Dissertação de Mestrado em Engenharia Informática

Computação Móvel

Sensor-2-Sensor

Nuno Alexandre De Oliveira Moreira

Leiria, 2013

Dissertação de Mestrado em Engenharia Informática

Computação Móvel

Sensor-2-Sensor

Nuno Alexandre De Oliveira Moreira

Dissertação de Mestrado realizada sob a orientação do Professor Doutor António Pereira, Professor da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria e coorientação do Professor Doutor Nuno Costa, Professor da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria

Leiria, 2013

Aos meus pais
Abílio e Madalena

Esta página foi intencionalmente deixada em branco

Agradecimentos

Agradeço a todos os professores e funcionários da Escola Superior de Tecnologia e Gestão, que ao longo dos últimos anos contribuíram de forma direta ou indireta para o desenvolvimento desta dissertação.

Agradeço em especial aos Professores Doutores António Pereira e Nuno Costa pelo acompanhamento, orientação, apoio e motivação ao longo do desenvolvimento da dissertação e que em muito me ajudaram a atingir os objetivos propostos.

Agradeço ainda aos meus pais Abílio e Madalena, pelo suporte e coragem que me deram para conseguir ultrapassar mais uma fase da minha vida.

Esta página foi intencionalmente deixada em branco

Resumo

O avanço na indústria informática e eletrónica fez com que o uso e aplicabilidade das Redes Sensoriais Sem Fios (RSSF) tenham crescido notavelmente, permitindo o desenvolvimento de dispositivos de baixo consumo energético e com capacidades de comunicação sem fios.

Dada a elevada aplicabilidade das RSSF, é por vezes difícil identificar e definir as entidades necessárias para atingir um determinado objetivo aplicacional. Identificando as diversas entidades e as suas características poder-se-ão definir as funções e responsabilidades dos vários agentes que constituem a arquitetura, acabando por facilitar o desenvolvimento de aplicações.

Esta dissertação vem apresentar o estudo realizado com o intuito de identificar e definir as várias entidades necessárias para a implementação de uma arquitetura genérica que poderá servir para o uso de diversas aplicações.

Ao longo deste documento serão apresentadas as principais características de uma RSSF, efetuando um estudo e avaliação dessas características na tentativa de apresentar uma arquitetura, genérica e escalável, que melhor sirva as necessidades de uma variedade de aplicações.

Para comprovar a viabilidade da solução, foi desenvolvido um protótipo com as várias entidades que constituem a arquitetura proposta e testada a interação/comunicação entre elas.

Desta dissertação resulta a especificação da arquitetura proposta, identificando as várias entidades e definindo as suas funções, bem como, um protótipo que poderá ser usado como base para o desenvolvimento de aplicações.

Tendo em conta a especificação apresentada, conclui-se que recorrendo às entidades

indicadas e às suas principais funções e capacidades, é possível desenvolver um conjunto de aplicações tendo como base uma arquitetura genérica e escalável.

Palavras-chave: Redes, Arquitetura, Topologia, Entidades, Cluster, RSSF.

Abstract

The advancement of computer and electronics industry has made the use and applicability of wireless sensor networks (WSN) has grown remarkably, enabling the development of devices with low power consumption and wireless communication capabilities.

Given the high applicability of WSNs, it is sometimes difficult to identify and define the entities required to achieve a certain goal. By identifying the various entities and their characteristics is possible to define the roles and responsibilities of the various actors that constitute the architecture, facilitating the development of applications.

This dissertation describes the research in order to identify and define the various entities necessary for the implementation of a generic architecture that can serve for use in various applications.

Throughout this dissertation the main characteristics of a WSN will be presented, performing a study and evaluation of these characteristics in an attempt to present a generic and scalable architecture that best suits the needs of a variety of applications.

To prove the viability of the solution, a prototype was developed with the various entities that make up the architecture proposed and tested the interaction and communication between them.

This dissertation results in the architecture specification, identifying the various entities and defining their rules, as well as a prototype that can be used as a basis for the development of applications.

Given the specification presented, we conclude that using the mentioned entities and their main functions and capabilities, it is possible to develop a set of applications based on a generic and scalable architecture.

Keywords: Networks, Architecture, Topology, Entities, Cluster, WSN.

Esta página foi intencionalmente deixada em branco

Índice de Figuras

Esta página foi intencionalmente deixada em branco

Índice de Tabelas

Esta página foi intencionalmente deixada em branco

Lista de Acrónimos

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
AAA	Authentication, Authorization and Accounting
ACE	Algorithm for Cluster Establishment
AES	Advanced Encryption Standard
AODV	Ad-Hoc On-Demand Distance Vector
API	Application Programming Interface
APTEEN	The Adaptive Threshold sensitive Energy Efficient sensor Network protocol
ATE	Análise Temporária de um Espaço
BCDCP	Base-Station Controlled Dynamic Clustering Protocol
BS	Base-Station
BWRC	Berkeley Wireless Research Center
CCS	Concentric Clustering Scheme
CH	Cluster-Head
CHIRON	Chain-Based Hierarchical Routing Protocol
CL	Chain-Leader
CM	Cluster-Member

CMOS	Complementary Metal–Oxide–Semiconductor
CORMOS	C# Open Source Operating System
COSEN	Chain Oriented Sensor Network
CS	Control-Station
CSMA-CA	Carrier Sense Multiple Access with Collision Avoidance
DE	Deteção de Eventos
DSSS	Direct Sequence Spread Spectrum
DWEHC	Distributed Weight-based Energy-efficient Hierarchical Clustering protocol
EECS	Energy Efficient Clustering Scheme
EEUC	Energy-Efficient Uneven Clustering
ESTG	Escola Superior de Tecnologia e Gestão
FIFO	First In, First Out
GPS	Global Positioning System
GSM	Global System for Mobile communications
GTS	Guarantee Time Slots
HART	Highway Addressable Remote Transducer Protocol
HEED	Hybrid Energy-Efficient Distributed clustering
HGMR	Hierarchical Geographic Multicast Routing
HTTP	Hypertext Transfer Protocol
ID	Identificador

IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISA	International Society of Automation
ISM	Industrial, Scientific and Medical
ITALH	Information Technology For Assisted Living at Home
LEACH	Low-energy Adaptive Clustering Hierarchy
MAC	Media Access Control
MANET	Mobile Ad-hoc Network
MEMS	Microelectromechanical Systems
MESSAGE	Mobile Environmental Sensing System Across Grid Environments
MIT	Massachusetts Institute of Technology
MR	Message Reader
MS	Main-Station
NCAP	Network Capable Application Processor
NEST	Networked Embedded Software Technology
OSI	Open Systems Interconnection
PC	Personal Computer
PDA	Personal Digital Assistant
PEGASIS	Power-Efficient Gathering in Sensor Information Systems

QoS	Quality of Service
RAM	Random-Access Memory
RF	Radiofrequência
RK	Resource-Kernel
ROM	Read-Only Memory
RSSF	Rede Sensorial Sem Fios
SF	Serial Forwarder
SG	Sink-Gateway
SMS	Short Message Service
SN	Sensor-Node
SNR	Signal-to-Noise Ratio
SoC	System on Chip
SPIN	Sensor Protocols for Information via Negotiation
SPOF	Single Point Of Failure
SRAM	Static Random-Access Memory
TEDS	Transducer Electronic Data Sheets
TEEN	Energy Efficient sensor Network protocol
TL-LEACH	Two-Level Hierarchy LEACH
TSMP	Time Synchronized Mesh Protocol
TTDD	Two-Tier Data Dissemination

UAV	Unmanned Aerial Vehicle
UCLA	University of California, Los Angeles
UCS	Unequal Clustering Size
UDP	User Datagram Protocol
WBAN	Wireless Body Area Network
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network
ZC	ZigBee Coordinator
ZED	ZigBee End Device
ZR	ZigBee Router

Esta página foi intencionalmente deixada em branco

Índice

Esta página foi intencionalmente deixada em branco

1 Introdução

O notável avanço na indústria informática e eletrónica tem vindo a permitir que novas tecnologias sejam usadas na criação de dispositivos e equipamentos com dimensões e consumo energético cada vez mais reduzidos. A composição entre microprocessadores/microcontroladores, módulos de comunicação e elementos sensoriais permitiu o desenvolvimento do conceito de Redes Sensoriais Sem Fios (RSSF).

Uma RSSF consiste num sistema distribuído, constituído por vários elementos cujo principal objetivo é a recolha e encaminhamento dos dados recolhidos. Este sistema em rede é constituído por dispositivos autónomos que cooperam entre si e que se encontram dispersos em diferentes locais de modo a monitorizar, por intermédio de sensores, diversos fenómenos físicos, como por exemplo temperatura, movimento ou som.

Recorrendo a RSSF podem desenvolver-se soluções revolucionárias em diversas áreas. As RSSF possuem a capacidade de recolha de dados acerca de vários fenómenos físicos naturais podendo estes depois ser tratados no mundo digital.

Com o crescimento do interesse neste tipo de soluções surgiram linguagens e métodos de programação orientados a RSSF, facilitando assim, o desenvolvimento de novas aplicações baseadas em RSSF.

1.1 Motivação

Dada a elevada aplicabilidade das RSSF, é por vezes difícil identificar e definir as entidades necessárias para atingir um determinado objetivo aplicacional. Para além da definição das entidades presentes numa dada arquitetura é também importante definir a estrutura da rede, definindo hierarquias e aplicando as regras e métodos de comunicação necessários para que se retire a melhor performance da RSFF.

Identificando as diversas entidades e as suas características poder-se-á definir as funções e responsabilidades dos vários agentes que constituem a arquitetura, acabando por facilitar o desenvolvimento de aplicações para RSSF.

1.2 Objetivos e Contribuições

Tendo em conta diversas características nucleares que definem uma RSSF, como por exemplo a topologia ou as responsabilidades a cargo de cada entidade, os objetivos desta dissertação resumem-se da seguinte forma:

- Identificação das entidades necessárias para a implementação de uma arquitetura genérica e escalável;
- Definição das características e funções de cada entidade;
- Apresentação de métodos para interoperação entre as diversas entidades;
- Desenvolvimento de um protótipo que demonstre a funcionalidade da arquitetura e a comunicação entre as suas entidades.

Com o trabalho desenvolvido na tentativa de atingir os objetivos da dissertação, foram efetuadas diversas consultas e análises, acabando assim por contribuir com os seguintes pontos:

- Identificação, estudo e apresentação de diversos temas e opções relevantes para a otimização da arquitetura e performance das aplicações de RSSF, nomeadamente:
    - Levantamento das principais características e antepassado tecnológico das RSSF;
    - Estudo acerca das características das diversas topologias usadas em RSSF e suas vantagens e desvantagens;
    - Levantamento das diversas técnicas que definem o processo de *clustering* em RSSF;
    - Apresentação de temas e opções relevantes para a otimização da arquitetura e performance das aplicações;

- Arquitetura modelo para o desenvolvimento de aplicações, identificando as entidades envolvidas, apresentando as suas características e definindo as suas funções;
- Protótipo que pode ser usado como base no desenvolvimento de aplicações para RSSF.

1.3 Estrutura da Dissertação

No seguimento deste capítulo introdutório irá ser apresentada, no segundo capítulo, a revisão da literatura onde se pode ver a aplicabilidade e principais características das RSSF.

No terceiro capítulo são definidos os requisitos da solução, o estudo e avaliação das diversas abordagens possíveis, a apresentação e definição das entidades presentes na solução.

No quarto capítulo é apresentada a implementação do protótipo desenvolvido e os resultados dos testes realizados são apresentados no quinto capítulo.

A conclusão da dissertação será efetuada no sexto capítulo, onde se apresentam também as contribuições e indicações para trabalho futuro.

