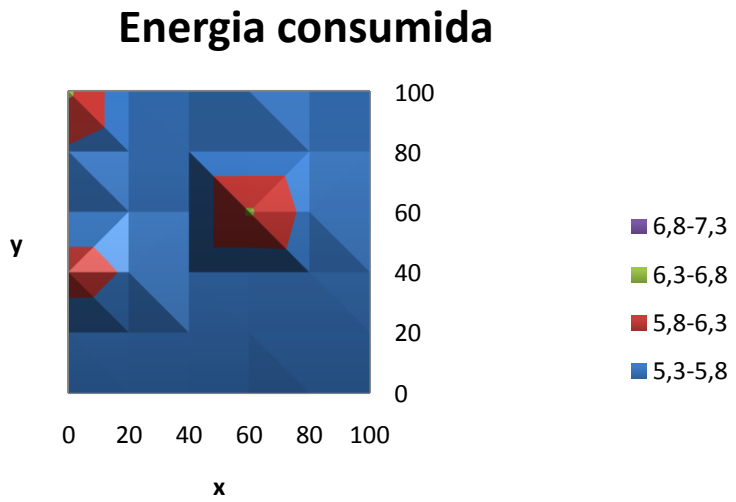


Figura 5.10 – Consumo de energia em cada nó no protocolo T-MAC (*contentionPeriod* = 1)

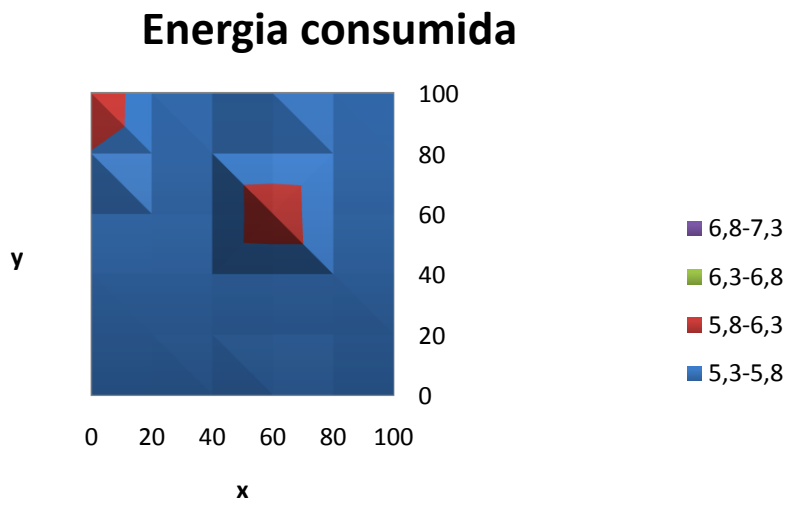


Figura 5.11 – Consumo de energia em cada nó no protocolo T-MAC (*contentionPeriod* = 10)

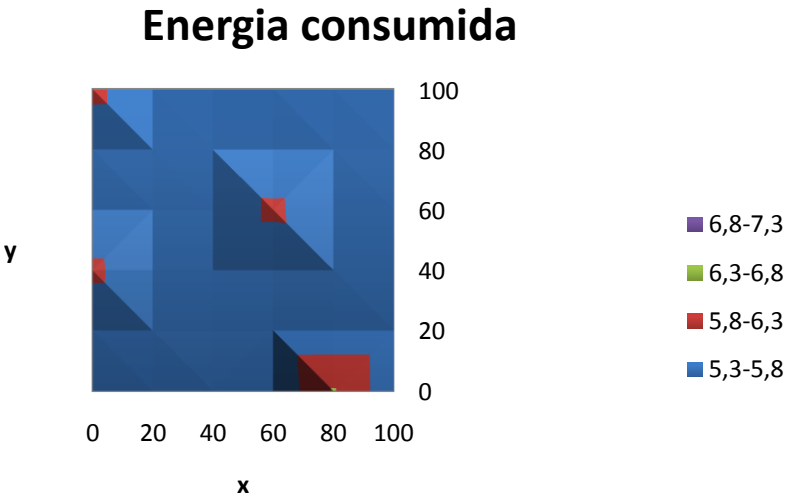


Figura 5.12 – Consumo de energia em cada nó no protocolo T-MAC (*contentionPeriod* = 20)

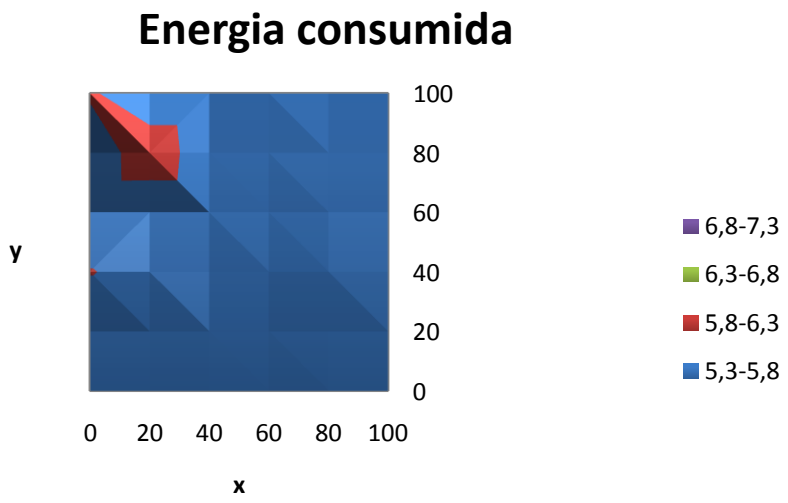


Figura 5.13 – Consumo de energia em cada nó no protocolo T-MAC (*contentionPeriod* = 30)

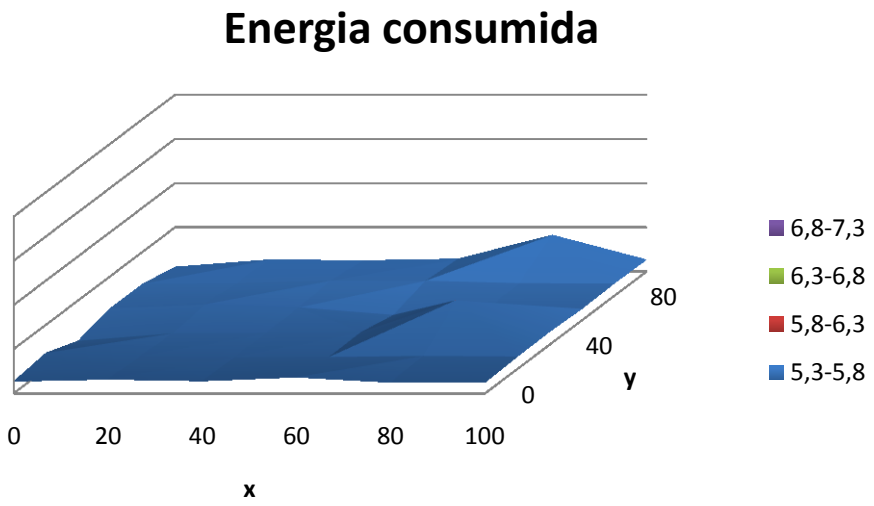


Figura 5.14 – Consumo de energia em cada nó no protocolo T-MAC (*contentionPeriod* = 40)

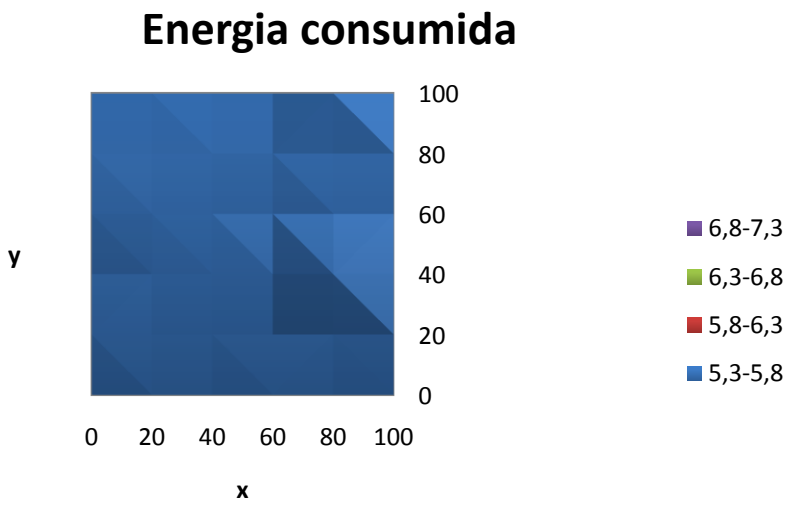


Figura 5.15 – Consumo de energia em cada nó no protocolo T-MAC (*contentionPeriod* = 40)

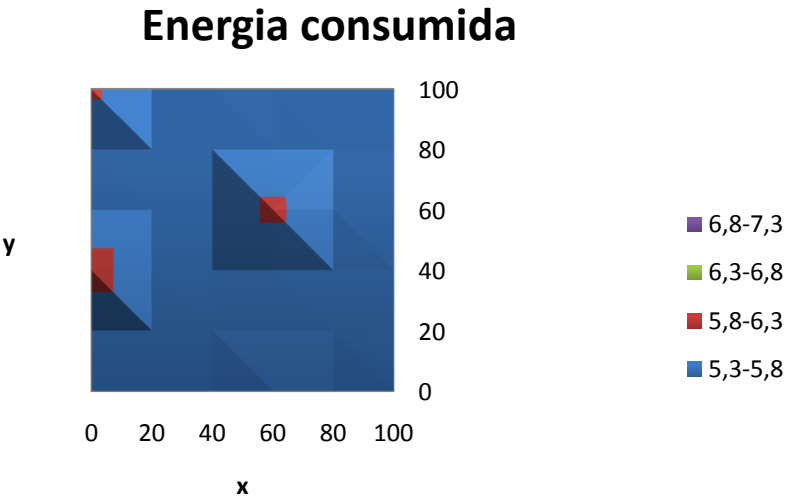


Figura 5.16 – Consumo de energia em cada nó no protocolo T-MAC (*contentionPeriod* = 50)

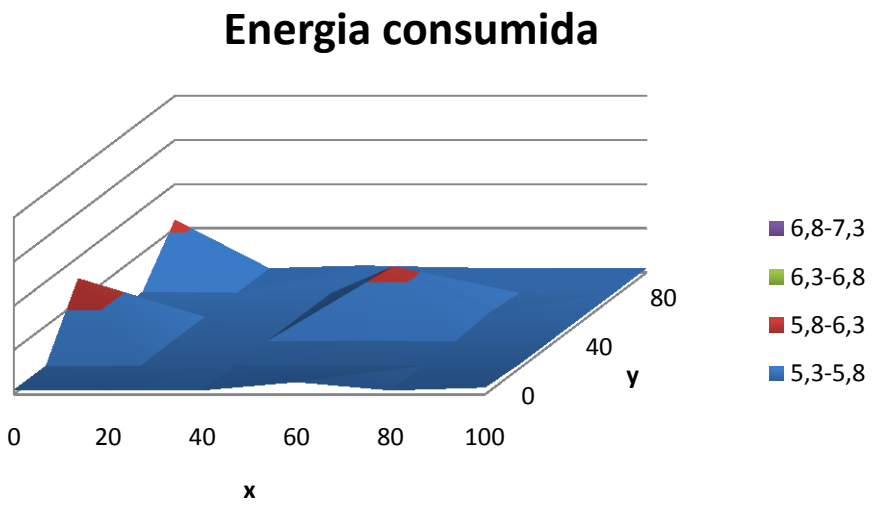


Figura 5.17 – Consumo de energia em cada nó no protocolo T-MAC (*contentionPeriod* = 50)

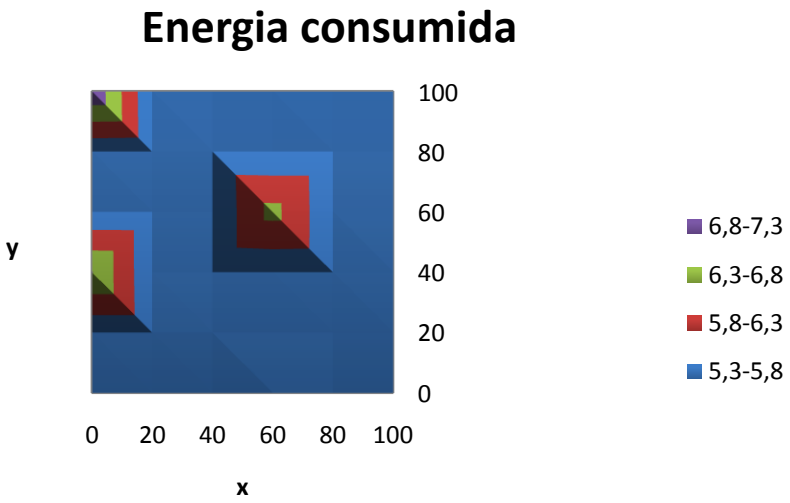


Figura 5.18 – Consumo de energia em cada nó no protocolo T-MAC (*contentionPeriod* = 100)

Podemos verificar que os nós 12, 21 e 30 são os que apresentam picos mais elevados de consumo de energia e habitualmente, estes nós foram seleccionados para *cluster heads*. Apenas se verifica um comportamento diferente para um *contentionPeriod* = 40.