Esta página foi intencionalmente deixada em branco

2 Revisão da Literatura

Neste capítulo serão apresentados alguns dos projetos existentes na área a ser estudada nesta dissertação bem como um resumo acerca da história das Redes Sensoriais Sem Fios (RSSF) e das suas principais características. Numa segunda fase, serão apresentadas em maior detalhe algumas das soluções existentes para a componente de rede de uma RSSF.

2.1 Visão Geral das Redes de Sensores Sem Fios

O notável avanço na indústria eletrónica tem permitido que novas tecnologias sejam usadas na criação de dispositivos e equipamentos com dimensões e custos cada vez mais reduzidos. A composição entre microcontroladores, elementos sensor, módulos de comunicação e gestores de energia permitiu um grande avanço no desenvolvimento do conceito de RSSF, permitindo uma ligação ubíqua entre os vários nós-sensor que constituem a RSSF.

As RSSF possuem as características normalmente encontradas nas diversas tecnologias de redes sem fios, adicionando a capacidade de aquisição de dados sensoriais [1]. Recorrendo a esta tecnologia podem ser desenvolvidas soluções revolucionárias em diversas áreas. As RSSF possuem a capacidade de recolher dados acerca de fenómenos físicos naturais podendo estes depois ser tratados digitalmente. Entre outras vantagens, as RSSF podem, por exemplo, facilitar a recolha de dados em locais de difícil acesso.

Uma RSSF é tipicamente um sistema distribuído, constituído por vários nós-sensor, cujo principal objetivo é recolher e entregar dados. Este sistema em rede é constituído por dispositivos autónomos que cooperam entre si e que se poderão encontrar dispersos em diferentes locais de modo a monitorizar, por intermédio de sensores, diversos fenómenos físicos, como por exemplo, temperatura, movimento ou som. Várias empresas como a Ember [2], Shimmer [3] ou Memsic [4] oferecem um vasto leque de dispositivos para comercialização. Com o crescimento das capacidades no equipamento disponível, o interesse

neste tipo de soluções fez com que o aparecimento de linguagens e métodos de programação orientados a este tipo de dispositivos se tornasse uma realidade. Cada nó da rede é tipicamente equipado com um *transceiver* de rádio frequência, por um microcontrolador e por uma fonte de energia, tipicamente uma bateria. Um nó-sensor pode variar em tamanho, desde o tamanho de um grão de areia até ao tamanho de uma caixa de sapatos. O custo dos nós-sensor é igualmente variável, desde alguns cêntimos até centenas de euros, dependendo do tamanho da rede e da complexidade exigida. Determinadas características do equipamento, como por exemplo, a memória, a velocidade de processamento, o módulo de comunicação ou os requisitos de energia, influenciam o preço final de um nó-sensor.

As RSSF são aplicadas em diversas áreas, nomeadamente na área industrial e civil, são usadas no controlo de processos industriais, na vigilância do estado de equipamento, na monitorização do meio ambiente, no apoio a serviços de saúde, na automação residencial entre outras. Seguidamente são apresentadas algumas das características das RSSF.

2.1.1 Características da Rede

Uma RSSF tipicamente consiste num conjunto de nós-sensor, multifuncionais e de baixo custo. Estes nós de tamanho reduzido, estão equipados de forma a que possam processar e transmitir a informação que recolhem. A comunicação entre os vários nós é feita a curta distancia através de um meio de comunicação sem fios. Quando comparada com as tradicionais redes sem fios, as RSSF contam com as seguintes características:

- **Elevada Densidade:** Os nós-sensor são colocados no local que se pretende analisar e podem atingir números muito superiores aos que encontramos nas Mobile Ad-hoc Network (MANET);

- **Consumo Energético:** Os nós-sensor são tipicamente alimentados por baterias. Em certas aplicações o local de instalação é hostil, sendo por vezes difícil, ou mesmo impossível, a recarga/substituição das baterias;

- **Parcos Recursos:** Os nós-sensor são limitados nas suas capacidades energéticas, de processamento e armazenamento;

- **Autoconfiguração:** Em muitas aplicações os nós-sensor são instalados de forma aleatória, sendo necessário que os nós-sensor formem uma rede de comunicação de

forma independentemente;

- **Especificidade da Aplicação:** As RSSF são desenvolvidas para uma aplicação específica. Os requisitos da RSSF mudam consoante a aplicação;
- **Falhas de Hardware:** Os nós-sensor são por vezes instalados em ambientes hostis e sem supervisão, podendo assim ocorrer falhas físicas ou até roubo do *hardware*;
- **Alteração de Topologia:** A topologia de rede pode mudar frequentemente devido a falhas de nós-sensor causadas por danos físicos, falta de energia ou condicionamento do canal de comunicação;
- **Identificação Global:** Devido ao elevado número de nós-sensor usado em determinadas aplicações, é por vezes impossível usar um esquema de endereçamento global, uma vez que esta abordagem traria uma grande sobrecarga ao sistema;
- **Fluxo de Tráfego:** Na maioria das RSSF, os dados recolhidos pelos vários nós-sensor são enviados para o ponto de saída da rede, exibindo um padrão de comunicação "muitos-para-um";
- **Redundância dos Dados:** Dada a abundância de nós-sensor usados, muita da informação recolhida pode acabar por ser redundante.

As características únicas das RSSF apresentam desafios no desenho e na implementação da rede de sensores.

2.1.2 Aplicações

A ideia de pequenos dispositivos autónomos que colaboraram entre si é já usada há algum tempo. A pesquisa inicial em RSSF, tal como em muitas outras tecnologias, foi motivada por aplicações militares, financiando uma série de projetos de investigação que em muito impulsionaram o crescimento das RSSF [5].

Em 1967 no âmbito do projeto *"Igloo White"* foi desenvolvida uma RSSF com a capacidade de analisar dados sísmicos e acústicos, com o intuito de detetar movimento inimigo [6].

Em 2001, o projeto *Smart Dust*, desenvolveu um sistema autónomo de deteção e comunicação baseado num dispositivo com a dimensão de apenas um milímetro cúbico [7].

Em 2002, o projeto *"Networked Embedded Software Technology*" (NEST) iniciou o desenvolvimento de uma plataforma experimental de *software* e hardware para tecnologias de RSSF [8]. Estes projetos são comummente considerados como o berço da investigação e desenvolvimento de RSSF [5].

O desenvolvimento na indústria eletrónica possibilita a existência de um grande número de projetos nesta área. As aplicações para RSSF são variadas, geralmente envolvendo monitorização e/ou controlo.

2.1.2.1 Monitorização de Áreas e Espaços

A monitorização de uma determinada área é uma aplicação bastante comum em RSSF. O conjunto de nós-sensor é instalado numa região onde determinado fenómeno ou grandeza é alvo de monitorização.

Um exemplo deste tipo de aplicação é o projeto "Alternatives to Landmines" onde uma grande quantidade de nós-sensor é implantada ao longo de um campo de batalha, tendo como objetivo detetar intrusões do inimigo em vez de usar minas antipessoais [9].

O projeto "DARPA SensIT" tem como objetivo detetar intrusos e monitorizar veículos em estradas ou terrenos abertos, recorrendo a um veículo aéreo não tripulado (UAV). Estabelecendo uma rede de comunicação entre os nós-sensor no chão é possível detetar veículos que passem na área que se pretende monitorizar. É também possível estimar a direção e a velocidade dos veículos e os dados recolhidos são registados e enviados para o UAV, que por sua vez os enviará até à estação base [10].

2.1.2.2 Monitorização Ambiental

A monitorização ambiental é outra das aplicações das RSSF. Um dos projetos exemplo na área ambiental é o *"Mobile Environmental Sensing System Across Grid Environments"* (MESSAGE) e tem como objetivo distribuir por algumas cidades de Inglaterra, nós-sensor sem fios para medir a poluição do ar. Os dispositivos serão colocados em carros, autocarros, bicicletas e pessoas e são medidos os níveis de monóxido de carbono e de dióxido sulfúrico emitidos pelo trânsito, sendo os dados recolhidos transmitidos através de telemóvel. Os nós-sensor mais pequenos podem ser transportados pelas pessoas e ligados aos seus próprios telemóveis. Os maiores, capazes de monitorizar também a temperatura e os níveis de ruído, poderão ser colocados em postes de iluminação ou em sinais de trânsito [11].

Outro exemplo é o projeto "PermaSense", que tem como principal objetivo construir e personalizar um conjunto de unidades de medição para uso em áreas remotas com condições ambientais adversas. O segundo objetivo é a recolha de dados ambientais que ajudem a compreender os processos que ligam as alterações climáticas e a queda de rochas em áreas de *permafrost* [12].

Serve também de exemplo nesta área o projeto "GlacsWeb". O objetivo deste projeto é monitorizar o comportamento glaciar recorrendo a RSSF. Os dados recolhidos incluem a temperatura, pressão, tempo e movimento sub-glaciar. Os nós-sensor são instalados a diferentes profundidades, dentro de cápsulas protetoras, no interior de glaciares, para estudar a sua deslocação e a sua dinâmica interna. Estão equipados com sensores de pressão, temperatura, inclinação, comunicando diretamente com uma estação base que está no topo do glaciar. A estação base mede o deslocamento supra-glaciar com o uso de um Global Positioning System (GPS) diferencial e envia os dados adquiridos via Global System for Mobile communications (GSM) [13].

2.1.2.3 Controlo de Processos Industriais

Na área industrial as RSSF são usadas em variadíssimas aplicações, desde a indústria química, petrolífera, nuclear, entre muitas outras. Um exemplo do uso na área industrial é a deteção de veículos, onde as RSSF usam uma variedade de sensores para detetar a presença e localização exata de veículos que vão desde motos até comboios. É possível colocar um nó-sensor com um módulo GPS em cada veículo, sendo as coordenadas da posição atual analisadas em tempo real.

Outro exemplo de aplicação é o controlo de temperatura nas indústrias alimentar e farmacêutica, em que os nós-sensor são equipados com sensores de temperatura para analisar e controlar a temperatura ambiente.

O uso de RSSF na indústria agrícola é também cada vez mais comum onde, por exemplo, os sistemas de rega podem ser controlados por meio de transmissores de pressão para controlar os níveis dos reservatórios de água, as bombas podem ser controladas usando dispositivos sem fios e o uso da água pode ser controlado por um centro de controlo. A automação do sistema de rega permite o uso mais eficiente da água e reduz o desperdício. As RSSF são também usadas para controlar a temperatura e os níveis de humidade no interior de estufas. Quando a temperatura e a humidade fica abaixo dos níveis pretendidos, o responsável pela

estufa pode ser notificado via *email* ou Short Message Service (SMS) e posteriormente acionar sistemas de nebulização e ventilação ou controlar uma grande variedade de respostas do sistema [14].

Vários projetos, nomeadamente na monitorização de infraestruturas, estão a ser desenvolvidos. É exemplo disso um projeto da universidade do Texas, onde são usados nós-sensor para monitorização do comportamento de pontes usando equipamento de baixo custo, móvel, escalável e com pouco consumo de energia, onde o objetivo é averiguar o desgaste e degradação da construção de modo a prevenir eventuais acidentes [15].

2.1.2.4 Aplicações na Área da Saúde

A área da saúde é também uma das áreas com grande margem de aplicabilidade por parte das RSSF, existindo muitos projetos nesta área. Um dos projetos conhecidos em Portugal é o projeto "Smart-Clothing" que tem como objetivo principal impulsionar a economia regional (do centro de Portugal) na área de conhecimento e inovação tecnológica, utilizando novos materiais, sensores embutidos e sistemas avançados de comunicação para o desenvolvimento de protótipos de têxteis inteligentes. O desenvolvimento dos protótipos combina investigação em sensores, têxteis funcionais e redes de comunicação sem fios no contexto do corpo humano. Um sistema de comunicação hierárquico é utilizado para fornecer os dados da Wireless Body Area Network (WBAN) instalada no corpo de uma mulher grávida [16].