Apresentam-se de seguida, nas Figura 5.19 e Figura 5.20 gráficos com o desvio padrão e a média de consumo de energia que nos auxiliam na obtenção do valor ideal para o período de contenção, de acordo com a perspectiva de igual tempo de vida entre nós (Figura 5.19) ou de menor consumo de energia da rede (Figura 5.20).

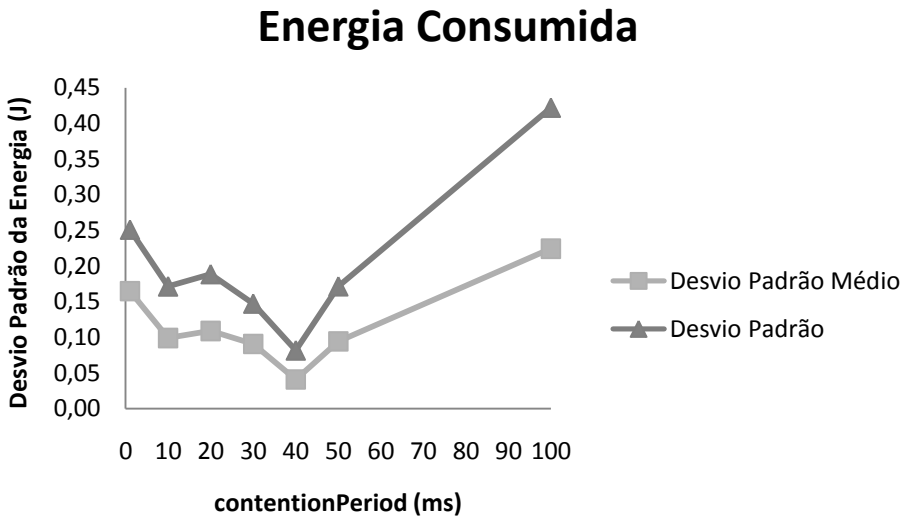


Figura 5.19 – Desvio Padrão do consumo de energia ao longo da variação do período de contenção

Para sistemas críticos em que é necessário que todos os nós cumpram a sua função o *contentionPeriod* deve ser de 40 ms, valor para o qual o desvio padrão da energia consumida é inferior.

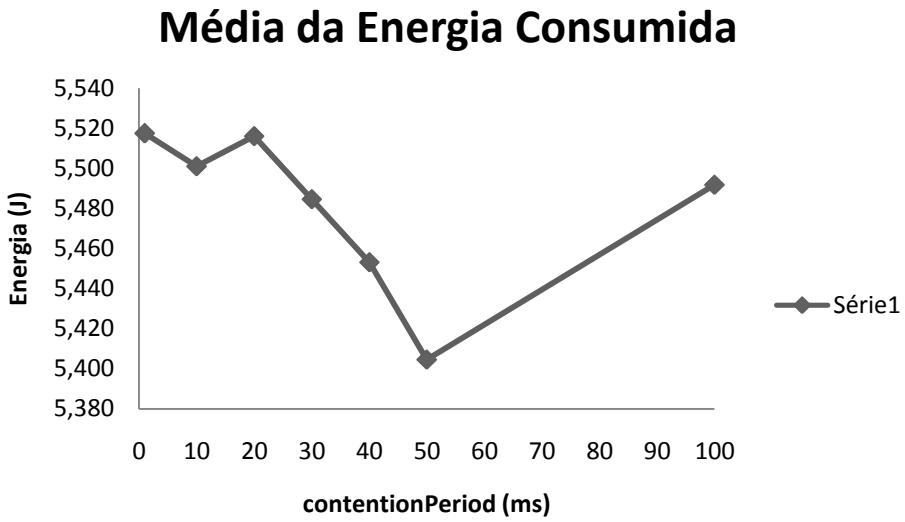


Figura 5.20 – Média aritmética da energia consumida ao longo da variação do período de contenção

Para sistemas em que a prioridade seja a poupança de energia da rede, deverá ser utilizado o *contentionPeriod* de 50ms.

5.4 Sumário e Conclusões

Neste Capítulo abordam-se protocolos MAC para RSSFs tendo como objectivo optimizar a eficiência energética e maximizar o tempo de vida da rede, de acordo com a sua aplicação.

A variação de vários parâmetros ao mesmo tempo ao invés de apenas um, assim como a criação de novos estados ou comportamentos do protocolo, poderá levar a resultados mais precisos de acordo com os objectivos e cenário pretendidos.

Em primeiro lugar, verificou-se que quanto maior o período de escuta (*listenTimeout*) maior será o consumo de energia, No entanto, deve-se ter em conta que a diminuição excessiva desse período irá ter consequenciais negativas no desempenho do sistema tal como o aumento de mensagem não entregues. Para trabalho futuro, poderá ser analisada a evolução do número das mensagens não entregues com a diminuição do período de escuta tentando estabelecer-se um compromisso entre o tempo de entrega das mensagens e o consumo de energia.

Nas aplicações com mobilidade, o protocolo S-MAC é o que apresenta uma eficiência energética superior. Nas aplicações sem mobilidade, nas experiências realizadas considerando uma grelha com 36 nós, conclui-se que o protocolo T-MAC tem um desempenho energético ligeiramente mais favorável relativamente aos outros protocolos simulados (nomeadamente o S-MAC) para as várias aplicações.

Neste cenário com 36 nós, caso se esteja a utilizar uma RSSF, numa aplicação prática em que todos os nós têm uma função individual "crítica", por exemplo, localização de membros de uma manada, não sendo possível que o sistema funcione correctamente assim que ocorre a primeira falha de um nó sensor, o período de contenção deverá ser de 40 ms, valor para o qual o desvio padrão da energia consumida é inferior. Este valor foi escolhido por corresponder a um mínimo para ambos os valores dos desvios padrão analisados, o que significa que ao longo da rede não existem nós com um consumo de energia muito superior aos restantes. Não é por isso de esperar que um nó falhe relativamente cedo quando comparado com a vida útil dos restantes nós.

Caso a função individual de cada nó sensor não seja "crítica" para o sistema, pode-se utilizar o valor do sensor mais próximo para saber se é necessário regar o terreno. Este é o caso do cenário do sistema de rega apresentado neste Capítulo, onde a falha de um nó sensor apenas impediria a leitura de humidade do solo numa certa região do terreno, mas permitiria a continuação do bom funcionamento do sistema, visto que o nó mais próximo se encontra a 20m e é geralmente bastante improvável que o valor da

humidade no solo seja muito diferente entre nós vizinhos. Neste caso, o valor do período de contenção óptimo é dado pelo mínimo da média de consumo de energia da rede (50 ms). Sendo a rede mais eficiente em termos energéticos, será possível ter a maior parte dos nós sensores em funcionamento durante um período de vida longo.

Capítulo 6
Conclusões e Trabalho futuro

6.1 Conclusões

Nesta dissertação abordam-se protocolos MAC para RSSFs tendo como objectivo optimizar a sua eficiência energética e maximizar o tempo de vida da rede, de acordo com a sua aplicação.

As RSSFs possuem um vasto campo de aplicações existindo cada vez mais programas que facilitam a sua utilização, tanto para instalação nos nós sensores como para simulação das RSSFs, ou mesmo para conversão entre ambos, existindo um vasto leque de projectos em curso a nível mundial. Em relação aos efeitos da radiação no corpo humano é de notar que a radiação acima de certos valores tem efeitos negativos no corpo humano, tal como o aquecimento de certas zonas do corpo, não existindo porém estudos que comprovem, sem margem para dúvidas, alguns dos efeitos secundários nefastos que se acreditam serem causados pela proximidade do corpo humano com fontes emissoras de radiação electromagnética, como acontece nas WBAN.

A norma IEEE 802.15.4 é bastante robusta e funcional para ser utilizada em RSSFs. A aliança ZigBee adoptou um sub-conjunto de especificações da norma IEEE 802.15.4 e reprenta actualmente uma boa solução para dispositivos que não necessitam de banda larga mas apenas de baixo consumo de energia como, por exemplo, sensores e controladores [8]. O ZigBee tem como alvo aplicações ainda não utilizadas pelo

Bluetooth ou por outra norma sem fios, ou seja, estas duas tecnologias poderão ser o complemento uma da outra para uma solução em larga escala.

O simulador Castalia é uma ferramenta que auxilia a análise de características ou comportamentos das RSSF. Inicialmente, deve-se estudar o desempenho dos protocolos através de um simulador adequado, passando depois á sua análise e optimização, a ser efectuada num protocolo implementado numa RSSFs real.

A variação simultânea de vários parâmetros, ao invés de apenas um, assim como a criação de novos estados ou comportamentos do protocolo, poderão levar a resultados mais precisos de acordo com os objectivos e cenário pretendidos.

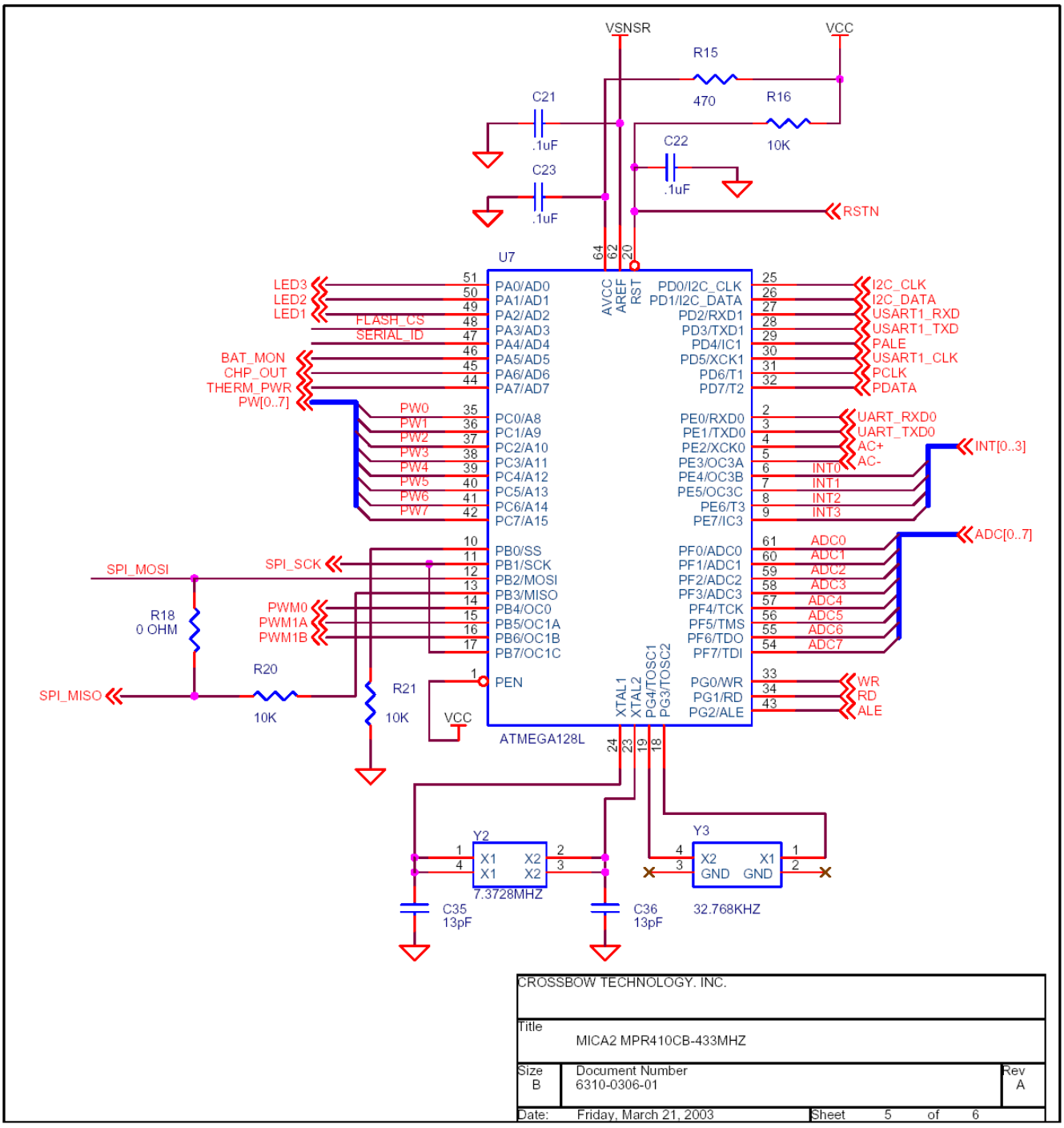
Das simulações realizadas, em primeiro lugar, verificou-se que quanto maior o período de escuta (*listenTimeout*) maior será o consumo de energia. No entanto, deve-se ter em conta que a diminuição excessiva desse período irá ter consequenciais negativas no desempenho do sistema tal como o aumento de mensagem não entregues. Nas aplicações com mobilidade o protocolo S-MAC é o que apresenta uma eficiência energética superior. Nas aplicações sem mobilidade, das experiências realizadas com 36 nós dispostos regularmente, em grelha, conclui-se que o protocolo T-MAC tem um desempenho energético ligeiramente superior.