O projeto "Mercury" consiste numa plataforma baseada em equipamento Shimmer [3] onde são analisados os movimentos de pacientes em tratamento de distúrbios neurológicos como a doença de Parkinson, epilepsia ou vítimas de acidente vascular cerebral. Os pacientes usam até oito nós-sensor equipados com sensores para a monitorização de condições fisiológicas. Os pacientes usam os sensores continuamente durante um intervalo entre doze a dezoito horas por dia [17].

O projeto *"Information Technology For Assisted Living at Home"* (ITALH) presta assistência médica à distância, ou seja, na própria casa do paciente. Permite o acompanhamento de doentes com doenças congénitas, como epilepsia ou diabetes, melhorando a qualidade de vida dos mesmos, implementando medicina preventiva, com auxílio da RSSF. Este projeto serve também para ajudar em ensaios clínicos executados por médicos, farmacêuticos, biólogos ou outros agentes da saúde [18].

A atividade desportiva é também uma área com grande margem de aplicabilidade. Recorrendo a RSSF poderão ser monitorizados parâmetros neurológicos, musculares e cardíacos resultantes da intensa prática desportiva. Por exemplo, imagine-se um bombeiro em ação, está prestes a entrar em colapso depois de várias horas exposto ao calor de um incêndio, antes que isso aconteça, a camisola que tem vestida vai enviar um sinal de alerta para um Personal Digital Assistant (PDA) ou Smartphone do líder da equipa. Este é um dos principais objetivos do projeto "Vital Responder" [19].

2.1.2.5 Aplicações na Área da Domótica

As RSSF podem ser usadas para criar ambientes domésticos mais convenientes e inteligentes.

Com o conceito de "Smart-Home", os nós-sensor podem ser instalados em vários pontos da casa formando uma rede autónoma. Por exemplo, um frigorífico inteligente em comunicação com um forno ou micro-ondas inteligente pode preparar uma refeição tendo por base o inventário disponível, enviando parâmetros importantes à preparação da refeição para o micro-ondas ou forno, definindo a temperatura e tempo de confeção adequados [20].

A monitorização remota é outra aplicação onde as RSSF têm grande aplicabilidade. A rede doméstica de nós-sensor pode ser usada para que, remotamente, sejam lidos valores e parâmetros relacionados com a temperatura e consumo de eletricidade, entre outros [21] [22].

2.1.3 Objetivos da Rede

As características e requisitos particulares a uma dada aplicação têm um impacto decisivo no *design* de toda a RSSF, influenciando as suas capacidades e performance geral. O principal objetivo do *design* da rede engloba os seguintes aspetos:

- **Tamanho reduzido do nó-sensor:** Reduzir o tamanho do nó-sensor é um dos objetivos principais aquando da construção da RSSF. A instalação dos nós é geralmente feita em ambientes hostis e em grande número. Reduzindo o tamanho dos nós-sensor pode facilitar a instalação do nó-sensor, por exemplo, através do ar;

- **Baixo Custo:** A redução do custo do nó-sensor é outro objetivo a ser levado em conta num projeto de RSSF. Dada a natureza de utilização e o ambiente de instalação, grandes quantidades de nós podem ser danificados frequentemente, ou até roubados, impossibilitando a sua reutilização;

- **Baixo Consumo Energético:** Reduzir o consumo energético é talvez o maior objetivo na conceção de RSSF. Uma vez que os nós-sensor são geralmente alimentados por baterias, muitas vezes difíceis de substituir, é crucial reduzir o consumo para que o tempo de vida do nó-sensor seja o mais elevado possível;

- **Autoconfiguração dos nós-sensor:** Em várias implementações de RSSF, os nós-sensor são instalados sem obedecer a qualquer tipo de planeamento prévio e à possibilidade de falha de nós, seja por avaria ou falta de bateria, é também uma possibilidade. Desta forma, os nós-sensor devem ter a capacidade de se autoconfigurarem de maneira a que seja criada uma rede de comunicação entre os vários nós-sensor que constituem a RSSF;

- **Escalabilidade:** O número de nós-sensor de uma RSSF pode chegar à ordem das dezenas, centenas ou milhares. Neste sentido, os protocolos de rede desenvolvidos devem ser escaláveis, de maneira a permitir que a rede possa suportar o elevado número de nós-sensor caso seja necessário;

- **Adaptabilidade:** A falha, adição, ou deslocação de um nós-sensor numa RSSF resulta em mudanças na densidade e topologia da rede. Assim, os protocolos de rede devem também permitir a adaptabilidade às condições da rede num dado momento;

- **Confiabilidade:** Em determinas aplicações de RSSF, exige-se que a entrega dos dados recolhidos seja eficaz, rápida e integra. Para que estas exigências sejam atendidas, os protocolos de rede devem fornecer mecanismos de controlo e correção de erros, para que seja garantida a fiabilidade na entrega dos dados;

- **Tolerância a falhas:** Os nós-sensor são propensos a falhas em determinadas aplicações. Devem existir mecanismos de tolerância a falhas que permitam efetuar autotestes, autocalibrarão e autorrecuperação [23];

- **Segurança:** Em cenários onde os nós-sensor estão acessíveis por pessoas não autorizadas, a RSSF deve introduzir mecanismos de segurança eficazes de maneira a evitar que os dados da rede e do nó-sensor sejam comprometidos;

- **Utilização do canal:** As RSSF têm largura de banda limitada. Os protocolos de comunicação projetados devem eficientemente utilizar a largura de banda de forma a

melhorar a utilização do canal;

- **Suporte para QoS:** Dependendo da aplicação, os requisitos de qualidade de serviço (QoS) podem mudar no que diz respeito à latência e perda de pacotes. Aplicações críticas não podem permitir elevada latência ou elevada perda de pacotes. Desta forma, os protocolos de rede devem estar conscientes dos requisitos de QoS exigidos pela aplicação.

A maioria das RSSF são pensadas para uma dada aplicação e têm requisitos específicos face a essa aplicação. Por outro lado, é inadequado tentar implementar todos os objetivos apresentados. Deve-se sim tentar implementar os objetivos de *design* que permitam à RSSF atingir, da melhor forma, o objetivo da aplicação.

2.1.4 Desafios da Rede

Dadas as características únicas das RSSF, são encontrados vários desafios de *design*, envolvendo os seguintes aspetos:

- **Baixa Capacidade Energética:** Os nós-sensor são alimentados por bateria, tendo assim um tempo de vida limitado. Esta restrição apresenta muitos desafios no desenvolvimento de *hardware*, *software* e na arquitetura protocolar de RSSF. Para prolongar o tempo de vida do nó-sensor, e consequentemente o tempo de vida de toda a rede, a utilização de energia deve ser levada em conta em todos os aspetos do *design* da rede, não apenas no *hardware* e *software*, mas também na arquitetura de rede e protocolos;

- **Limitação de Recursos de Hardware:** A capacidade de processamento e armazenamento dos nós-sensor é baixa. Assim, o processamento de determinadas funcionalidades computacionais é limitado. Estas limitações apresentam muitos desafios no desenvolvimento de *software* da aplicação e também no desenvolvimento dos protocolos de rede;

- **Implementação massiva e aleatória:** A maioria das RSSF consiste num elevado número de nós-sensor. A forma de instalação dos nós-sensor é dependente da aplicação, podendo ser manual e planeada, ou completamente aleatória. Os nós-sensor devem-se organizar de forma autónoma, construindo uma rede de comunicação para

que possam efetuar a tarefa que lhes foi atribuída;

- **Ambiente dinâmico e não confiável:** A topologia de rede pode mudar com frequência devido a falhas de nós, danos, energia ou adição de novos nós. Por outro lado, os nós-sensor estão ligados entre si usando um meio sem fios, que é suscetível a ruído e propenso a erros. A conectividade da rede pode ser frequentemente interrompida devido a falhas no canal ou atenuação de sinal;

- **Diversidade de Aplicações:** As RSSF têm uma ampla aplicabilidade. Os requisitos da rede podem variar significativamente consoante a aplicação pretendida, sendo impossível a existência de um protocolo que abranja todas as características das diversas aplicações.

2.2 Evolução Tecnológica

O conceito de rede de sensores foi originalmente introduzido há cerca de três décadas [24]. Nessa altura, este conceito foi mais uma visão do que uma tecnologia pode ser explorando outras tecnologias (Sensores, Computadores, Comunicações Sem Fios). Como resultado, a sua aplicação foi limitada principalmente a grandes sistemas militares. No entanto, os recentes avanços tecnológicos em Microelectromechanical Systems (MEMS), comunicações sem fios e baixo custo, associados à lei de Moore [25], permitiram o desenvolvimento de nós-sensor de tamanho reduzido e de baixo custo, em particular o desenvolvimento de RSSF.

2.2.1 MEMS

MEMS [26] [27] é uma tecnologia-chave para o fabrico de nós-sensor pequenos, de baixo-custo e baixo consumo. Baseia-se em técnicas de *"micromachining"* que têm sido desenvolvidas para fabricar componentes mecânicos de tamanho reduzido controlados eletricamente. Através de processos integrados, estes componentes eletromecânicos podem constituir sistemas complexos, dando origem a um nó-sensor de tamanho reduzido. Ao utilizar a tecnologia MEMS, muitos componentes de nó-sensor podem ser miniaturados, por exemplo, sensores de comunicação, blocos e unidades de fornecimento de energia. A fabricação em elevadas quantidades leva a uma significativa redução dos custos de fabricação.

2.2.2 Tecnologia de Comunicação Sem Fios

A comunicação sem fios é uma tecnologia-chave para permitir o funcionamento de uma RSSF. Nas últimas décadas, a comunicação sem fios tem sido extensivamente estudada para utilização nas redes convencionais, permitindo assim avanços significativos. No que à camada física diz respeito, foram desenvolvidos vários métodos de modulação, sincronização e de antenas, de modo a satisfazer os diferentes tipos de rede e de aplicações. Nas camadas superiores foram desenvolvidos diversos protocolos com o intuito de resolver vários problemas típicos em comunicações sem fios, como por exemplo, controlo de acesso ao meio, encaminhamento, QoS ou segurança da rede. Estas técnicas de comunicação fornecem uma boa base tecnológica para o desenvolvimento de RSSF. No entanto, a maioria dos protocolos de comunicação para as convencionais redes sem fios, por exemplo, sistemas de comunicação móvel (GSM), redes de área local sem fios (WLAN), redes de área pessoal (WPAN) e MANET, não levam em conta as características únicas das RSSF, em particular, a restrição de energia dos nós-sensor. Desta forma, estes protocolos não podem ser aplicados diretamente às RSSF. Um novo conjunto de protocolos de rede são necessários para tratar das questões de rede particulares às características únicas que se encontram nas RSSF.

2.2.3 Hardware

As plataformas de *hardware* de nós-sensor podem ser classificadas em três categorias [28]:

- **PC de propósito Geral:** Esta classe inclui vários Personal Computer (PC) de baixo consumo (Ex: PC104) e PDA. Normalmente executam os típicos sistemas operativos usados nesta plataforma, por exemplo, Windows CE, Linux ou sistemas operativos de tempo real. Usam também os protocolos de comunicação sem fio padrão, tais como, IEEE 802.11 ou Bluetooth. Comparando com nós-sensor dedicados ou "System on a Chip" (SoC), estes PC têm maior capacidade de processamento, podendo assim incorporar um conjunto de protocolos de rede mais rico, linguagens de programação populares, *middleware*, interfaces de programação de aplicação (API) e outros conjuntos de *software* típicos. No entanto, requerem maiores capacidades de energia;

- **Nó-Sensor dedicado:** Esta classe de plataformas inclui a família de nós-sensor Berkeley [29], a família UCLA Medusa [30] e μAMP do MIT [31], que por norma, usam chips comuns. São caracterizados pelo seu tamanho reduzido, baixo poder de

processamento e de comunicação, e interfaces de sensor simples;

- **Nó-Sensor System on a Chip (SoC):** Esta classe de plataformas inclui o nó-sensor Smart Dust [7] e o BWRC PicoNode [32], tendo por base a tecnologia "Complementary Metal–Oxide–Semiconductor" (CMOS), MEMS e Radiofrequência (RF). Têm o objetivo de consumir o mínimo de energia, oferecendo capacidades de computação, comunicação e aquisição sensorial.

Entre todas as plataformas de *hardware* apresentadas, o nó-sensor Berkeley tornou-se bastante popular entre a comunidade de pesquisa em RSSF, devido às suas dimensões reduzidas, software de código aberto e disponibilidade comercial [28].

O *hardware* usado em RSSF é um elemento muito importante na performance de toda a rede. A escolha do *hardware* depende das necessidades de cada aplicação. Com a evolução das RSSF, o aparecimento de novo equipamento, com maiores capacidades e de menor custo é cada vez mais visível.

2.2.3.1 Nó-Sensor

Muitas das plataformas usadas em RSSF são baseadas em micro controladores comuns, tais como, o MSP430 da Texas Instruments ou o ATmega128L da Atmel. Estes microcontroladores são projetados para operar em baixa potência e são fundamentalmente de base monolítica e de propósito geral. Suportam estados de baixo consumo, consumindo menos de 5uA (MSP430).

- **Mica2**

O Mica2 [33] é baseado no microcontrolador Atmel ATmega128L, de 8-bit, e tem sido amplamente utilizado. O microcontrolador ATmega128L funciona a 7.3MHz e conta com 128KB de memória (ROM) para código e 4KB (RAM) para dados. Recorre ao rádio CC1000 da ChipCon, operando na frequência de 433MHz ou na de 868/915MHz. Existe também o MicaZ [34] da empresa Memsic [4], que é em tudo idêntico ao Mica2 exceto no rádio usado. O MicaZ usa o chip CC2420 também da ChipCon e opera nos 2.4GHz.

Figura 1 - Mica2 / MicaZ

Existem ainda outros dispositivos disponíveis e vulgarmente utilizados pela comunidade científica, sendo eles o Epic [35], Iris [36] TelosB [37], Sun SPOT [38], etc.

- **iMote2**

O *iMote2* [39] fornece uma plataforma de alto desempenho para uso em diversas aplicações. Conta com um microcontrolador de baixo consumo XScale PXA271, de 32-bit, onde a frequência de operação que pode ir até aos 416MHz. Conta também com rádio compatível com a norma IEEE 802.15.4 e com antena de 2.4GHz embutida. Está equipado com 256KB de memória SRAM *on-chip*, 32 MB de SDRAM e 32 MB de memória FLASH.

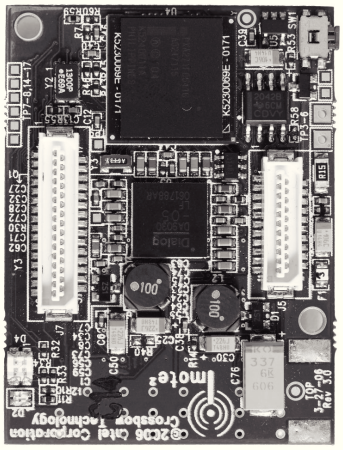

Figura 2 - iMote2

2.2.3.2 Placas Sensoriais

Para que os nós-sensor possam adquirir dados sensoriais, são necessárias placas com sensores embutidos. As placas disponíveis dispõem de uma enorme variedade de sensores que vão

desde termómetro, acelerômetro e osciloscópio até magnetómetros e GPS. Seguidamente serão apresentadas algumas placas sensoriais disponíveis para os nós-sensor Mica2/MicaZ e iMote2.

- **Mica2 e MicaZ**

A placa MTS310 [40] oferece um sensor de luz, termómetro, magnetómetro, acelerômetro de dois eixos, microfone e um *Buzzer*.

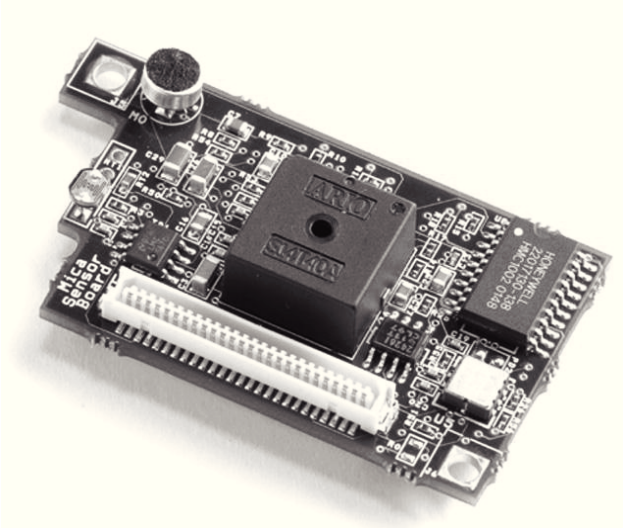

Figura 3 - MTS310

A placa MTS420 [41] é indicada para análises climáticas, já que inclui sensores de luz, temperatura, humidade, pressão e sísmico e suporta também GPS.

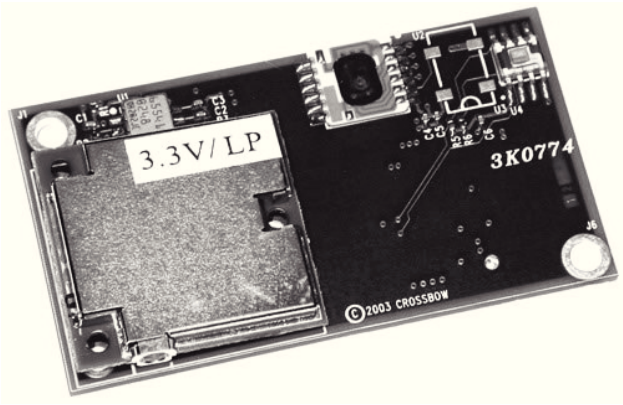

Figura 4 - MTS420

- **iMote2**

Para a plataforma iMote2 existem também placas sensoriais disponíveis, entre as quais se destaca a ITS400 [42] com sensores de humidade, temperatura, luz e acelerômetro de três eixos.

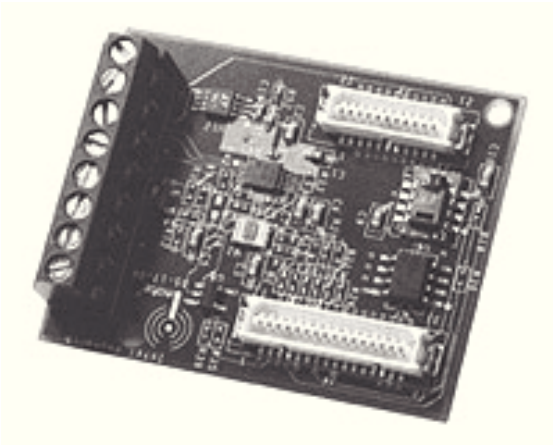

Figura 5 - ITS420

Também é disponibilizada a placa IMB400 [43], com câmara de vídeo integrada, suportando a captura de vídeo e áudio, tornando-a adequada para aplicações onde a monitorização de movimento e/ou som é necessária.

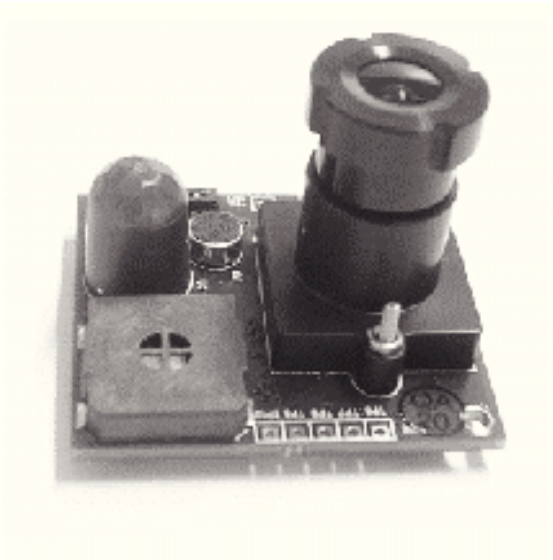

Figura 6 - IMB400

2.2.4 Software

Uma plataforma de *software* pode ser um sistema operativo que fornece um conjunto de serviços para aplicações, incluindo gestão de ficheiros, memória, tarefas, dispositivos e comunicação ou pode ser uma plataforma de programação que fornece uma biblioteca de componentes para desenvolvimento de aplicações [28]. Algumas da plataformas para RSSF incluem, por exemplo, o TinyOS [44], nesC [45], TinyGALS [46] e Maté [47].

O TinyOS é um dos primeiros sistemas operativos a suportar aplicações de rede para nós-sensor de parcos recursos. É um sistema operativo baseado em eventos e usa apenas 178 bytes de memória, suportando comunicação, multitarefa, e modularidade. Não tem sistema de ficheiros, suporta apenas alocação de memória estática, implementa um gestor de tarefas simples e fornece abstrações de rede mínimas.

O nesC é uma extensão da linguagem C para apoiar o projeto TinyOS. Fornece um conjunto de construções de linguagem e restrições para implementar componentes do TinyOS e das suas aplicações.

TinyGALS é uma linguagem para TinyOS que fornece uma maneira de construir eventos numa forma desencadeada.

Maté é uma máquina virtual para os nós-sensor Berkeley. Define instruções de máquina virtual para abstrair operações comuns, como por exemplo, acesso a estados internos. Desta forma, o *software* escrito em Maté não precisa de ser reescrito para acomodar uma nova plataforma de hardware.

O *software* tem um papel muito importante nas RSSF. A escolha do sistema operativo (SO), modo de operação e outras características influenciam a performance de toda a aplicação.

2.2.4.1 Sistemas Operativos

Um SO é composto por diversas características, desde a arquitetura, modelo de execução, entre outras. Na Tabela 1 são indicadas as arquiteturas de SO tipicamente usadas em RSSF.

Arquitetura	Funcionamento
Monolítica	Aplicações e componentes necessárias ao SO numa imagem de sistema única
Modular	Aplicações e SO são construídas de forma a interagir modularmente
Máquina Virtual	Aplicação construída como um conjunto de componentes estáticas e dinâmicas

Tabela 1 - Arquiteturas de Sistemas Operativos [48]

Na Tabela 2 são apresentadas algumas das características dos SO usados em RSSF.

Característica	Variantes
Arquitetura	Monolítico
	Modular / Componentes
	Máquina Virtual
Modelo de Execução	Eventos
	Threads
	Híbrido
Reprogramação	Aplicação
	Módulo / Componente
	Instrução
	Variável
Agendamento	Tempo-Real
	Não Tempo-Real
Gestão de Energia	
Outros	Simulação
	Portabilidade

Tabela 2 - Quadro de classificação para os SO usados em RSSF [48]

Existe uma relação direta entre o desempenho e a flexibilidade, dependendo da arquitetura utilizada. Com o uso de *kernels* monolíticos é criada uma imagem única para o nó logo, este tipo de arquitetura não é adequado para aplicações que mudam frequentemente de requisitos. *Kernels* modulares são mais adequados a cenários onde os requisitos possam mudar frequentemente, simplifica os problemas de manutenção e modificação de código mas traz uma sobrecarga ao sistema pelo facto de carregar/descarregar módulos. Esta sobrecarga também inclui a necessidade de alocar memória para o novo módulo. As máquinas virtuais podem ver os nós da rede como um todo ou em separado. De qualquer forma, as máquinas virtuais trazem flexibilidade ao desenvolvimento das aplicações, sendo estas compostas por instruções específicas para a máquina virtual, tornando a reprogramação um processo fácil.