Neste último cenário, caso se esteja a utilizar uma RSSF numa aplicação prática em que todos os nós têm uma função individual "crítica", deve-se escolher um valor para o período de contenção que corresponda um valor mais reduzido do desvio padrão da energia consumida. Caso a função individual de cada nó sensor não seja "crítica" para o sistema (mas apena o funcionamento global da RSSF seja essencial como um todo), o valor óptimo do período de contenção será dado pelo mínimo da média de consumo de energia da rede. No cenário considerado, os valores óptimos para o período de contenção são 40 e 50 ms, para os casos onde missão individual de cada nó é "crítica" ou "não crítica", respectivamente.

6.2 Proposta para trabalho futuro

Como trabalho futuro propõe-se fazer variar outros parâmetros e estudar outras para características além do consumo de energia, que poderão ser críticos para o desempenho que se pretende atingir. Por exemplo, poderá ser analisada a evolução do número das mensagens não entregues com a diminuição do período de escuta, tentando-se estabelecer um compromisso entre o tempo de entrega das mensagens e o consumo de energia.

A. Anexo – Microcontrolador ATMega 128L



B. Anexo – Mensagens do Castalia

As Tabelas B.1 a B.14 representam as mensagens do Castalia cujo diagrama de transição de estados foi representado na Secção 4.

Tabela B.1 – APP_NODE_STARTUP

APP_NODE_STARTUP [13]	
Senders	Módulo de Aplicação
Triggered messages	MAC_2_RADIO_ENTER_LISTEN, MAC_SELF_SET_RADIO_SLEEP
Direct State transitions	-
Indirect State transitions	-
Handling Description	Esta mensagem é enviada pelo módulo de aplicação para arrancar/ligar o resto dos módulos do nó. É enviado a si próprio, ao Gestor de Dispositivos Sensores (Sensor Device Manager) e ao sub-módulo do Módulo de Comunicação. O Mac passa então para activado e por sua vez envia uma mensagem ao módulo de Rádio para que este fique no "estado de escuta". Finalmente se um *duty cycle* for especificado na lista de parâmetros do omnetpp.ini, calendariza uma mensagem automática para que depois de decorrido o intervalo de tempo "listenInterval" o Módulo de Rádio passe para o "estado de repouso"

Tabela B.2 – APP_DATA_PACKET

APP_DATA_PACKET [13]	
Senders	Módulo de Aplicação
Triggered messages	MAC_FRAME_SELF_PUSH_TX_BUFFER, MAC_SELF_INITIATE_TX
Direct State transitions	-
Indirect State transitions	A mensagem desencadeada com o tipo

	MAC_SELF_INITIATE_TX pode alterar por sua vez (também de forma indirecta) o estado para MAC_STATE_TX ou para MAC_STATE_CARRIER_SENSING.
Handling Description	Alguns dos parâmetros do módulo Mac são: - numTx, - probTx, - randomTxOffset and - reTxInterval. A acção descrita a seguir será realizada na maioria das vezes a "numTx". Cada um das tentativas "numTx" tem uma probabilidade de acontecer igual a "probTx". **Acção**: Obtém o pacote de dados recebidos da Aplicação e encapsula-o numa frame do Mac. Esta frame não será puxada directamente a partir do *buffer* do Mac, sendo em vez disso agendada uma mensagem automática: MAC_FRAME_SELF_PUSH_TX_BUFFER Muitas vezes (se não mesmo a maioria das vezes), quando agendamos mensagens é necessário especificar um atraso. O atraso é a quantidade de tempo que a mensagem terá de esperar antes de ser entregue ao módulo especificado (neste caso será o módulo Mac). Aqui o atraso para esta acção é uma fracção aleatória dos parâmetros ["randomTxOffset" + "reTxInterval"]. A *frame* será empurrada no *buffer* depois do intervalo de tempo ["randomTxOffset" + "reTxInterval"] quando o Mac recebe a auto-mensagem (para si mesmo). Finalmente o módulo Mac agenda a inicialização do TX, logo após a *frame* ser empurrada no *buffer*. Não se empurra simplesmente a *frame* directamente para o *buffer* devido às características erróneas do circuito

	integrado do rádio dos nós sensores. Usando os parâmetros numTx, probTx, randomTxOffset e reTxInterval nós podemos controlar a quantidade de vezes, a probabilidade, e o espaçamento temporal que o MAC utilizará para encaminhar uma mensagem recebida pelo módulo de Aplicação. Nota: Usando os parâmetros seguintes provocar-se-á o Mac para enviar, somente uma vez, o pacote da aplicação: numTx = 1, probTx = 1, randomTxOffset = 0, reTxInterval = 0 (não é obrigatório).

Tabela B.3 – MAC_SELF_CHECK_TX_BUFFER

MAC_SELF_CHECK_TX_BUFFER* [13]	
Senders	Módulo Mac
Triggered messages	MAC_FRAME (ambos os beacons e dados), MAC_2_RADIO_ENTER_TX, MAC_SELF_CHECK_TX_BUFFER, MAC_SELF_SET_RADIO_SLEEP
Direct State transitions	-
Indirect State transitions	-
Handling Description	MAC verifica o próprio *buffer*. Caso o *buffer* não esteja vazio: Se o estado for MAC_STATE_CARRIER_SENSING ou MAC_STATE_EXPECTING_RX então não faz nada. Se o estado for MAC_STATE_DEFAULT, então troca para o estado MAC_STATE_TX. A seguir aparece a primeira *frame* (no topo) no do seu *buffer*. Se na lista de parâmetros do ficheiro omnetpp.ini for especificado o *duty cycle* então o módulo Rádio vai enviar alguns *beacons* para os nós vizinhos a fim de os despertar para que recebam os

	dados reais. A *frame* de dados reais é enviada para o módulo Rádio logo após o último *beacon* (como o intervalo de tempo apropriado). Finalmente o Mac agenda uma nova auto-mensagem (a si próprio) MAC_SELF_CHECK_TX_BUFFER para repetir as acções acima. O protocolo Mac irá continuar a repetir estes passos (enviar para o rádio e agendar uma nova auto-mensagem para verificar novamente o *buffer*) até o *buffer* ficar vazio. Caso o *buffer* esteja vazio: O estado do protocolo Mac muda para MAC_STATE_DEFAULT. Caso o parâmetro *duty cycle* tenha sido especificado na lista do omnetpp.ini, então agenda uma auto-mensagem para colocar "listenInterval" do módulo de rádio, no "modo de repouso".

Tabela B.4 – MAC_SELF_SET_RADIO_SLEEP

MAC_SELF_SET_RADIO_SLEEP [13]	
Senders	MacModule
Triggered messages	MAC_2_RADIO_ENTER_SLEEP, MAC_SELF_WAKEUP_RADIO
Direct State transitions	-
Indirect State transitions	-
Handling Description	É possível receber uma mensagem com um tipo somente se o *duty cycle* for especificado na lista de parâmetros do ficheiro omnetpp.ini Se o protocolo Mac estiver no estado MAC_STATE_DEFAULT então uma mensagem de controlo é enviada para o módulo de Rádio, tendo uma instrução para entrar no "modo de repouso". Mac também agenda uma auto-mensagem

	(MAC_SELF_WAKEUP_RADIO) com atraso "sleepInterval". De notar que se o Mac estiver a transmitir ou á espera de receber um pacote de dados, depois de ter recebido alguns beacons, irá ignorar qualquer mensagem do tipo MAC_SELF_SET_RADIO_SLEEP

Tabela B.5 – MAC_SELF_WAKEUP_RADIO

MAC_SELF_WAKEUP_RADIO [13]	
Senders	Módulo Mac
Triggered messages	MAC_2_RADIO_ENTER_LISTEN, MAC_SELF_SET_RADIO_SLEEP
Direct State transitions	-
Indirect State transitions	-
Handling Description	É possível receber uma mensagem com um tipo somente se o *duty cycle* é especificado na lista de parâmetros omnetpp.ini. Se o estado do protocolo Mac é MAC_STATE_DEFAULT então a mensagem de controlo é enviada para o módulo Mac que contem a instrução de entrar no "modo de repouso". O Mac também agenda uma auto-mensagem (MAC_SELF_WAKEUP_RADIO) com atraso "sleepInterval". De notar que se o Mac estiver a transmitir ou á espera de receber uma *frame* de dados depois de ter recebido alguns *beacons*, irá ignorar qualquer mensagens do tipo MAC_SELF_WAKEUP_RADIO.

Tabela B.6 – MAC_SELF_INITIATE_TX

MAC_SELF_INITIATE_TX [13]	
Senders	Módulo Mac
Triggered messages	MAC_SELF_PERFORM_CARRIER_SENSE (se a

	função *carrier sense* estiver definida como verdade no ficheiro omnetpp.ini) MAC_SELF_CHECK_TX_BUFFER (se a função *carrier sense* estiver definida como falsa), MAC_SELF_SET_RADIO_SLEEP (se o *buffer* estiver vazio).
Direct State transitions	Se o buffer do Mac estiver vazio, os estados do módulo Mac mudam para MAC_STATE_DEFAULT
Indirect State transitions	MAC_STATE_CARRIER_SENSING (se *carrier sense* for verdade), MAC_STATE_TX (se *carrier sense* não for verdade).
Handling Description	O módulo Mac agenda o estado MAC_SELF_INITIATE_TX quando recebe um pacote de dados a partir do módulo Aplicação (que precisa de ser transmitido para a rede) ou quando não recebe a partir do módulo de Rádio nenhuma frame de dados para "sleepInterval" depois do primeiro beacon. Processamento: Quando o módulo Mac processa esta mensagem primeiramente realiza *carrier sense* (se estiver activo) e depois disso verifica se o *buffer* está vazio. Se estiver, não faz nada a não ser agendar uma auto-mensagem do tipo MAC_SELF_SET_RADIO_SLEEP para colocar o rádio no "modo de repouso", depois do período "listenInterval". No caso do *buffer* estar vazio, Mac agenda uma auto-mensagem do tipo MAC_SELF_SET_RADIO_SLEEP para processar se o *duty cycle* estiver especificado na lista de parâmetros do ficheiro omnetpp.ini. No caso do *buffer* estar vazio, Mac agenda uma auto-mensagem do tipo MAC_SELF_SET_RADIO_SLEEP

	para processar se um *duty cycle* for especificado na lista de parâmetros do ficheiro omnetpp.ini.

Tabela B.7 – MAC_FRAME_SELF_PUSH_TX_BUFFER

MAC_FRAME_SELF_PUSH_TX_BUFFER [13]	
Senders	Módulo Mac
Triggered messages	-
Direct State transitions	-
Indirect State transitions	-
Handling Description	O módulo Mac simplesmente puxa a mensagem recebida no seu buffer. A mensagem recebida é actualmente MAC_FRAME que tem o seu tipo de campo definido para MAC_FRAME_SELF_PUSH_TX_BUFFER Se o *buffer* está cheio não faz nada.