- **Sistemas Operativos baseados em Eventos**

O TinyOS [44] é um dos SO mais conhecidos para uso em RSSF. É baseado em eventos e fornece uma *framework* de programação para sistemas embebidos. Tem um modelo de execução baseado em componentes, implementado na linguagem de programação nesC [45]. O modelo de concorrência do TinyOS baseia-se em comandos, eventos assíncronos, tarefas e interfaces divididas em fases (*Split-Phase*). A invocação de funções (*Command*) e sua execução (*Events*) são separadas em duas fases nas interfaces fornecidas pelo TinyOS. As aplicações devem ser programadas com os *handlers* apropriados para cada evento. Os *handlers* de eventos e comandos podem criar uma tarefa que será colocada no escalonador de tarefas "*First In, First Out*" (FIFO) disponível no TinyOS. As tarefas podem ser precedidas por eventos mas não por outras tarefas. As questões de *"Data Race"* que possam surgir podem ser controladas usando secções atómicas. A arquitetura de comunicação do TinyOS usa o conceito de *"Active Messages"*. Este conceito baseia-se no uso de pequenos pacotes com cerca de 36 bytes e um *handler* ID de um byte. Um nó após receber uma *"Active Message"*, encaminha-a para o *handler* correspondente. O modelo baseado em eventos do TinyOS tem algumas desvantagens, como por exemplo, a pouca flexibilidade do código [48] e uma curva de aprendizagem elevada.

Outro exemplo de SO baseado em eventos é o "C# Open Source Operating System" (CORMOS) [49]. As abstrações básicas que o CORMOS fornece são eventos e *handlers*. Os eventos podem ser ações locais ou remotas. Os eventos são explicitamente criados pelos *handlers*, sendo estes invocados quando são escalonados. Usa um escalonador do tipo "*Earliest-Deadline-First"* que facilita o escalonamento de eventos por parte das aplicações e

esconde das aplicações a existência de um temporizador baseado em interfaces assíncronas [48].

- **Sistemas Operativos baseados em Threads**

O MantisOS [50] usa um modelo de execução baseado em *threads.* Desta forma oferece uma maior flexibilidade no desenvolvimento das aplicações, uma vez que o programador não está preocupado com o tamanho da tarefa, algo que é essencial em modelos baseados em eventos. A pilha de rede e o escalonador são também implementados usando *threads*, tal como uma aplicação. À parte destas *threads*, existe uma *"idle-thread"* que corre enquanto todas as outras estão bloqueadas, invocando as rotinas de gestão de consumo energético necessárias. Para manter as *threads*, o *kernel* mantém uma tabela que consiste na prioridade de cada *thread* e outras informações acerca das *threads*. O agendamento entre *threads* é feito através de um algoritmo que segue a semântica *Round-Robin*. Os problemas de *"Data Race"* são evitados usando semáforos. O MantisOS apresenta um consumo de memória RAM acima da média devido à mudança de contexto e segmentos da pilha atribuídos a cada *thread*. Esta sobrecarga pode ser limitativa em sistemas com recursos limitados [48].

- **Sistemas Operativos Híbridos**

O ContikiOS [51] é um exemplo deste tipo de SO. Combina as vantagens dos modelos baseados em eventos e dos modelos baseados em *threads*. É maioritariamente um modelo orientado a eventos, mas suporta *multi-threading* com uma biblioteca opcional ao nível da aplicação. A aplicação pode associar-se a esta biblioteca se necessitar de *multi-threading.* Os eventos são classificados como assíncronos e síncronos. Os eventos síncronos são executados imediatamente, enquanto os eventos assíncronos são executados depois. Tudo é implementado como sendo um serviço, sejam comunicações, controladores de dispositivos ou tratamento de dados sensoriais. Cada um destes serviços tem uma interface e uma implementação. A aplicação conhece apenas as interfaces. A implementação do serviço pode ser alterada durante a execução [48].

- **Outros tipos de Sistema Operativo**

O SenOS [52] é baseado numa máquina de estados finitos. O *kernel* do SenOS escolhe um evento da fila e toma as ações correspondentes que resultarão em mudanças de estado. As bibliotecas de *callback* fornecem ao programador acesso ao *hardware*, comunicações e

tratamento de eventos. Cada tabela de transição define uma aplicação em execução na RSSF. A concorrência entre as aplicações é gerida por comutação das tabelas de transição para execução [48].

O Nano-RK [53] é um SO em tempo-real com escalonador preemptivo. As unidades computacionais e tarefas têm uma prioridade associada e as tarefas de alta prioridade são sempre executadas antes das outras tarefas. As aplicações podem definir os requisitos de recursos e prazo necessários. O Nano-RK é uma extensão do paradigma "*Resource-Kernel"* (RK) para uso em dispositivos com limitações energéticas, como os usados em RSSF.

2.2.4.2 Algoritmos Distribuídos

As RSSF são compostas por um grande número de nós, deste modo os algoritmos usados são considerados algoritmos distribuídos. O recurso mais escasso nas RSSF é a energia e uma das operações mais caras é a transmissão de dados e a escuta ativa. Por esta razão, a investigação em RSSF concentra-se no estudo de algoritmos para poupança de energia reduzindo a quantidade de informação a ser transmitida, usando técnicas como agregação de dados, mudando a potência de transmissão dos nós ou mesmo desligando os nós preservando a conectividade. Os algoritmos de modelagem, simulação e análise de RSSF diferenciam-se dos protocolos de abordagem, uma vez que os modelos matemáticos (idealizados) são mais abstratos, mais gerais e mais fáceis de analisar. No entanto, são por vezes menos realistas do que os modelos utilizados para a conceção do protocolo, uma vez que uma abordagem algorítmica muitas vezes ignora as questões de tempo e sobrecarga do protocolo [48].

2.2.5 Arquitetura

Existem três fatores determinantes na estruturação de uma RSSF, sendo eles:

- **Infraestrutura:** Consiste nos componentes físicos que constituem a RSSF, características internas dos dispositivos, tais como a autonomia, capacidade de memória/processamento e o modo/local de instalação;
- **Pilha Protocolar:** Refere-se às camadas protocolares existentes em cada nó;
- **Aplicação:** Evidencia os objetivos e funções da rede. Na fase inicial das RSSF os nós-sensor eram muito simples, apenas com a função de recolher dados do meio envolvente e enviar a informação recolhida através da rede para mais tarde ser

processada por algo ou alguém.

Com a evolução tecnológica, foi possível passar a usar com mais facilidade microcontroladores de maior capacidade nos nós sensoriais, fazendo com que a informação recolhida possa ser tratada no próprio nó-sensor antes de ser enviada para a rede [54].

2.2.5.1 Infraestrutura

A infraestrutura de uma RSSF define as características dos nós-sensor que a constituem, tais como, a localização, memória, sensores a que estão acoplados, etc. Todas as RSSF possuem pelo menos um nó-sensor, o nó de saída (*Gateway*). Os denominados nós-sensor têm a função de reunir e transmitir a informação recolhida do meio que monitorizam [55].

Sendo os dispositivos de aquisição sensorial usados em RSSF equipamento tipicamente de recursos limitados, os dados recolhidos são enviados para um nó-base (Gateway) com maiores capacidades onde serão analisados mais profundamente ou então reenviados pelo nó-base para uma aplicação central a ser executada num PC, por exemplo. Esta aplicação encontra-se numa rede externa à rede de sensores.

A importância dos nós de saída (*Gateway*) é elevada visto que são responsáveis por fazer a ligação entre a rede de sensores e a rede externa. Dependendo dos objetivos, esta comunicação pode ser unidirecional ou bidirecional. O nó de saída coleciona a informação recolhida na rede, podendo ainda alterar parâmetros desta em tempo real. Os dados recolhidos pelos nós são apresentados por intermédio de um computador ou outro dispositivo elétrico. Visto que o nó de saída receberá e enviará muita informação, faz sentido que esteja ligado a uma fonte de energia permanente, sem recorrer a baterias. A sensibilidade e intensidade do sinal são fatores importantes. Podem ser definidos os modos de funcionamento do módulo de rádio frequência como transmissão, receção e *sleep* (*idle*).

A alimentação dos nós-sensor é garantida recorrendo a baterias e a duração das mesmas depende de diversos fatores, como por exemplo, o tamanho e o material usado. Cada nó sensor pode ser composto por um número limitado de sensores, devido às limitações físicas de cada dispositivo. Como consequência da limitação física e energética, a capacidade de processamento dos nós é limitada, sendo necessário garantir que as operações realizadas sejam estritamente as necessárias de maneira a que a durabilidade das baterias seja a maior possível [56].

2.2.5.2 Pilha-Protocolar

A pilha protocolar usada nos nós-sensor é constituída pelas tradicionais cinco camadas horizontais (Física, Ligação, Rede, Transporte e Aplicação) e por três camadas verticais (Gestão de Energia, Gestão de Conectividade e Gestão de Tarefas). Nas aplicações para RSSF, a maioria dos protocolos exploram a colaboração entre os nós com a finalidade de obter a melhor eficiência de recursos, mais concretamente ao nível de processamento e de poupança de energia. Recorrendo a uma estrutura em camadas consegue-se obter um melhor controlo, flexibilidade e desempenho. Numa arquitetura em camadas tradicional, uma determinada camada só comunica com as camadas que lhe são adjacentes.

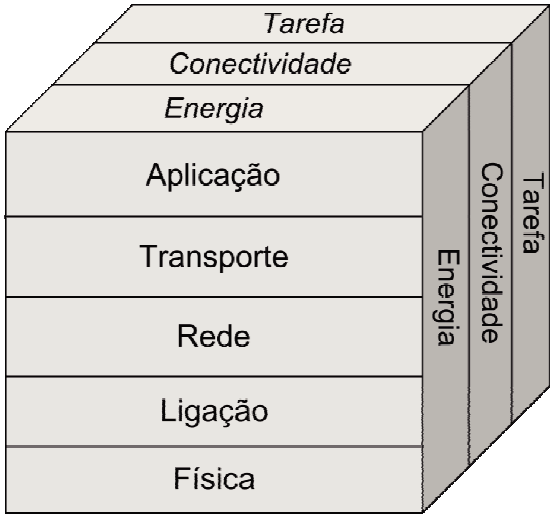

Figura 7 - Pilha Protocolar [55]

2.2.5.3 Aplicação

Uma vez que a aplicação é parte integrante da RSSF, esta define também a arquitetura da rede. Como referido anteriormente, as aplicações possíveis para uma RSSF são diversas. O tipo de aplicação pretendido deve influenciar o desenho e construção da rede em diversas características, como por exemplo a topologia de rede a usar ou o equipamento necessário para que a aplicação atinja o fim a que se propõe.

2.2.6 Standards

Para facilitar o desenvolvimento de aplicação para RSSF, existe a necessidade de criar um mercado de nós-sensor de baixo custo. Para este efeito, é importante especificar normas que permitam aos diferentes nós-sensor interagir. Tem sido feito um elevado esforço de maneira a

unificar o mercado, levando à criação de dispositivos de baixo custo que comuniquem entre si, evitando a proliferação de dispositivos proprietários que usam protocolos de rede incompatíveis. Em certa medida, o sucesso das RSSF em muito depende do sucesso destes *standards.*

Com o desenvolvimento crescente das RSSF o aparecimento de *standards* e especificações tornou-se uma necessidade. Existem atualmente diversos padrões para o desenvolvimento de RSSF, entre eles o IEEE 802.154, IEEE 1451, "*IPv6 over Low power Wireless Personal Area Networks"* (6LoWPAN), *"Highway Addressable Remote Transducer Protocol"* (WirelessHART), *"International Society of Automation"* (ISA) 100 e ZigBee apresentados em detalhe seguidamente.

2.2.6.1 IEEE 802.15.4

O IEEE 802.15.4 [57] é uma norma desenvolvida pelo Grupo de Trabalho IEEE 802.15.4 e especifica a camada física e de acesso ao meio para WPAN (Wireless Personal Area Network) de baixo débito. Como definido na solicitação de autorização do projeto, o objetivo do Grupo de Trabalho 4 é fornecer uma norma para dispositivos de baixa complexidade, baixo consumo, baixo debito e baixo custo. A primeira versão do IEEE 802.15.4 foi realizada em 2003 [58], sendo revista em 2006. A pilha protocolar do IEEE 802.15.4 é simples e flexível, incluindo as seguintes características:

- Débitos de 250 kbps, 40 kbps e 20 kbps;
- Dois modos de endereçamento, 16-bit e 64-bit IEEE;
- Suporte para dispositivos onde a latência é um fator crítico, por exemplo, *joysticks;*
- Acesso ao canal baseado em *"Carrier Sense Multiple Access with Collision Avoidance"* (CSMA-CA);
- Estabelecimento de rede de forma automática, através do nó coordenador;
- Protocolo de *"hand-shaking"* para fiabilidade na transferência de dados;
- Gestão de energia para garantir um baixo consumo;

- 16 canais na banda *"Industrial, Scientific and Medical"* (ISM) 2,4 GHz, 10 canais nos 915 MHz e 1 canal nos 868MHz.

A camada física do IEEE 802.15.4 foi especificada de forma a coexistir com outros padrões IEEE para redes sem fio, como por exemplo, o IEEE 802.11 (WLAN) ou o IEEE 802.15.1 (Bluetooth).

A norma 802.15.4 pode operar numa das seguintes bandas livres:

- 868-868,6 MHz (Europa), com um débito até 20 kbps;
- 902-928 MHz (América do Norte), com um débito até 40 kbps;
- 2400 - 2483,5 MHz (Mundial) com um débito até 250 kbps.

A camada de acesso ao meio (MAC) oferece serviços de dados e de gestão às camadas superiores. O serviço de dados permite a transmissão e a receção de pacotes MAC sobre a camada física. Os serviços de gestão incluem sincronização, gestão de tempo e associação e dissociação de dispositivos à rede. A camada MAC implementa também mecanismos básicos de segurança.

As camadas superiores não são abrangidas pela norma, podendo ser usado "em cima" da norma 802.15.4 um número variado de protocolos, como por exemplo, o ZigBee, WirelessHART, ISA SP100 ou 6LoWPAN.

Como usa a técnica de modulação "*Direct Sequence Spread Spectrum"* (DSSS) torna-se bastante resistente ao ruído. Basicamente, cada bit de informação a ser transmitido é modulado em quatro sinais diferentes. Este processo faz com que o total de informação a ser transmitida ocupe uma maior largura de banda.

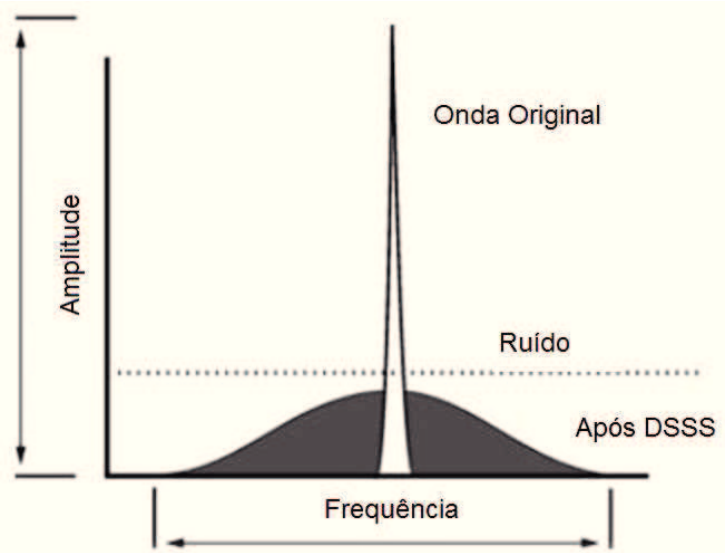

Figura 8 - Modulação DSSS [59]

Existem menos interferências nas faixas de frequências utilizadas e melhora a relação "*Signal to Noise Ratio"* (SNR) no recetor devido ao facto de ser mais fácil detetar e descodificar a mensagem que está a ser enviada pelo transmissor [59].

Na Figura 9 é a apresentada a estrutura de uma *frame* IEEE 802.15.4.

Cabeçalho 802.15.4 MAC					**802.15.4 MAC Payload**	**802.15.4 MAC Footer**
Controlo Frame	Número Sequência	Endereço Destino	Endereço Origem	Cabeçalho Segurança	Payload	Controlo Erros

Figura 9 - Frame IEEE 802.15.4 [60]

A norma IEEE 802.15.4 utiliza duas técnicas para evitar que todos os nós iniciem a transmissão ao mesmo tempo. A mais comum é a CSMA-CA. Esta técnica é descrita da seguinte forma:

- Cada nó escuta o meio antes de transmitir. Se a energia detetada for maior que um determinado nível o nó transmissor espera durante um tempo aleatório (incluindo um intervalo) e tenta novamente.

A energia é detetada através da funcionalidade *Channel Energy Scan* (PLME-ED Request) que tem a capacidade de saber a energia (ruído, atividade e interferências) existente nos

vários canais, antes de começar a usá-lo. A segunda técnica é *"Guarantee Time Slots"* (GTS). Este sistema usa um nó central (Coordenador da *Personal Area Network*) que dá *slots* de tempo a cada nó-sensor de forma a que os vários nós-sensor da rede saibam quando devem transmitir. Como primeiro passo, o nó-sensor que pretende transmitir deve enviar ao coordenador uma mensagem de solicitação GTS. A resposta do Coordenador irá conter a informação acerca do *slot* alocado para a transmissão pretendida [59].

2.2.6.2 IEEE 1451

Os padrões IEEE 1451 são uma família de padrões de interface de transdutores inteligentes. Definem um conjunto de interfaces de comunicação aberto e comum, de maneira a interligar transdutores (sensores ou atuadores) [61].

Os transdutores têm uma grande variedade de aplicações na indústria, por exemplo, na fabricação, no controle industrial, engenharia aeroespacial, construção civil e biomedicina. O elevado número de redes sem fios e com fios disponível no mercado, é um dos maiores problemas que os fabricantes de transdutores enfrentam. De maneira a colmatar este tipo de problemas, foi desenvolvido um conjunto de padrões abertos, universalmente aceites, como por exemplo, o conjunto de padrões IEEE 1451. Em resumo, o principal objetivo do IEEE 1451 é facilitar o desenvolvimento de transdutores, permitindo o acesso aos dados através de um conjunto de interfaces comum.

A família de padrões IEEE 1451 é patrocinada pelo "Sensor Technology Technical Committee of the IEEE Instrumentation and Measurement Society". As definições do IEEE 1451 [61] são apresentadas resumidamente de seguida:

- **IEEE P1451.0:** Define um conjunto de comandos comuns, operações comuns e "*Transducer Electronic Data Sheets"* (TEDS) para a família IEEE 1451. Através deste conjunto de comandos, pode-se aceder a qualquer nó-sensor ou atuador da rede baseada em IEEE 1451;

- **IEEE P1451.1:** Define um objeto de modelação comum que descreve o comportamento dos transdutores. Define também um modelo de medição que agiliza a análise de processos e de modelos de comunicação utilizados, incluído os modelos cliente/servidor e publicar/subscrever;

- **IEEE P1451.2:** Define uma interface Transdutor-NCAP (*Network Capable*

Application Processor) e as TEDS para configurações ponto-a-ponto;

- **IEEE P1451.3:** Define uma interface Transdutor-NCAP (*Network Capable Application Processor*) e as TEDS para transdutores multiponto utilizando uma arquitetura de comunicação distribuída. Permite que vários transdutores sejam dispostos numa rede multiponto, partilhando um par de fios comum entre si;

- **IEEE P1451.4:** Define uma interface de modo misto para transdutores com modo de operação analógico e digital;

- **IEEE P1451.5:** Define uma interface Transdutor-NCAP (*Network Capable Application Processor*) e as TEDS para transdutores sem fio. Protocolos comuns, como por exemplo, 802.11 (Wi-Fi), 802.15.1 (Bluetooth) e 802.15.4 (ZigBee), estão a ser considerados como algumas das interfaces físicas para IEEE P1451.5;

- **IEEE P1451.6:** Define uma interface Transdutor-NCAP (*Network Capable Application Processor*) e as TEDS para interfaces de rede de alta velocidade CANopen. Aplicações intrinsecamente seguras e não intrinsecamente seguras são suportadas.

2.2.6.3 6LoWPAN

O grupo de trabalho 6lowPAN tem vindo a propor o uso de soluções IPv6 em RSSF. O uso de IP em RSSF não tem sido só aceite como uma extensão das redes convencionais, mas também como uma solução para o uso de protocolos bem conhecidos em RSSF, tais como o UDP ou HTTP. Define mecanismos que permitem a compressão do cabeçalho IPv6 permitindo o envio e receção de pacotes em redes baseadas em IEEE 802.15.4 [62]. Os principais objetivos do 6lowPAN são [63]:

- Adaptação do pacote IP e interoperabilidade;

- Esquemas de endereçamento e gestão de endereços;

- Gestão de rede;

- Encaminhamento adaptativo em topologias dinâmicas;

- Segurança, incluindo *Set-Up* e manutenção;

- Interface de Programação de Aplicação (API);
- Descoberta de dispositivos e serviços.

A aplicação da tecnologia IP nas RSSF traz vários benefícios:

- Dado o uso massivo de redes IP, podem ser usadas as infraestruturas já existentes nas RSSF;
- As tecnologias baseadas em IP são bem conhecidas e o seu funcionamento está comprovado;
- As especificações da tecnologia de rede IP são disponibilizadas em formato aberto e livre, o que é mais favorável para o desenvolvimento, comparando com soluções proprietárias;
- Existem diversas ferramentas para diagnóstico e gestão de redes IP;
- Dispositivos baseados em IP podem ser facilmente ligados sem a necessidade de entidades intermediárias, como por exemplo um *proxy*.

Tanto a norma IEEE 802.15.4 bem como a especificação do formato 6LoWPAN não definem as topologias de rede usadas. Deste modo, devem ser consideradas capacidades de *Routing Multi-Hop*. Existem algumas implementações 6lowPAN desenvolvidas para o SO TinyOS (6LowPANcli e b6LowPAN), existindo também uma implementação para o SO *ContikiOS* (SicsLowPAN) [63].

2.2.6.4 WirelessHART

WirelessHART é a versão *wireless* do protocolo HART, usado na automação industrial e em aplicações que requerem funcionalidades em tempo real. As suas principais capacidades são as seguintes [59]:

- Confiável, mesmo na presença de interferências graças à topologia em *Mesh*, ao salto de canal *"channel hopping"* e à sincronização de mensagens. A coexistência de WirelessHART com outras redes sem fio não é problemática;
- Segurança e privacidade nas comunicações através de criptografia, autenticação, gestão de chaves e de outras boas práticas disponíveis em padrões abertos;

- Gestão de energia eficaz através de técnicas que tornam prático o uso de pilhas, energia solar e outras opções de baixa potência;

- Usa "*Time Synchronized Mesh Protocol"* (TSMP). É necessário, um nó coordenador de modo a que seja atribuído um *"time slot"* a todos os nós da rede.