Tabela B.8 – MAC_FRAME

MAC_FRAME* [13]	
Senders	Módulo Radio
Triggered messages	APP_DATA_PACKET, MAC_SELF_EXIT_EXPECTING_RX
Direct State transitions	MAC_STATE_EXPECTING_RX (se a frame recebida for beacon e se for a primeira vez que se recebe)
Indirect State transitions	MAC_STATE_DEFAULT (recebe a *frame* de dados reais depois de alguns beacons prévios, pára-se de esperar
Handling Description	Quando o módulo de Rádio recebe com sucesso uma *frame* a partir do módulo do Canal Sem Fios encaminha-a para o módulo Mac, descascando-a e enviando uma mensagem do tipo MAC_FRAME. O módulo Mac vai processar esta mensagem a não ser que esteja no estado MAC_STATE_TX (a seguir a mensagem recebida será descartada)

	Processo: Se MAC_FRAME é *beacon*, então o Mac muda de estado para MAC_STATE_EXPECTING_RX e agenda uma mensagem que o vai forçar a voltar atrás ao estado por defeito após decorrido o período "sleepInterval". Isto foi feito a fim de evitar a espera indefinida no MAC_STATE_EXPECTING_RX para a *frame* de dados a ser recebida. Por outro lado, se a mensagem não é *beacon* de seguida, sem atrasos, Mac agenda uma auto-mensagem do tipo MAC_SELF_EXIT_EXPECTING_RX para que entre o estado MAC_STATE_DEFAULT. Finalmente desmembra e entrega a mensagem/*frame* de dados reais ao módulo Aplicação.

Tabela B.9 – MAC_SELF_EXIT_EXPECTING_RX

MAC_SELF_EXIT_EXPECTING_RX [13]	
Senders	Módulo Mac
Triggered messages	MAC_2_RADIO_ENTER_LISTEN, MAC_SELF_INITIATE_TX
Direct State transitions	Módulo Mac inicia com MAC_STATE_DEFAULT
Indirect State transitions	-
Handling Description	Se o estado Mac é MAC_STATE_EXPECTING_RX: - Muda o seu estado para MAC_STATE_DEFAULT, - Define o modo do módulo de Rádio para "listen" Qualquer pacote que o módulo de aplicação envie para o Mac enquanto o último espera por dados da rede, será empurrado para o *buffer*

	Mac agenda (sem atraso) uma auto-mensagem do tipo MAC_SELF_INITIATE_TX com vista a transmitir o pacote

Tabela B.10 – MAC_SELF_PERFORM_CARRIER_SENSE

MAC_SELF_PERFORM_CARRIER_SENSE* [13]	
Senders	Módulo Mac
Triggered messages	MAC_2_RADIO_SENSE_CARRIER, MAC_SELF_EXIT_CARRIER_SENSE, MAC_SELF_CHECK_TX_BUFFER, MAC_2_RADIO_ENTER_LISTEN, MAC_SELF_PERFORM_CARRIER_SENSE
Direct State transitions	MAC_STATE_CARRIER_SENSING (se a indicação do *carrier sense* do módulo de Rádio é valida)
Indirect State transitions	MAC_STATE_TX
Handling Description	O processo de mensagem com o tipo MAC_SELF_PERFORM_CARRIER_SENSE é um bocado mais complicado. Primeiro que tudo, Mac vai processar esta mensagem somente se o estado actual for MAC_STATE_DEFAULT. O Mac não deverá executar o *carrier sense* enquanto este estiver á espera de um pacote de dados, ou durante uma transmissão ou mesmo quando já esta a decorrer um *carrier sense.* Todavia, é improvável (se não mesmo impossível) que o controlo do programa chegue a tal ponto Tal como acontece para a maioria dos circuitos integrados da Rádio utilizados em nós sensores a indicação de carrier sense nem sempre é valida. Quando o circuito integrado é ligado ou quando entra no modo de escuta (*listen mode*) depois do modo de repouso ou

	depois TX ou RX, precisa de receber uma quantidade finita de símbolos antes da indicação do CS se tornar válida. Se a indicação do *carrier sense* for válida o módulo Mac envia para o módulo de Rádio uma mensagem do tipo MAC_2_RADIO_SENSE_CARRIER, dando desta forma ao módulo de Rádio uma instrução para começar a executar o *Carrier Sense*. Ao mesmo tempo, Mac agenda uma auto-mensagem do tipo MAC_SELF_EXIT_CARRIER_SENSE com a função de limitar o tempo, e que ao expirar significa que a transportadora está livre. O estado muda para MAC_STATE_CARRIER_SENSING. Se a transportadora não for válida, Mac tomará as medidas apropriadas baseando-se nas razões de não validade. Se o motivo for RADIO_IN_TX_MODE então tirará vantagens disso e o Mac automaticamente verifica o seu *buffer* e encaminhará a *frame* de dados para o módulo de Rádio. Se o motivo for RADIO_SLEEPING então o módulo Mac acorda o rádio (enviando para o Módulo de Rádio uma mensagem do tipo MAC_2_RADIO_ENTER_LISTEN) e agenda uma nova MAC_SELF_PERFORM_CARRIER_SENS e uma auto-mensagem após o período de tempo "radioDelayForValidCS" Se o motivo é o RADIO_NON_READY, o módulo Mac simplesmente irá esperar pelo período de tempo "radioDelayForValidCS" e agendará um novo MAC_SELF_PERFORM_CARRIER_SENS

Tabela B.11 – MAC_SELF_EXIT_CARRIER_SENSE

MAC_SELF_EXIT_CARRIER_SENSE* [13]	
Senders	Módulo Mac
Triggered messages	MAC_SELF_CHECK_TX_BUFFER
Direct State transitions	MAC_STATE_TX
Indirect State transitions	-
Handling Description	Quando o tempo para ser efectuado o *carrier sensing* expira (o expirar do tempo significa que o Mac entretanto não recebeu nenhuma mensagem RADIO_2_MAC_SENSED_CARRIER) a portadora é assumida como estando livre. O estado do módulo Mac muda para MAC_STATE_TX e ao mesmo tempo é agendada uma nova auto-mensagem do tipo MAC_SELF_CHECK_TX_BUFFER com vista a iniciar uma nova transmissão

Tabela B.12 – RADIO_2_MAC_SENSED_CARRIER

RADIO_2_MAC_SENSED_CARRIER* [13]	
Senders	Módulo Rádio
Triggered messages	MAC_SELF_SET_RADIO_SLEEP
Direct State transitions	MAC_STATE_DEFAULT
Indirect State transitions	-
Handling Description	Este tipo de mensagem é enviado pelo módulo de Rádio quando a portadora é verificada. O meio não está livre para transmissão e por isso o módulo Mac retorna para MAC_DEFAULT_STATE e agenda uma auto-mensagem do tipo MAC_SELF_SET_RADIO_SLEEP (se o *duty cycle* está definido na lista de parâmetros do omnetpp.ini) Finalmente, se uma auto-mensagem do tipo MAC_SELF_EXIT_CARRIER_SENSE está já agendada, será cancelada e apagada. Não vale a pena deixar esgotar-se o tempo para

	indicação da transportadora livre ocorrer pois já sabemos, que o meio está ocupado.

Tabela B.13 – RADIO_2_MAC_FULL_BUFFER

RADIO_2_MAC_FULL_BUFFER [13]	
Senders	Módulo de Rádio
Triggered messages	-
Direct State transitions	-
Indirect State transitions	-
Handling Description	Este tipo de mensagem indica que o *buffer* do módulo de Rádio esta cheio.

Tabela B.14 – RESOURCE_MGR_OUT_OF_ENERGY

RESOURCE_MGR_OUT_OF_ENERGY [13]	
Senders	Módulo de Gestão de Recursos
Triggered messages	-
Direct State transitions	-
Indirect State transitions	-
Handling Description	Se o módulo Mac recebe uma mensagem com este tipo de mensagem, estará a indicar que o nó está sem energia sendo por isso que, todos os módulos do nó devem ser desligados, incluindo também o módulo Mac.

C. Anexo –Tabela de Consumo de Energia

A Tabela C.1 apresenta os valores para o consumo de energia para protocolos de comunicação referidos na Secção 5.3.

Tabela C.1 – Consumo de energia para vários protocolos utilizando a aplicação valueReportng do Castalia

Nó ID	Mac Bypass	MAC Just Carrier Sense	MAC Tunable	S-MAC	T-MAC
0	40,7942	40,7942	11,7098	8,147160	5,443020
1	40,7942	40,7942	11,7099	8,146710	5,426240
2	40,7942	40,7942	11,7100	8,145500	5,429580
3	40,7942	40,7942	11,9995	8,302460	5,516480
4	40,7942	40,7942	11,7101	8,148070	5,427530
5	40,7942	40,7942	11,7103	8,144910	5,422630
6	40,7942	40,7942	11,7098	8,186210	5,473810
7	40,7942	40,7942	11,7096	8,143330	5,454640
8	40,7942	40,7942	11,7100	8,144320	5,428360
9	40,7942	40,7942	11,7099	8,147160	5,442110
10	40,7942	40,7942	11,7096	8,146050	5,429230
11	40,7942	40,7942	11,7101	8,145500	5,426850
12	40,7940	40,7940	12,2205	8,147870	5,499330
13	40,7942	40,7942	11,9311	8,148070	5,470850
14	40,7942	40,7942	11,7112	8,144540	5,444020
15	40,7942	40,7942	11,9343	8,143820	5,477570
16	40,7942	40,7942	11,7100	8,179390	5,476160
17	40,7942	40,7942	11,7100	8,143950	5,449780
18	40,7942	40,7942	11,7097	8,147030	5,424730
19	40,7942	40,7942	11,7101	8,145500	5,435020
20	40,7942	40,7942	11,7099	8,145070	5,430950
21	40,7940	40,7940	12,0655	8,283110	6,129560
22	40,7942	40,7942	11,8313	8,148010	5,448440
23	40,7942	40,7942	11,7099	8,144910	5,442590
24	40,7942	40,7942	11,8752	8,146140	5,775330
25	40,7942	40,7942	11,7104	8,143410	5,420190
26	40,7942	40,7942	11,7097	8,144320	5,444690
27	40,7942	40,7942	11,7100	8,186720	5,469860
28	40,7942	40,7942	11,7097	8,146710	5,430770
29	40,7942	40,7942	11,7096	8,145500	5,434100
30	40,7940	40,7940	12,1895	8,122800	6,118400
31	40,7942	40,7942	11,7096	8,147430	5,435110
32	40,7942	40,7942	11,7098	8,144910	5,428040
33	40,7942	40,7942	11,7337	8,145830	5,746240
34	40,7942	40,7942	11,7098	8,143360	5,445550
35	40,7942	40,7942	11,7106	8,144320	5,437420

D. Anexo –Tabela de Consumo de Energia

As tabelas D.1 a D.3 apresentam os valores para o consumo de energia para protocolos de comunicação referidos na Secção 5.3.

Tabela D.1 – Consumo de energia de vários protocolos utilizando a aplicação mobility1 do Castalia

	MAC Bypass	MAC justCarrierSense	MAC Tunable	T-MAC	S-MAC
0	6,794000	6,794000	3,8587000	1,439460	1,406270
1	6,790280	6,790280	6,7485700	6,236290	1,400630
2	6,790280	6,790280	6,7495700	5,404130	1,404120

Tabela D.2 – Consumo de energia de vários protocolos utilizando a aplicação mobility2 do Castalia

	MAC Bypass	MAC justCarrierSense	MAC Tunable	T-MAC	S-MAC
0	1,3540	1,3540	1,3514	1,3540	1,3540
1	1,2947	1,2970	1,3391	1,2947	1,2970
2	1,2947	1,2947	1,3401	1,2947	1,2947

Tabela D.3 – Consumo de energia de vários protocolos utilizando a aplicação mobility3 do Castalia

	MAC Bypass	MAC justCarrierSense	MAC Tunable	T-MAC	S-MAC
0	1,354000	1,3540000	1,351410	1,0839400	0,3380680
1	1,294650	1,2961700	1,339060	1,2970200	0,3453210
2	1,294650	1,2952700	1,340060	1,2499000	0,3510920

E. Anexo – Tabela de Consumo de Energia

A tabela E.1 apresenta valores para o consumo de energia para protocolos de comunicação referidos na Secção 5.3.

Tabela E.1 – Consumo de energia de vários protocolos utilizando a aplicação SimpleAgregation do Castalia

	MAC Bypass	MAC justCarrier Sense	MAC Tunable	T-MAC	S-MAC
0	6,795000	6,79500	4,44372	1,15318	1,34220
1	6,796000	6,79600	4,44453	1,15254	1,34337
2	6,796000	6,79600	4,42063	1,02171	1,34344
3	6,795990	6,79599	4,44484	1,14978	1,34749
4	6,795320	6,79532	5,89394	1,14979	1,34243
5	6,796000	6,79600	4,42016	1,01989	1,34341
6	6,794320	6,79432	5,93419	1,15318	1,34220
7	6,795000	6,79500	4,44394	1,15591	1,34216
8	6,796000	6,79600	4,42039	1,01898	1,34344
9	6,796000	6,79600	4,44458	1,15254	1,34337
10	6,795000	6,79500	4,44384	1,15591	1,34216
11	6,796000	6,79600	4,42056	1,01989	1,34344
12	6,794320	6,79432	4,66176	1,15509	1,34137
13	6,795320	6,79532	4,66427	1,14723	1,34169
14	6,795000	6,79500	4,41946	1,02253	1,37557
15	6,794320	6,79432	4,64426	1,15335	1,34146

F. Anexo – Gráficos de Consumo de Energia

As figuras F.1 a F.6 apresentam gráficos 3D de consumo de energia para uma grelha de 36 nós do cenário conciderado para protocolo T-MAC para valores de contentionPeriod 1, 20, 30, 40, 50, e 100 ms.

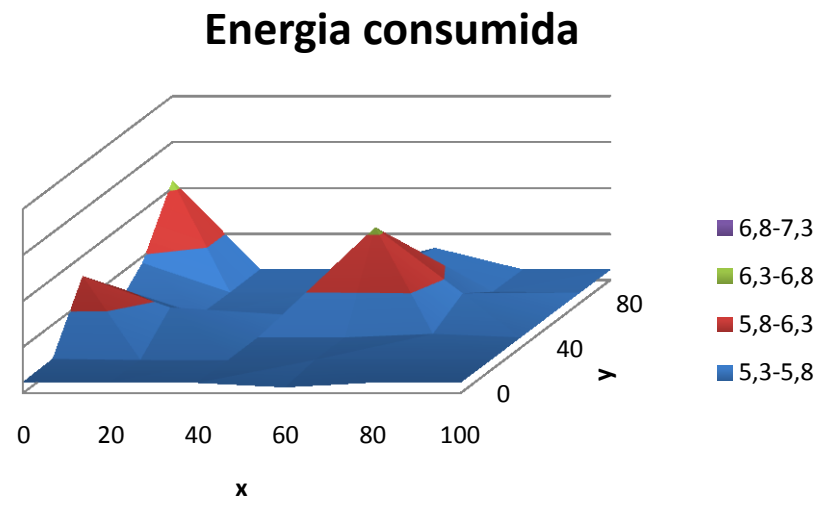


Figura F.1 – Consumo de energia em cada nó no protocolo T-MAC (*contentionPeriod* = 1)

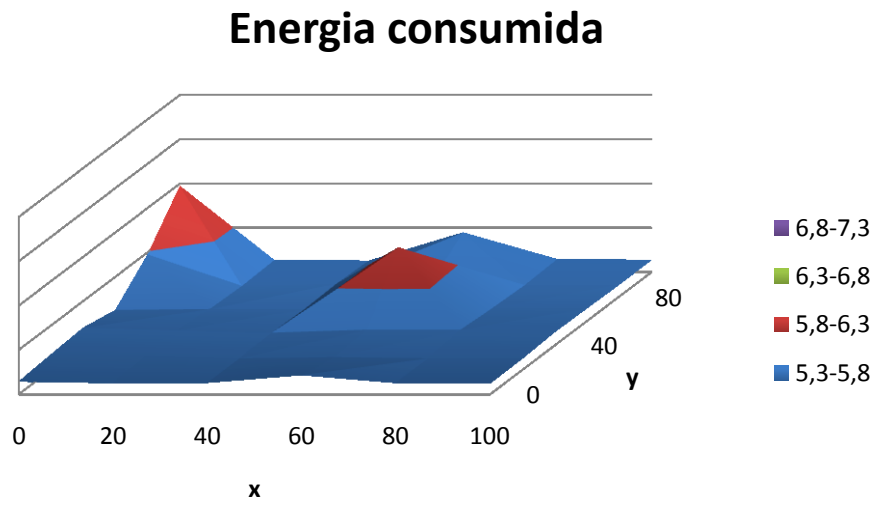


Figura F.2 – Consumo de energia em cada nó no protocolo T-MAC (*contentionPeriod* = 10)

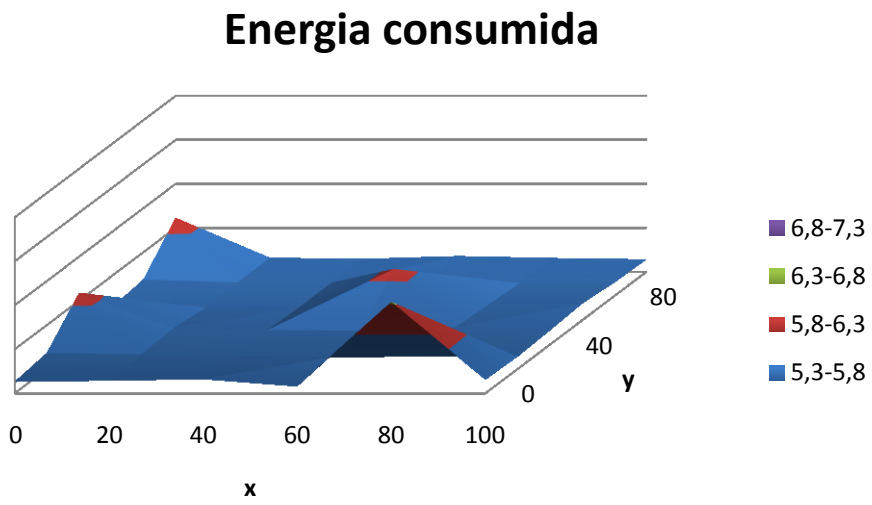


Figura F.3 – Consumo de energia em cada nó no protocolo T-MAC (*contentionPeriod* = 20)

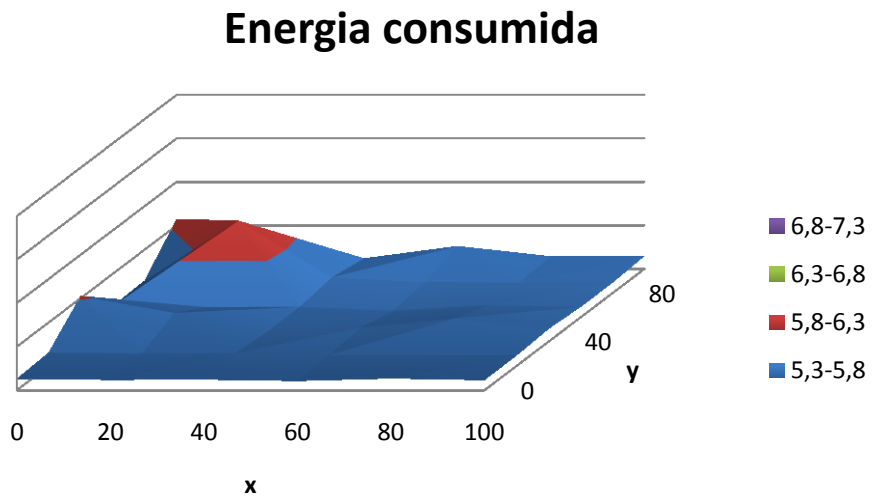


Figura F.4 – Consumo de energia em cada nó no protocolo T-MAC (*contentionPeriod* = 30)

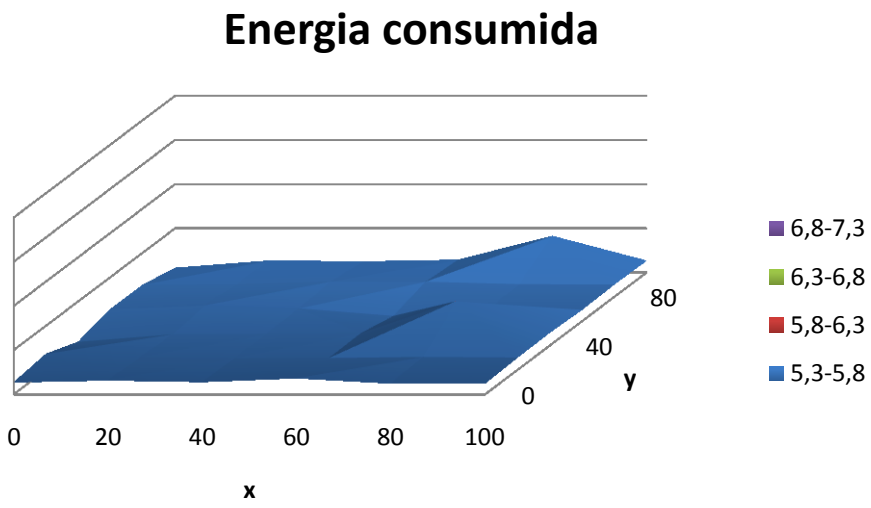


Figura F.5 – Consumo de energia em cada nó no protocolo T-MAC (*contentionPeriod* = 40)

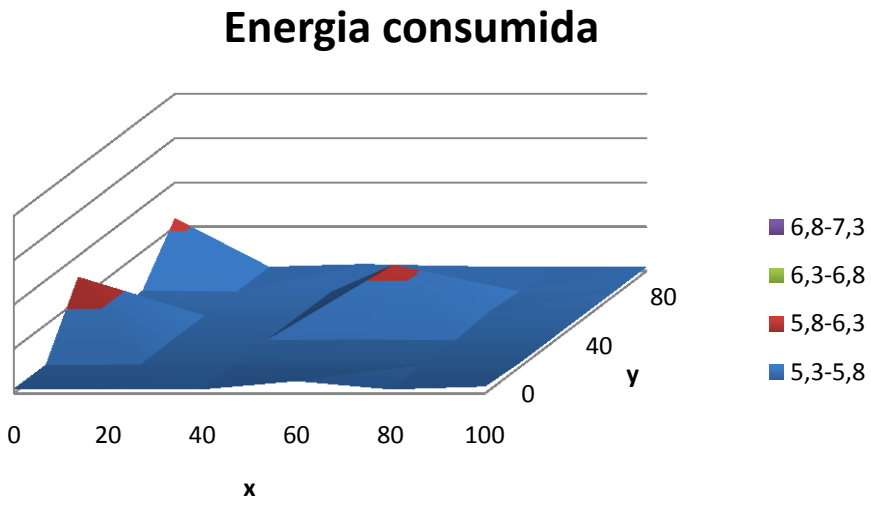


Figura F.6 – Consumo de energia em cada nó no protocolo T-MAC (*contentionPeriod* = 50)

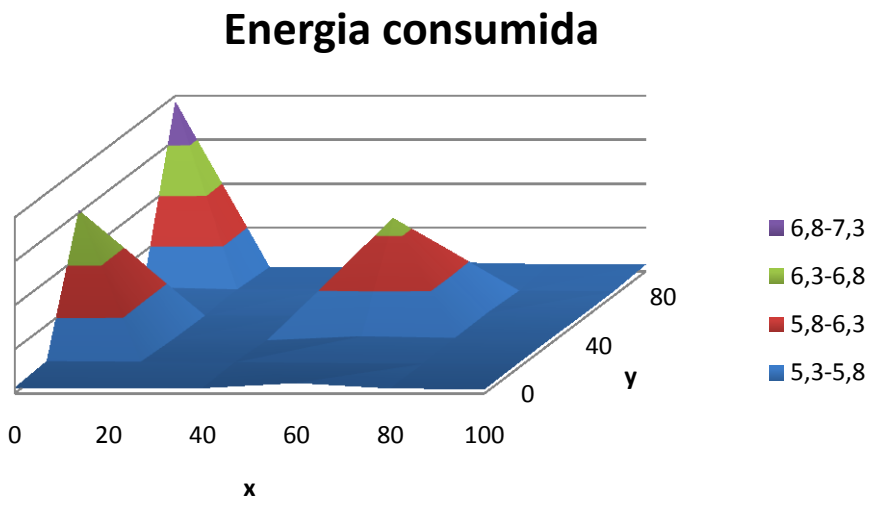


Figura F.7 – Consumo de energia em cada nó no protocolo T-MAC (*contentionPeriod* = 100)

G. Anexo – Tabela de Consumo de Energia

A Tabela G.1 apresenta os valores de energia consumida para o protocolo T-MAC dependendo dos valores do tempo de escuta (ListenTimeout).