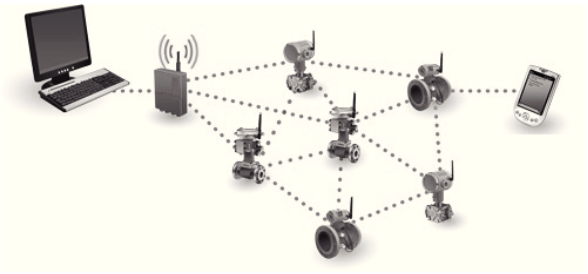

Figura 10 - Funcionamento de uma rede WirelessHART [59]

A International Electrotechnical Commission (IEC) aprovou a especificação WirelessHART como uma norma internacional (IEC 62591Ed. 1.0). A 26 de Março de 2010, o Comité Nacional da IEC, composto por 28 países, confirmou o apoio global para a tecnologia WirelessHART como norma internacional para comunicação sem fios em automação de processos [59].

2.2.6.5 ISA100

A norma ISA100 foi criada em 2005 e aprovada em 2009. Pretende fornecer comunicações sem fios fiáveis e seguras em diversas áreas como, por exemplo, nas áreas da monitorização não crítica, sistemas de alerta e sistemas de supervisão. Esta norma define um conjunto de protocolos e especificações de segurança para comunicações sem fios em rede de baixa largura de banda, com recursos e consumos de energia muito limitados. O objetivo principal é fornecer os requisitos de desempenho para aplicações de monitorização e controlo de acesso, onde latências na ordem de 100ms podem ser toleradas, adotando um comportamento opcional para menores latências. Para responder às necessidades dos utilizadores e operadores, a norma ISA100 leva em conta a presença de interferências normalmente encontradas em ambientes industriais agressivos. A coexistência da norma ISA100 num ambiente industrial com outros dispositivos sem fios, como por exemplo telemóveis e dispositivos baseados em IEEE 802.11, IEEE 802.15, IEEE 802.16 e outros, é também levada em conta [64].

2.2.6.6 ZigBee

A norma IEEE 802.15.4 apenas define as camadas físicas e de acesso ao meio, sem especificar as camadas protocolares superiores, incluindo as camadas de rede e de aplicação. A norma ZigBee [65] é desenvolvida sobre a norma IEEE 802.15.4 e define as camadas de rede e de aplicação. A camada de rede fornece funcionalidades de rede para diferentes topologias de rede e a camada aplicacional fornece uma estrutura para comunicação e desenvolvimento de aplicações. A primeira versão da pilha ZigBee foi revista no final de 2006, introduzindo extensões na padronização de perfis de aplicação e algumas melhorias nas camadas de rede e aplicação.

É destinada a usos nas áreas dos sensores embutidos, aquisição de dados médicos, dispositivos de consumo e domótica. O nome ZigBee é inspirado no comportamento das abelhas. Tem como principal objetivo criar uma topologia de rede hierárquica que permita a uma série de dispositivos comunicar entre eles definindo recursos de comunicação extra, como por exemplo, autenticação, criptografia, associação e serviços de aplicação [66].

O ZigBee foi criado por um conjunto de empresas que formam a ZigBee Alliance, da qual fazem parte empresas como a FreeScale, Philips, Texas Instruments, AT&T, ATMEL, Cisco, Intel, Samsung entre outras [67].

<table>
<tr><th>Modelo OSI</th><th>IEEE 802.15.4 / ZigBee</th></tr>
<tr><td>Aplicação</td><td rowspan="2">Objetos de Aplicação ZigBee
Serviços de Segurança ZigBee</td></tr>
<tr><td>Transporte</td></tr>
<tr><td>Rede</td><td>Encaminhamento ZigBee (AODV)</td></tr>
<tr><td>Ligação de Dados</td><td>802.11 LLC</td></tr>
<tr><td>Acesso ao Meio</td><td>802.15.4 MAC</td></tr>
<tr><td>Física</td><td>868MHz / 915 MHz / 2.4GHz</td></tr>
</table>

Tabela 3 - Relação entre o modelo OSI e ZigBee [66]

A norma ZigBee tem aplicações nas mais variadas áreas, como por exemplo:

- **Entretenimento e Controlo Doméstico:** Iluminação inteligente, controlo avançado de temperatura, segurança, filmes e música;

- **Sensibilização Ambiental:** Sensores de água e energia, monitorização do consumo energético, detetores de fumo, aparelhos inteligentes e sensores de acesso;
- **Serviços Móveis:** M-Pagamento, M-Vigilância e Controlo, M-Segurança e Controlo de Acesso, M-Saúde e Teleassistência;
- **Infraestruturas Comerciais:** Monitorização do consumo energético, Climatização, Iluminação, Controlo de Acesso;
- **Indústria:** Controlo de Processos, Gestão de Ativos, Gestão Ambiental, Gestão Energética, Controlo de Equipamento Industrial, Comunicações Máquina-Máquina.

Analisando mais detalhadamente os serviços oferecidos pelo ZigBee [66]:

- **Serviços de Cifragem Extra:** Implementa Criptografia *"Advanced Encryption Standard"* (AES) 128bit em chaves de rede e de aplicação;
- **Associação e Autenticação:** Apenas nós-sensor válidos poderão aceder à rede;
- **Protocolo de Encaminhamento:** O protocolo de encaminhamento Ad-Hoc Reativo *"Ad-Hoc On-Demand Distance Vector"* (AODV) [68];
- **Serviços de Aplicação:** É usado o conceito abstrato de "Cluster". Cada nó pertence a um conjunto pré-definido e pode ter um número predefinido de ações.

Existem três tipos de dispositivos numa rede ZigBee, sendo eles [66]:

- **ZigBee Coordinator (ZC):** É normalmente o dispositivo com maiores capacidades. Cria a árvore da rede (nó pai) e poderá ligar a sua rede a outras redes. Existe apenas um coordenador ZigBee em cada rede. É capaz de armazenar informações sobre a rede, atuando como o "*Centro de Confiança"* e repositório de chaves de segurança;
- **ZigBee End Device (ZED):** Contém apenas as funcionalidades necessárias para falar com o nó pai (*Coordinator* ou Router). Não pode transmitir dados recebidos de outros nós, o que permite o uso do modo *sleep* durante uma quantidade significativa de tempo, otimizando assim o tempo de vida da bateria. Requerem menos capacidade de memória, podendo assim ser menos dispendioso do que um *Coordinator* ou um Router;

- **ZigBee Router (ZR):** Encaminha a informação enviada pelos ZigBee *End-Devices*.

Seguidamente são apresentadas algumas das regras básicas de operação:

- O nó-sensorial (End Device) liga-se a um nó-*Router* ou a um nó-*Coordinator*;
- Os nó-*Router* podem ligar-se entre si e com o nó-*Coordinator*;
- Os nó-*Router* e *Coordinator* não podem estar em modo *sleep* e devem guardar nos seus *buffers* os pacotes que são enviados para os nós-sensor;
- Os nós-sensor *End-Device* podem estar em modo *sleep*;
- Os nó-*Coordinator* e nó-*Router* não podem ser alimentados por baterias.

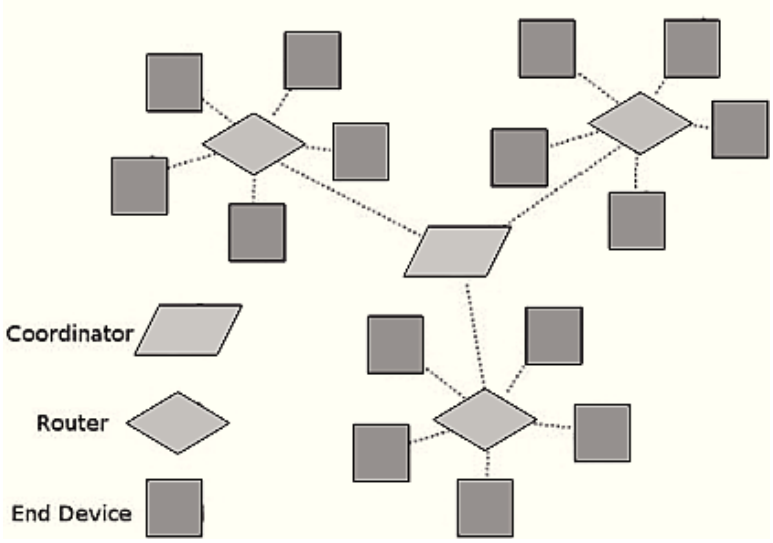

Figura 11 - Topologia ZigBee [66]

Como se pode ver na Figura 11, o ZigBee cria topologias de rede em Estrela (*Star*) e não em Malha (*Mesh*).

O ZigBee é uma "*camada"* pensada para organizar a rede. A primeira coisa que um nó faz quando se quer associar à rede é pedir ao *Coordinator* um endereço de rede (16-bit). Toda a informação na rede é encaminhada através desse endereço. No processo de associação são realizados os procedimentos de autenticação e cifragem. Depois de associado à rede, o nó pode trocar informação com os outros nós através de nó-*Router* que estão sempre à espera de receber informação. Quando o nó-*Router* recebe um pacote e o destino está ao seu alcance,

verifica se o nó destino está em modo *sleep*. Se não estiver, o nó-*Router* envia o pacote para o dispositivo final. Se estiver em modo *sleep*, o nó-*Router* colocará o pacote num *buffer* e quando o nó de destino sair do modo *sleep* e perguntar por novos pacotes ao nó-*Router*, o pacote será então enviado [66]. A especificação ZigBee está disponível gratuitamente para o público em geral, caso se trate de projetos sem fins comerciais.

2.3 Síntese

Neste capítulo foi efetuado um levantamento do conhecimento científico na área das Redes Sensoriais Sem Fios, na qual se insere este trabalho. Pretendeu-se dar a conhecer quais os desafios em causa e possíveis soluções.

Na primeira parte do capítulo foi apresentada uma visão geral das características e aplicações das RSSF. Posteriormente, este capítulo centrou-se na evolução tecnológica do *hardware* e *software* tipicamente usado em RSSF e nas suas particularidades arquiteturais. Por último, foram apresentados os principais *standards* usados no desenvolvimento de aplicações para RSSF.

No capítulo seguinte é apresentada a solução proposta que tem como objetivo definir uma especificação de uma arquitetura genérica com o intuito de facilitar a estruturação de uma RSSF.

Esta página foi intencionalmente deixada em branco

3 Solução Proposta

Neste terceiro capítulo são apresentados os requisitos da solução e qual a solução proposta para a implementação de uma arquitetura que tenta ser genérica e escalável.

Inicialmente serão definidos os requisitos da solução, apresentando o modelo conceptual da solução proposta e a avaliação às diferentes topologias tipicamente usadas em Redes Sensoriais Sem Fios (RSSF), identificando assim qual a que melhor responde aos requisitos apresentados. Posteriormente, é apresentado um conjunto de técnicas que definem as técnicas de *clustering* e a especificação da arquitetura proposta. Nesta última secção são identificadas as funcionalidades e os requisitos das várias entidades que constituem a arquitetura, nomeadamente, a pilha-protocolar de cada entidade definida.

Por último é apresentada a análise da solução proposta.

3.1 Requisitos da Solução

Uma das características mais importantes no desenho de aplicações para RSSF é a topologia de rede usada. A topologia tem elevado impacto em vários constrangimentos tipicamente encontrados em RSSF, como o consumo de energia, a latência, baixo poder computacional e a qualidade da comunicação. Por natureza, a topologia utilizada define o tipo de rotas, o uso de comunicação *broadcast* ou *unicast*, o tamanho dos pacotes, o *overhead*, entre outras particularidades.