Tabela G.1 – Consumo de energia de cada nó ao longo da variação do tempo de escuta

Nó ID	ListenTimeout (ms)			
	1	15	30	300
0	4,58305	5,443020	6,3611100	23,11940
1	4,58275	5,426240	6,3189000	23,44270
2	4,58164	5,429580	6,3340200	23,24270
3	4,51785	5,516480	6,3910700	28,05720
4	4,58887	5,427530	6,3274600	22,95180
5	4,58117	5,422630	6,3425000	23,11550
6	4,58827	5,473810	6,3754100	22,77740
7	4,58040	5,454640	6,3881700	23,26130
8	4,58083	5,428360	6,3364500	22,80660
9	4,58281	5,442110	6,3611100	22,88300
10	4,58703	5,429230	6,3400000	23,33370
11	4,58158	5,426850	6,3176900	22,98820
12	8,66814	5,499330	6,4072200	23,75680
13	4,58613	5,470850	6,3256500	23,26080
14	4,58061	5,444020	6,3603000	23,35210
15	4,58148	5,477570	6,3541300	23,50430
16	4,58429	5,476160	6,3902300	23,93670
17	4,57986	5,449780	6,3597200	23,26120
18	4,58442	5,424730	6,3264500	23,35180
19	4,58170	5,435020	6,3430900	23,31560
20	4,58102	5,430950	6,3336000	23,26100
21	6,47974	6,129560	7,0295500	23,55600
22	4,58579	5,448440	6,3456400	23,57000
23	4,58062	5,442590	6,3606800	23,13380
24	4,58465	5,775330	6,7016100	23,19270
25	4,57951	5,420190	6,3373200	23,02420
26	4,58016	5,444690	6,3600800	22,87940
27	4,58871	5,469860	6,3716000	22,68480
28	4,58318	5,430770	6,3352100	23,42460
29	4,58134	5,434100	6,3449300	23,29740
30	6,06512	6,118400	7,0037600	23,42890
31	4,58454	5,435110	6,3468400	23,22460
32	4,58062	5,428040	6,3388600	22,87930
33	4,58448	5,746240	6,7052400	23,35650
34	4,57909	5,445550	6,3754500	22,98840
35	4,58016	5,437420	6,3419000	23,02490

H. Anexo –Tabela de Consumo de Energia

A Tabela H.1 apresenta os valores de energia consumida para o protocolo T-MAC quando o período de contenção varia (1, 10, 20, 30, 40, 50, 100 ms). Estes são os valores que originam as figuras 5.10 a 5.18 da Secção 5.3

Tabela H.1 – Consumo de energia de cada nó ao longo da variação do período de contenção

Nó ID	*contentionPeriod*						
	1	**10**	**20**	**30**	**40**	**50**	**100**
0	5,41868	5,44302	5,43092	5,42947	5,43485	5,34850	5,36030
1	5,41844	5,42624	5,44517	5,42122	5,45748	5,35190	5,35874
2	5,41328	5,42958	5,46079	5,42460	5,43888	5,34668	5,36090
3	5,36526	5,51648	5,37890	5,40817	5,48175	5,43989	5,41953
4	5,41736	5,42753	6,34001	5,43715	5,42231	5,34575	5,35697
5	5,41410	5,42263	5,45476	5,41796	5,42324	5,37921	5,35039
6	5,42632	5,47381	5,48213	5,45179	5,48134	5,34522	5,38841
7	5,41348	5,45464	5,46413	5,44101	5,44022	5,34607	5,38314
8	5,41291	5,42836	5,44217	5,44629	5,42935	5,34588	5,37574
9	5,41777	5,44211	5,43520	5,42725	5,43485	5,34778	5,36621
10	5,41753	5,42923	5,44245	5,42591	5,44658	5,34735	5,35783
11	5,41328	5,42685	5,44084	5,42062	5,44026	5,35148	5,35636
12	6,07642	5,49933	5,88683	5,82315	5,34254	6,05835	6,82347
13	5,73994	5,47085	5,48692	5,63071	5,42685	5,35121	5,35984
14	5,41398	5,44402	5,46566	5,41963	5,43320	5,34854	5,34970
15	5,41456	5,47757	5,45874	5,46491	5,44339	5,35577	5,39532
16	5,45235	5,47616	5,49797	5,56882	5,80061	5,34607	5,37678
17	5,41372	5,44978	5,46181	5,44721	5,44570	5,34971	5,35788
18	5,41862	5,42473	5,44247	5,42907	5,43750	5,34850	5,36601
19	5,41844	5,43502	5,44335	5,41851	5,47969	5,35099	5,35454
20	5,41416	5,43095	5,45478	5,42113	5,44570	5,34697	5,35333
21	6,36189	6,12956	5,88492	5,37640	5,36024	5,92331	6,45960
22	5,64825	5,44844	5,47424	5,43989	5,42685	5,38389	5,38333
23	5,41410	5,44259	5,49060	5,42058	5,42966	5,34966	5,35221
24	5,73147	5,77533	5,48862	5,44270	5,43704	5,35612	5,40023
25	5,41257	5,42019	5,45413	6,08728	5,44203	5,34698	5,37577
26	5,41291	5,44469	5,45399	5,48150	5,47408	5,34539	5,38483
27	5,41777	5,46986	5,42978	5,42546	5,47122	5,34687	5,36166
28	5,41834	5,43077	5,44787	5,41867	5,44385	5,34581	5,36223
29	5,41410	5,43410	5,45351	5,42062	5,44026	5,34875	5,35275
30	6,37931	6,11840	5,91058	5,86960	5,35166	5,89354	7,22950
31	5,41736	5,43511	5,45751	5,41895	5,43408	5,34573	5,35243
32	5,41322	5,42804	5,44475	5,41796	5,42421	5,37715	5,35221
33	5,64696	5,74624	5,47684	5,55953	5,44339	5,35340	5,40205
34	5,41348	5,44555	5,45413	5,44101	5,71382	5,34431	5,38041
35	5,41291	5,43742	5,43947	5,44903	5,43389	5,34948	5,38119

Média	**5,517**	**5,501**	**5,516**	**5,485**	**5,453**	**5,405**	**5,492**
Desvio Padrão da Média	**0,1646**	**0,0990**	**0,1088**	**0,0907**	**0,0408**	**0,0943**	**0,2243**
Desvio Padrão	**0,2510**	**0,1712**	**0,1885**	**0,1473**	**0,0814**	**0,1716**	**0,4220**

I. Anexo –.Castalia-Primary-Output.txt

Apresenta-se na Tabela I.1 o ficheiro gerado ao compilar a aplicação valueReporting onde se pode ver todos os ficheiros e valores de variáveis utilizados nesta simulação.

Tabela I.1 – Ficheiro Castalia-Primary-Output.txt da aplicação valueReporting

```
OMNeT++/OMNEST Discrete Event Simulation  (C) 1992-2005 Andras Varga
Release: 3.3p1, edition: Academic Public License.
See the license for distribution terms and warranty disclaimer
Setting up Cmdenv...

Processing listfile: /home/luis/omnetpp33/Castalia-
2.1b/Simulations/valueReporting/../../nedfiles.lst
Loading NED file:
./src/Physical_Process/CustomizablePhysicalProcess/CustomizablePhysicalProcess.ned
Loading NED file:
./src/Physical_Process/CarsPhysicalProcess/CarsPhysicalProcess.ned
Loading NED file: ./src/Physical_Process/PhysicalProcessGenericModule.ned
Loading NED file: ./src/SensorNetwork.ned
Loading NED file: ./src/Wireless_Channel/WirelessChannel.ned
Loading NED file: ./src/Node/Mobility_Module/VirtualMobilityModule.ned
Loading NED file:
./src/Node/Mobility_Module/LineMobilityModule/LineMobilityModule.ned
Loading NED file:
./src/Node/Mobility_Module/NoMobilityModule/NoMobilityModule.ned
Loading NED file: ./src/Node/Resource_Manager/ResourceGenericManager.ned
Loading NED file: ./src/Node/Communication/CommunicationModule.ned
Loading NED file: ./src/Node/Communication/Radio/RadioModule.ned
Loading NED file: ./src/Node/Communication/MAC/MacGenericModule.ned
Loading NED file:
./src/Node/Communication/MAC/TunableMac/TunableMacModule.ned
Loading NED file: ./src/Node/Communication/MAC/TMac/TMacModule.ned
Loading NED file:
./src/Node/Communication/MAC/BypassMac/BypassMacModule.ned
Loading NED file:
./src/Node/Communication/Network/simpleTreeRouting/simpleTreeRoutingModule.ne
d
Loading NED file:
./src/Node/Communication/Network/multipathRingsRouting/multipathRingsRoutingMo
dule.ned
Loading NED file: ./src/Node/Communication/Network/NetworkGenericModule.ned
Loading NED file:
./src/Node/Communication/Network/BypassRouting/BypassRoutingModule.ned
Loading NED file: ./src/Node/Sensor_Device_Manager/SensorDevMgrModule.ned
Loading NED file:
./src/Node/Application/throughputTest/throughputTest_ApplicationModule.ned
Loading NED file:
```

```
./src/Node/Application/BridgeTest/BridgeTest_ApplicationModule.ned
Loading NED file:
./src/Node/Application/valueReporting/valueReporting_ApplicationModule.ned
Loading NED file: ./src/Node/Application/ApplicationGenericModule.ned
Loading NED file:
./src/Node/Application/templateApplication/template_ApplicationModule.ned
Loading NED file:
./src/Node/Application/valuePropagation/valuePropagation_ApplicationModule.ned
Loading NED file:
./src/Node/Application/connectivityMap/connectivityMap_ApplicationModule.ned
Loading NED file:
./src/Node/Application/simpleAggregation/simpleAggregation_ApplicationModule.ned
Loading NED file: ./src/Node/Node.ned

Preparing for Run #1...
Setting up network `SN'...
Initializing...

Running simulation...
** Event #0   T=1.28000000e-04 (128us)   Elapsed: 0.000s (0m 00s)   ev/sec=0
** Event #100000   T=     101 ( 1m 41s)   Elapsed: 0.311s (0m 00s)   ev/sec=321543
** Event #200000   T=206.90638 ( 3m 26s)   Elapsed: 0.748s (0m 00s)   ev/sec=228833
** Event #300000   T=317.34765 ( 5m 17s)   Elapsed: 1.134s (0m 01s)   ev/sec=259740
** Event #400000   T=421.08042 ( 7m  1s)   Elapsed: 1.451s (0m 01s)   ev/sec=316456
** Event #500000   T=525.92085 ( 8m 45s)   Elapsed: 1.777s (0m 01s)   ev/sec=307692

<!> Simulation time limit reached -- simulation stopped.


Calling finish() at end of Run #1...
Node [0] spent energy: 5.41868
Node [1] spent energy: 5.41844
Node [2] spent energy: 5.41328
Node [3] spent energy: 5.36526
Node [4] spent energy: 5.41736
Node [5] spent energy: 5.4141
Node [6] spent energy: 5.42632
Node [7] spent energy: 5.41348
Node [8] spent energy: 5.41291
Node [9] spent energy: 5.41777
Node [10] spent energy: 5.41753
Node [11] spent energy: 5.41328
Node [12] spent energy: 6.07642
Node [13] spent energy: 5.73994
Node [14] spent energy: 5.41398
Node [15] spent energy: 5.41456
Node [16] spent energy: 5.45235
Node [17] spent energy: 5.41372
Node [18] spent energy: 5.41862
Node [19] spent energy: 5.41844
```

```
Node [20] spent energy: 5.41416
Node [21] spent energy: 6.36189
Node [22] spent energy: 5.64825
Node [23] spent energy: 5.4141
Node [24] spent energy: 5.73147
Node [25] spent energy: 5.41257
Node [26] spent energy: 5.41291
Node [27] spent energy: 5.41777
Node [28] spent energy: 5.41834
Node [29] spent energy: 5.4141
Node [30] spent energy: 6.37931
Node [31] spent energy: 5.41736
Node [32] spent energy: 5.41322
Node [33] spent energy: 5.64696
Node [34] spent energy: 5.41348
Node [35] spent energy: 5.41291

End run of OMNeT++


============================================================


Produced with the following version of Castalia:
Castalia 2.1b
This version released on 07-07-2009
[http://castalia.npc.nicta.com.au]

============================================================



************************************************************
Simulation's parameter settings were
[General]
preload-ned-files = *.ned @../../nedfiles.lst
network = SN                                    # this line is for Cmdenv, Tkenv will still
let you choose from a dialog
debug-on-errors = true
num-rngs = 11# 11 random number streams (or generators as OMNeT calls them)
seed-0-mt = 10          # the seeds for each of the RNGs
seed-1-mt = 20
seed-2-mt = 30
seed-3-mt = 40
seed-4-mt = 50
seed-5-mt = 60
seed-6-mt = 70
seed-7-mt = 80
seed-8-mt = 90
seed-9-mt = 100
seed-10-mt = 110
SN.wirelessChannel.rng-0 = 1                    # used to produce the random shadowing
```

```
effects
SN.wirelessChannel.rng-1 = 2                # used to decide if a receiver, with X
probability
SN.wirelessChannel.rng-2 = 9                        # used in temporal model
SN.node[*].nodeApplication.rng-0    = 3     # Randomizes the start time of the
application
SN.node[*].networkInterface.MAC.rng-0     = 4     # Produces values compared against
txProb
SN.node[*].networkInterface.MAC.rng-1     = 5     # Produces values between [0 ....
randomTxOffset]
SN.node[*].nodeResourceMgr.rng-0 = 6      # Produces values of the clock drift of the
CPU of each node
SN.node[*].nodeSensorDevMgr.rng-0         = 7     # Produces values of the sensor
devices' bias
SN.node[*].nodeSensorDevMgr.rng-1         = 8     # Produces values of the sensor
devices' noise
SN.physicalProcess[*].rng-0           = 10  # currently used only in
CarsPhysicalProcess
sim-time-limit = 600s
output-vector-file = Castalia-statistics.vec
output-scalar-file = Castalia-statistics.sca
[Cmdenv]
express-mode = yes
module-messages = yes
event-banners = no                          # "yes" outputs all events in the screen
performance-display = no
[Parameters]
SN.debugInfoFilename = "Castalia-Debug.txt"
SN.field_z = 0
SN.xCellSize = 5
SN.yCellSize = 5
SN.zCellSize = 5
SN.xGridSize = 3
SN.yGridSize = 3
SN.zGridSize = 3
SN.wirelessChannel.onlyStaticNodes = true
SN.wirelessChannel.bidirectionalSigma = 1
SN.wirelessChannel.temporalModelParametersFile = ""
SN.wirelessChannel.pathLossMapFile = ""
SN.wirelessChannel.PRRMapFile = ""
SN.node[*].zCoor = 0
SN.node[*].phi = 0
SN.node[*].theta = 0
SN.node[*].mobilityModuleName = "NoMobilityModule"
SN.node[*].nodeMobilityModule.printDebugInfo = false
SN.node[*].nodeSensorDevMgr.devicesSensitivity = "0"
SN.node[*].nodeSensorDevMgr.devicesResolution = "0.001"
SN.node[*].nodeSensorDevMgr.devicesSaturation = "1000"
SN.field_x = 100                                    # meters
SN.field_y = 100                                    # meters
```

```
SN.numNodes = 36
SN.deploymentType = 1                                   # 0 means a uniform random
deployment inside the network's X,Y boundaries
SN.numPhysicalProcesses = 1
SN.physicalProcessModuleName = "CustomizablePhysicalProcess"
SN.physicalProcess[0].printDebugInfo = false
SN.physicalProcess[0].description = "Fire"
SN.physicalProcess[0].inputType = 1                     #  0 --> values are dictated by
"directNodeValueAssignment" parameter
SN.physicalProcess[0].directNodeValueAssignment = "(0) 6:40"  #Statically assign
values to specific nodes.
SN.physicalProcess[0].multiplicative_k = 0.25       # multiplicative parameter (k)
SN.physicalProcess[0].attenuation_exp_a = 0.1       # attenuation exponent (a)
SN.physicalProcess[0].sigma = 0.2               # standart deviation to the zero-mean
additive gaussian noise (sigma)
SN.physicalProcess[0].max_num_snapshots = 11    # the maximum number of descrete
states/snapshots that can be specified in source_X params below
SN.physicalProcess[0].numSources = 1                    # number of differemt temperature
sources, currently only up to 5 sources are supported
SN.physicalProcess[0].source_0 = "0 2 38 10; 10 2 38 60; 20 2 38 60; 30 2 38 10; 40 2
38 10; 50 2 38 60; 60 2 38 60; 70 2 38 10; 80 2 38 10; 90 2 38 60; 100 2 38 60"   #
string showing how sources evolve over time (change in location and value)
SN.physicalProcess[0].source_1 = ""
SN.physicalProcess[0].source_2 = ""
SN.physicalProcess[0].source_3 = ""
SN.physicalProcess[0].source_4 = ""
SN.physicalProcess[0].tracefileName = ""
SN.node[*].nodeResourceMgr.printDebugInfo = false
SN.node[*].nodeResourceMgr.initialEnergy = 18720                    # joules
SN.node[*].nodeResourceMgr.sigmaCPUClockDrift = 0.00003
SN.node[*].nodeResourceMgr.ramSize = -1
SN.node[*].nodeResourceMgr.flashSize = -1
SN.node[*].nodeResourceMgr.flashWriteCost = -1
SN.node[*].nodeResourceMgr.flashReadCost = -1
SN.node[*].nodeResourceMgr.imageSize = -1
SN.node[*].nodeResourceMgr.cpuPowerSpeedLevelNames = ""
SN.node[*].nodeResourceMgr.cpuPowerPerLevel = ""
SN.node[*].nodeResourceMgr.cpuSpeedPerLevel = ""
SN.node[*].nodeResourceMgr.cpuInitialPowerLevel = -1
SN.node[*].nodeResourceMgr.baselineNodePower = 6                # in mW
SN.node[*].nodeResourceMgr.periodicEnergyCalculationInterval = 1000          # in ms
SN.node[*].nodeSensorDevMgr.printDebugInfo = false
SN.node[*].nodeSensorDevMgr.numSensingDevices = 1
SN.node[*].nodeSensorDevMgr.pwrConsumptionPerDevice = "0.02"      # in mW
SN.node[*].nodeSensorDevMgr.sensorTypes = "Temperature"
SN.node[*].nodeSensorDevMgr.corrPhyProcess = "0"                #string-array that
holds the index of the Physical process that each one of the sensors monitors
SN.node[*].nodeSensorDevMgr.maxSampleRates = "1"                # 1 sample per sec
SN.node[*].nodeSensorDevMgr.devicesBias = "0.1"                 #the sigmas (may be
more than one as the variable is an array) of the bias of each one of the sensors
```

```
SN.node[*].nodeSensorDevMgr.devicesDrift = ""
SN.node[*].nodeSensorDevMgr.devicesNoise = "0.1"                #the sigmas of the
noise of each one of the sensors
SN.node[*].nodeSensorDevMgr.devicesHysterisis = ""
SN.node[*].nodeSensorDevMgr.devicesSensitivity = "0"
SN.node[*].nodeSensorDevMgr.devicesResolution = "0.001"
SN.node[*].nodeSensorDevMgr.devicesSaturation = "1000"
SN.wirelessChannelModuleName = "WChannel_Interference"
SN.wirelessChannel.printDebugInfo = true
SN.wirelessChannel.printStatistics = true     #if true it prints aggregate statistics about
Tx, Rx, collisions
SN.wirelessChannel.pathLossExponent = 2.4
SN.wirelessChannel.PLd0 = 55                          # dBm  the Path Loss at d0, should
be a positive value
SN.wirelessChannel.d0 = 1                     # meters
SN.wirelessChannel.collisionModel = 2       # 0 --> No Collisions (no Interference)
SN.wirelessChannel.sigma = 4
SN.wirelessChannel.bidirectionalSigma = 1
SN.wirelessChannel.temporalModelParametersFile = ""
SN.wirelessChannel.pathLossMapFile = ""
SN.wirelessChannel.PRRMapFile = ""
SN.node[*].networkInterface.Radio.printDebugInfo = false
SN.node[*].networkInterface.Radio.printDroppedPacketsStatistics = true #if true, it
prints an overall analysis of the dropped packets at radio upon Rx
SN.node[*].networkInterface.Radio.printStateTransitions = true              #if true, it
prints the transitions of the radio between it states (Tx, Listen(includes Rx), Sleep)
SN.node[*].networkInterface.Radio.dataRate = 250                            # kbps
250kbps for Telos motes   and 19.2kbps for MICA2 motes
SN.node[*].networkInterface.Radio.noiseBandwidth = 194                              # in
KHz (MICA2 motes = 30, Telos motes = 194)
SN.node[*].networkInterface.Radio.noiseFloor = -100               # dBm  (-105 for
mica2 motes and  -100 for telos motes)
SN.node[*].networkInterface.Radio.modulationType = "PSK"                # "IDEAL" --
> MODULATION_IDEAL (no transmission errors)
SN.node[*].networkInterface.Radio.encodingType = 0                          # 0 --> NRZ
(only NRZ is currently implemented)
SN.node[*].networkInterface.Radio.receiverSensitivity = -95                 # in dBm  a)-
98 for MICA2  b)-95 for TELOS
SN.node[*].networkInterface.Radio.rxPower = 62                              # mW,  62  for
CC2420 (Telos)
SN.node[*].networkInterface.Radio.listenPower = 62                          # mW,  62  for
CC2420 (Telos)
SN.node[*].networkInterface.Radio.sleepPower = 1.4                          # mW,  1.4 for
CC2420 (Telos)
SN.node[*].networkInterface.Radio.txPowerLevels = "0 -1 -3 -5 -7 -10 -15 -25"  # in
dBm, first element is the default level
SN.node[*].networkInterface.Radio.txPowerConsumptionPerLevel = "57.42 55.18
50.69 46.2 42.24 36.3 32.67 29.04"  # power consumption per TX level in mW
SN.node[*].networkInterface.Radio.txPowerLevelUsed = 0                              # 0 to
N-1: index referring to array txPowerLevels. If defined outside txPowerLevels limits
```

```
gets default value 0.
SN.node[*].networkInterface.Radio.txModeUsed = 0                    # a)0 for
CARRIER_SENSE_NONE , b) 1 for CARRIER_SENSE_ONCE_CHECK, c)2 for
CARRIER_SENSE_PERSISTENT
SN.node[*].networkInterface.Radio.bufferSize = 16                   # number of
MAC frames
SN.node[*].networkInterface.Radio.maxPhyFrameSize = 127                 # in
bytes - derived by the IEEE 802.15.4 standard specification
SN.node[*].networkInterface.Radio.phyFrameOverhead = 6                  # in
bytes - derived by the IEEE 802.15.4 standard specification
SN.node[*].networkInterface.Radio.delaySleep2Listen = 0.194         #in ms,  0.194
for CC2420 (Telos)
SN.node[*].networkInterface.Radio.delayListen2Sleep = 0.05          #in ms
SN.node[*].networkInterface.Radio.delayTx2Sleep = 0.05              #in ms
SN.node[*].networkInterface.Radio.delaySleep2Tx = 0.194                 #in ms,
0.194 for CC2420 (Telos)
SN.node[*].networkInterface.Radio.delayListen2Tx = 0.01                 #in ms
SN.node[*].networkInterface.Radio.delayTx2Listen = 0.01                 #in ms
SN.node[*].networkInterface.Radio.delayCSValid = 0.128              #in ms,  0.128
for CC2420 (Telos)
SN.node[*].networkInterface.macModuleName = "TMacModule"
SN.node[*].networkInterface.MAC.printDebugInfo = true
SN.node[*].networkInterface.MAC.printPotentiallyDroppedPacketsStatistics = true
        #if true, it prints an overall analysis of the dropped packets at MAC upon Rx
SN.node[*].networkInterface.MAC.printStateTransitions = false       #if true, it
prints the transitions of the MAC between it states (Default, Tx, Carrier Sensing,
Expecting Rx)
SN.node[*].networkInterface.MAC.maxMACFrameSize = 100    #100 bytes
SN.node[*].networkInterface.MAC.macFrameOverhead = 14    #14 bytes
SN.node[*].networkInterface.MAC.ackFrameSize = 14    #14 bytes
SN.node[*].networkInterface.MAC.syncFrameSize = 22   #22 bytes
SN.node[*].networkInterface.MAC.rtsFrameSize = 14    #14 bytes
SN.node[*].networkInterface.MAC.ctsFrameSize = 14    #14 bytes
SN.node[*].networkInterface.MAC.macBufferSize = 32   ##32 number of maximum
frames held from the upper layer
SN.node[*].networkInterface.MAC.maxTxRetries = 2     ##2 number of transmission
attempts per packet
SN.node[*].networkInterface.MAC.resyncTime = 40          ##40 seconds
SN.node[*].networkInterface.MAC.allowSinkSync = 1    ##1 bool
SN.node[*].networkInterface.MAC.contentionPeriod = 1 ##10 mseconds
SN.node[*].networkInterface.MAC.listenTimeout = 15   ##15 mseconds
SN.node[*].networkInterface.MAC.waitTimeout = 5          ##5 mseconds
SN.node[*].networkInterface.MAC.useFrts = 0              ##0 boolean, not
supported
SN.node[*].networkInterface.MAC.useRtsCts = 1        ##1 boolean
SN.node[*].networkInterface.MAC.disableTAextension = 0       ##0 boolean
SN.node[*].networkInterface.MAC.frameTime = 610          ##610 mseconds
SN.node[*].networkInterface.networkModuleName = "simpleTreeRoutingModule"
SN.node[*].networkInterface.Network.printDebugInfo = false
SN.node[*].networkInterface.Network.maxNetFrameSize = 86     # in bytes
```

```
SN.node[*].networkInterface.Network.netDataFrameOverhead = 16        # in bytes
SN.node[*].networkInterface.Network.netTreeSetupFrameOverhead = 16   # in bytes
SN.node[*].networkInterface.Network.netBufferSize = 32            # number of
maximum frames held from the upper layer
SN.node[*].networkInterface.Network.maxNeighborsTableSize =  30      # maximum
number of total records that can be stored
SN.node[*].networkInterface.Network.maxNumberOfParents = 1
SN.node[*].networkInterface.Network.netSetupTimeout = 50 # in msec
SN.node[*].networkInterface.Network.topoSetupUdateTimeout = 10000 # in msec
SN.node[*].networkInterface.Network.rssiBased_NeighborQuality = true
SN.node[*].networkInterface.Network.neighbor_RSSIThreshold = -90 # in dBm  (set it
to 0 in order to disable the
SN.node[*].appModuleName = "valueReporting_ApplicationModule"
SN.node[*].nodeApplication.applicationID = "valueReporting"
SN.node[*].nodeApplication.printDebugInfo = true
SN.node[*].nodeApplication.priority = 1
SN.node[*].nodeApplication.maxSampleInterval = 60000 # in msec
SN.node[*].nodeApplication.minSampleInterval = 1000  # in msec
SN.node[*].nodeApplication.isSink = false
SN.node[*].nodeApplication.maxAppPacketSize = 30    #in bytes
SN.node[*].nodeApplication.packetHeaderOverhead = 8 #in bytes
SN.node[*].nodeApplication.constantDataPayload = 12  # in bytes
[Run 1]
description = "Run 1"
seed-0-mt = 10
seed-1-mt = 11100000000000
seed-2-mt = 211
seed-3-mt = 40
seed-4-mt = 50
seed-5-mt = 60
seed-6-mt = 70
seed-7-mt = 80
seed-8-mt = 90
SN.wirelessChannel.printDebugInfo = true
SN.node[*].networkInterface.Radio.txPowerLevelUsed = 2
SN.node[*].networkInterface.Radio.printDebugInfo = true
SN.node[*].networkInterface.Network.maxNumberOfParents = 1
SN.node[*].networkInterface.Network.rssiBased_NeighborQuality = true
SN.node[*].networkInterface.Network.neighbor_RSSIThreshold = -89.3 # in dBm
SN.node[3].nodeApplication.isSink = true

---------------------------
End of simulation
---------------------------
```

J. Anexo – Omnetpp.ini

Na Tabela J.1 apresenta-se os parâmetros definidos pelo utilizador para a aplicação valueReporting no Castalia.

Tabela J.1 – Ficheiro omnetpp.ini da aplicação valueReporting

```
#
**********************************************************************
****************
# *  Copyright: Athens Information Technology (AIT),  2007, 2008
        *
# *              http://www.ait.gr                                    *
# *          Developed at the Broadband Wireless and Sensor Networks group (B-WiSe)
        *
# *              http://www.ait.edu.gr/research/Wireless_and_Sensors/overview.asp
        *
# *                                                                   *
# *  Author(s): Dimosthenis Pediaditakis                              *
# *                                                                   *
# *  This file is distributed under the terms in the attached LICENSE file.   *
# *  If you do not find this file, copies can be found by writing to:         *
# *                                                                   *
# *     NICTA, Locked Bag 9013, Alexandria, NSW 1435, Australia
         *
# *     Attention:  License Inquiry.                                 *
#
**********************************************************************
***************/


[General]
# The list of ned files to be dynamically loaded,
# make sure it points to $(CASTALIA_HOME)/nedfiles.lst
preload-ned-files = *.ned @../../nedfiles.lst

#always include the general and RNGs parameter file
include ../Parameter_Include_Files/general_and_RNGs.ini

# how long do you want your simulation to run?
sim-time-limit = 600s

output-vector-file = Castalia-statistics.vec
output-scalar-file = Castalia-statistics.sca


[Cmdenv]
# always include the omnet_cmdenv_reporting parameter file
```

```
include ../Parameter_Include_Files/omnet_cmdenv_reporting.ini


[Parameters]
# The filename that (potential) debug output will be written to
SN.debugInfoFilename = "Castalia-Debug.txt"

# COMPOUND MODULE: SN (the network)

include ../Parameter_Include_Files/BackwardCompatibility.ini

# define deployment details
SN.field_x = 100                                    # meters
SN.field_y = 100                                    # meters
SN.numNodes = 36
SN.deploymentType = 1                               # 0 means a uniform random
deployment inside the network's X,Y boundaries
                                                    # 1 means a grid deployment
                                                    # 2 means randomized grid deployment (i.e.
grid+ noise)
                                                    # any other number means that the user
must specify manually the locations of each one of the nodes

# how many physical processes do you have?
SN.numPhysicalProcesses = 1


# Choose a parameters file for module SN.physicalProcess[0] or define your own
include ../Parameter_Include_Files/physicalProcess_0_one_source_at_2_38.ini
#include ../Parameter_Include_Files/physicalProcess_0_node6_asssignedValue40.ini

# Choose a parameters file for module SN.node[*].resourceMgr or define your own
include ../Parameter_Include_Files/resourceMgr_2AAbatteries.ini

# Choose a parameters file for module SN.node[*].nodeSensorDevMgr or define your
own
include ../Parameter_Include_Files/nodeSensorDevMgr_Temperature.ini



# Choose a parameters file for module SN.wirelessChannel or define your own
#include ../Parameter_Include_Files/WChannel/No_Interference/WChannel_Ideal.ini
#include
../Parameter_Include_Files/WChannel/No_Interference/WChannel_noInterference_yesS
igma_yesAllBidirectional.ini
#include
../Parameter_Include_Files/WChannel/No_Interference/WChannel_noInterference_yesS
igma_noAllBidirectional.ini
#include
../Parameter_Include_Files/WChannel/Additive_Interference_Model/WChannel_yesInt
```

```
erference_noSigma_yesAllBidirectional.ini
#include
../Parameter_Include_Files/WChannel/Additive_Interference_Model/WChannel_yesInt
erference_yesSigma_yesAllBidirectional.ini
include
../Parameter_Include_Files/WChannel/Additive_Interference_Model/WChannel_Realist
ic.ini

# Choose a parameters file for module SN.node[*].networkInterface.Radio or define
your own
#include
../Parameter_Include_Files/Radio/TelosB_CC2420/radio_CC2420_IDEALmodulation.i
ni
include ../Parameter_Include_Files/Radio/TelosB_CC2420/radio_CC2420.ini

# Choose a parameters file for module SN.node[*].networkInterface.MAC or define
your own
#include ../Parameter_Include_Files/MAC_Bypass.ini
#include ../Parameter_Include_Files/MAC_Tunable.ini
#include ../Parameter_Include_Files/MAC_just_carrierSense.ini
include ../Parameter_Include_Files/T_MAC.ini

# Choose a parameters file for module SN.node[*].networkInterface.MAC or define
your own
#include ../Parameter_Include_Files/Routing_bypass.ini
#include ../Parameter_Include_Files/Routing_multipathRings.ini
include ../Parameter_Include_Files/Routing_simpleTree.ini

# ----------------------------------------------------------------
# Define the application module you want to use and its parameters
# ----------------------------------------------------------------
SN.node[*].appModuleName = "valueReporting_ApplicationModule"
SN.node[*].nodeApplication.applicationID = "valueReporting"
SN.node[*].nodeApplication.printDebugInfo = true
SN.node[*].nodeApplication.priority = 1
SN.node[*].nodeApplication.maxSampleInterval = 60000 # in msec
SN.node[*].nodeApplication.minSampleInterval = 1000  # in msec
SN.node[*].nodeApplication.isSink = false

# App_GenericDataPacket has the following real-world
# (non-simulation-specific) fields:
#         string source; ----------> 2bytes
#         string destination; -----> 2bytes
#         int seqNumber; ----------> 2bytes
#         string applicationID; ---> 2bytes
# Application header overhead: Total bytes = 4*2 = 8 (|*|)

# Additionally, the valueReporting_DataPacket (sub-class
# of App_GenericDataPacket) has some extra fields:
#    struct valueReportData  data; -----------> 12bytes
```

```
#       struct fireMessageData consists of:
#               unsigned short nodeID; --------> 2bytes
#               double locX; -------> 2bytes // we assume that 1 byte for the integral part
and 1byte for the fractional part are enough
#               double locY; -------> 2bytes
#               int sampleSN; ------> 4bytes
#               double value; ------> 2bytes
# Data Payload = 12bytes

# Total bytes (|*|) 8 + 12 = 20 (constant application packet size)

SN.node[*].nodeApplication.maxAppPacketSize = 30    #in bytes
SN.node[*].nodeApplication.packetHeaderOverhead = 8 #in bytes
SN.node[*].nodeApplication.constantDataPayload = 12  # in bytes



# Define your Runs. You can have none (in that case Castalia will run once with
# the parameter values defined above) or you can have several Runs (in that
# case you can give different values to selected parameters in each Run).

[Run 1]
description = "Run 1"
# the seeds for each of the RNGs
seed-0-mt = 10
seed-1-mt = 11100000000000
seed-2-mt = 211
seed-3-mt = 40
seed-4-mt = 50
seed-5-mt = 60
seed-6-mt = 70
seed-7-mt = 80
seed-8-mt = 90

SN.wirelessChannel.printDebugInfo = true

SN.node[*].networkInterface.Radio.txPowerLevelUsed = 2
SN.node[*].networkInterface.Radio.printDebugInfo = true

SN.node[*].networkInterface.Network.maxNumberOfParents = 1
SN.node[*].networkInterface.Network.rssiBased_NeighborQuality = true
SN.node[*].networkInterface.Network.neighbor_RSSIThreshold = -89.3 # in dBm

SN.node[3].nodeApplication.isSink = true
```

Referências

[1] CSIRO Sensor and Sensor Networks Research, Progress Report July 2006 to June 2007, 2007 (http://www.csiro.au/files/files/pi8m.pdf)

[2] Shi, X., Stromberg, G., "SyncWUF: An Ultra Low-Power MAC Protocol for Wireless Sensor Networks", *IEEE Transactions on Mobile Computing*, Vol. 6, No. 1, January 2007 (http://ieeexplore.ieee.org/iel5/7755/4016524/04016536.pdf)

[3] China's National Research Project on Wireless Sensor Networks, Ni, Lionel M.; Liu, Yunhao; Zhu, Yanmin; Shanghai Jiao Tong University and Hong Kong University of Science and Technology, IEEE Wireless Communications, Dezembro 2007 (http://ieeexplore.ieee.org/iel5/7742/4407216/04407230.pdf?tp=&arnumber=4407230&isnumber=4407216)

[4] Basile, C., Kalbarczyk Z., Iyer, R. K., "Inner-Circle Consistency for Wireless Ad Hoc Networks", *IEEE Transactions on Mobile Computing*, Vol. 6, No. 1, January 2007 (http://ieeexplore.ieee.org/iel5/7755/4016524/04016531.pdf?)

[5] Ruiz[1,2], Linner Beatrys, Correia[1,3], Luiz Henrique A., Vieira[1], Luiz Filipe M., Macedo[1], Daniel F., Nakamura[1], Eduardo F., Figueredo[1], Carlos M. S., Vieira[1], Marcos Augusto M., Mechelane[1], Eduardo Habib, Camara[1], Daniel, Loureiro[1], Antonio A. F., Nogueira[1], José Marcos S., Silva Jr.[4],Diógenes C., *Arquitectura para Redes de Sensores Sem Fio*, [1]Departamento de Ciência da Computação da UFMG, [2]Departamento de Informática da PUCPR, [3]Departamento de Ciência da Computação da UFLA, [4]Departamento de Engenharia Elétrica da UFMG, Brasil, (http://homepages.dcc.ufmg.br/~mmvieira/publications/04mc-sbrc.pdf, Jul 2006).

[6] http://www.tinyos.net/, Jul. 2008.

[7] http://nescc.sourceforge.net/, Jul. 2008.

[8] Carvalho, Leandro Ouriques M., Cunha, Pablo Salino, "Seminário de Redes de Computadores II - 802.15.4 e ZigBee", Brazil,14 de Julho de

2004, (http://www.gta.ufrj.br/~rezende/cursos/eel879/trabalhos/zigbee/Introduo(2).html)

[9] http://www.maxwell.lambda.ele.puc-rio.br/cgi-bin/PRG_0599.EXE/8044_4.PDF?NrOcoSis=23766&CdLinPrg=pt, Ago. 2008

[10] Luz, Giulian Dalton, *Roteamento em Redes de Sensores*, Instituto de Matemática e Estatística, Universidade de São Paulo, Brasil, Nov. 2004 (http://grenoble.ime.usp.br/movel/roteamentosensores.pdf).

[11] http://www.perl.org/, Jul. 2008

[12] http://www.xbow.com/Products/productdetails.aspx?sid=164, Jan. 2009.

[13] http://castalia-simulator.googlegroups.com/web/Castalia_MAC_ Analysis.pdf, Jul. 2009

[14] http://googlegroups.com/group/castalia-simulator/files, Jan. 2009