A escolha da topologia adequada para uma dada aplicação reduzirá a necessidade de comunicação acabando por otimizar o consumo de energia de toda a RSSF. A escolha de uma dada topologia poderá também facilitar a agregação de dados, o que reduz a necessidade de processamento e comunicação, aumentado consequentemente o tempo de vida da RSSF. A topologia define ainda o tamanho de um dado grupo de nós e como gerir a adição/remoção de nós.

O uso de uma topologia lógica facilitará a RSSF em vários modos, maximizando o tempo de vida da rede, reduzindo interferências e tornando a rede escalável. Desta forma, é crucial comparar as diferentes topologias e definir a que melhor se adapta à arquitetura que se pretende implementar. Cada uma das topologias tem as suas vantagens e desvantagens quando usadas sobre um dado ambiente. É por isso necessário especificar o modelo conceptual da RSSF que se pretende implementar e também, definir um conjunto de métricas para avaliação.

Seguidamente é apresentado o modelo conceptual e seus pressupostos.

3.1.1 Modelo Conceptual

Visto que as RSSF são em muito dependentes da aplicação, é importante parametrizar as características e objetivos da aplicação que se pretende implementar. Existem diferentes classificações para as aplicações de RSSF. Uma das possíveis classificações distingue as aplicações de acordo com o tipo de dados recolhidos pela rede. Em geral, todas as aplicações podem ser definidas como baseadas em Deteção de Eventos (DE) ou Análise Temporária de um Espaço (ATE). No caso das aplicações baseadas em DE, os vários nós-sensor têm como objetivo detetar um dado evento, como por exemplo um terramoto. No caso das aplicações baseadas em ATE, os vários nós-sensor têm como objetivo recolher periodicamente dados acerca de uma determinada variável presente num dado espaço, como por exemplo a temperatura de um dado local.

Para que se encontre a topologia adequada à aplicação pretendida devemos então definir o modelo conceptual, apresentando as características que serão usadas como critério para a escolha da topologia.

Tal como referido anteriormente, as RSSF são bastante dependentes da aplicação e do modelo conceptual. É portanto importante definir o modelo conceptual antes de apresentar qualquer algoritmo/protocolo/arquitetura.

O modelo conceptual é baseado nos seguintes pontos:

- A escala da rede não será elevada, não ultrapassando as dezenas/centenas de nós. Contudo, a adição de novos nós à RSSF deverá ser simples e transparente;
- A ligação entre a RSSF e outras redes será feita através de uma ou mais estações

base (*gateways*);

- As estações base serão estacionárias após a instalação;
- Os nós poderão ou não ser estacionários;
- Os nós poderão ou não ser alimentados por baterias. Devendo assim serem usadas técnicas para poupança de energia. Contudo, a substituição de baterias é possível;
- Os nós são heterogéneos, apresentando diferentes capacidades de processamento e de energia;
- Todos os nós têm limite de distância de transmissão, tendo a capacidade para controlar o poder de transmissão dependendo da distância entre si e o próximo salto;
- O canal de transmissão rádio é simétrico, ou seja, a energia necessária para transmitir de A para B é a mesma que de B para A;
- Os nós-sensor presentes na RSSF podem ter diferentes funções, podendo estes usar aplicações baseadas em DE ou ATE, variando assim o seu *Duty-Cycle*;
- A aplicação é sensível a atrasos, sendo que a latência deve ser mínima.

Após a apresentação do modelo conceptual que define as características da RSSF pretendida, serão agora apresentadas as métricas relevantes para a avaliação das diferentes topologias.

3.1.2 Métricas de Avaliação

Seguidamente são apresentadas as várias métricas de desempenho, comparando o desempenho esperado das várias topologias para cada uma dessas métricas. As métricas definidas baseiam-se nas principais particularidades das RSSF, nomeadamente, eficiência energética, fiabilidade, escalabilidade e latência. A avaliação realizada tendo por base este conjunto de métricas é apresentada de seguida.

3.1.2.1 Consumo Energético da RSFF

Dadas as limitações das RSSF e o típico uso de baterias para a alimentação dos vários nós, o consumo energético é uma métrica de avaliação comum. Algumas das métricas relacionadas

com o consumo energético são apresentadas de seguida:

- Dissipação energética por cada nó da RSSF. Estimativa linear do consumo geral da RSSF, dando uma ideia de qual será o tempo de vida da RSSF;
- Distribuição da carga energética. É importante balancear o consumo exigido pela transmissão de mensagens pelos vários nós da RSSF. Caso essa exigência recaia sempre sobre os mesmos nós, a energia não estará dividida equitativamente e acabará por afetar o tempo de vida de toda a RSSF;
- Média de dissipação energética por cada nó. Este valor é a relação entre o consumo de cada nó e o número de eventos detetados ou necessidade de transmissão.

3.1.2.2 Tempo de vida da RSSF

A métrica relacionada com o tempo de vida da RSSF está dependente da capacidade energética dos nós. Esta é uma métrica importante para as RSSF que não permitem a substituição de baterias. Algumas das métricas relacionadas com o tempo de vida da RSSF são apresentadas de seguida:

- Tempo até à falha do primeiro nó;
- Tempo até que uma certa percentagem de nós falhe, ou que o número de nós ainda em atividade ultrapasse um determinado valor;
- Tempo até que a entrega de pacotes (com sucesso / sem sucesso) caia abaixo de um determinado valor;
- Tempo até que todos os nós falhem;
- Tempo até que um determinado número de erros seja atingido.

3.1.2.3 Escalabilidade da RSFF

A escalabilidade é um fator importante no desenho de protocolos para RSSF. Os protocolos precisam de ser escaláveis e adaptáveis à mudança provocada por falha ou adição de nós. Para limitar o *overhead* das comunicações, são necessários protocolos de encaminhamento eficientes. Para RSSF de média/larga escala, a escalabilidade é uma métrica importante na avaliação do comportamento do protocolo em relação à densidade de nós, tamanho da rede

ou número de fontes de dados e pontos de saída da RSSF.

3.1.2.4 Overhead e Eficiência da RSSF

Em conjunto com as métricas de consumo energético, o *overhead* e eficiência da rede são outras das métricas que devem ser usadas. Algumas das possíveis métricas relacionadas com o *overhead* e eficiência da RSSF são apresentadas de seguida:

- Custo do envio de uma mensagem. Esta é uma métrica comum na avaliação da eficiência de um dado protocolo. Avalia o número de pacotes gerados pelo algoritmo/protocolo para cada comunicação com sucesso;
- Perda de mensagens. Avalia a percentagem de mensagens não recebidas pelos nós da RSSF;
- *Overhead* provocado por mensagens de controlo. Avalia a relação entre as mensagens de dados úteis e as mensagens de controlo transmitidas na RSSF;
- Média da distância das rotas usadas. Avalia o número de saltos desde a origem até ao destino. Apesar de estar relacionado com o consumo energético, determinada rota pode provocar resultados completamente diferentes, devido à relação não-linear entre o poder de transmissão e o alcance.

3.1.2.5 Latência e Tempo de Resposta da RSSF

A latência é avaliada tendo em conta a média de atraso entre a transmissão de uma mensagem pelo nó origem até à sua chegada à estação de controlo. O tempo de resposta avalia a média de tempo necessário para que a estação de controlo receba uma mensagem após a ocorrência de uma alteração na rede.

Após a apresentação de algumas métricas importantes na avaliação das possíveis topologias usadas em RSSF, são descritas de seguida diferentes topologias usadas em RSSF.

3.1.3 Topologias

As topologias tipicamente usadas em RSSF são, Flat, Cluster, Chain e Tree. Seguidamente é apresentada uma análise a cada uma destas topologias.

3.1.3.1 Flat

A topologia Flat ou Não Estruturada representa na realidade a ausência de topologia, onde cada nó (SN – Sensor-Node) desempenha o mesmo papel. Esta topologia é usada por protocolos de agregação de dados [69], recolha de dados [70], sincronização de nós [71] e de encaminhamento [72].

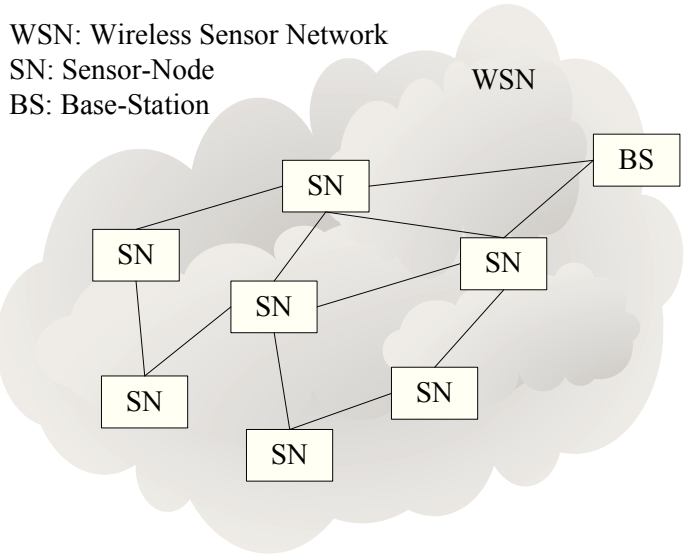

Figura 12 - Topologia Flat

Ao usar uma topologia Flat, todos os protocolos tentam por meio de *flooding* encontrar uma rota de boa qualidade desde os nós origem até à estação-base. *Flooding* é uma técnica onde um dado nó faz *broadcast* das mensagens recebidas para os restantes nós da rede, sendo o processo repetido até que a mensagem atinja o seu destino. É de notar que esta técnica não leva em conta as limitações energéticas das RSSF.

Como resultado da técnica de *flooding* são criados dois problemas, nomeadamente, *implosion* e *overlap* [73]. A circulação de pacotes duplicados é um dos problemas, fazendo com que os nós recebam pacotes duplicados, causando um problema de *implosion*. A comunicação com a estação-base é feita através de rotas *multi-hop* usando os nós vizinhos como retransmissores. Alguns dos protocolos usados em topologias Flat são, por exemplo, o *"Sensor Protocols for Information via Negotiation"* (SPIN) [73] [74], o Directed-Diffusion [75] ou o Rumor-Routing [76].

As vantagens e desvantagens dos protocolos baseados em topologias Flat são apresentadas de seguida:

- **Vantagens da topologia Flat**
  - Boa qualidade nas rotas entre a origem e a estação-base;
  - Inexistência de *overhead* necessário para a manutenção da topologia.
- **Desvantagens da topologia Flat**
  - O principal método de comunicação baseia-se no *flooding* de mensagens, sendo que o *flooding* é uma técnica dispendiosa e tipicamente evitada pelos protocolos de RSSF;
  - Criação e transmissão de várias mensagens duplicadas. A existência de mensagens duplicadas exige largura de banda e processamento desnecessário. A redundância de mensagens aumenta também a latência na entrega de mensagens devido à elevada contenção do meio de transmissão;
  - Distribuição não uniforme da energia pelos vários nós da rede. Esta é uma das principais razões da redução do tempo de vida da RSSF;
  - Os nós não estão conscientes da adição ou remoção de outros nós;
  - Elevado grau de imprevisibilidade;
  - Atraso elevado.

3.1.3.2 Cluster

As topologias baseadas em Cluster têm vindo a ser largamente usadas em RSSF, usando protocolos orientados a recolha de dados [77], localização [78] ou encaminhamento [79][80][81][82]. O *clustering* é particularmente útil em aplicações que requerem escalabilidade. Quando se fala em escalabilidade, as questões de balanceamento de carga e agregação de dados estão normalmente associadas. Vários protocolos de encaminhamento recorrem ao uso de *clustering* para criar uma estrutura hierárquica, minimizando assim o custo das rotas usadas para comunicação com a estação-base. De uma forma geral, quando se fala de topologias Cluster usadas em RSSF, podem ser identificados três elementos fundamentais, o *Sensor-Node* (SN), o *Cluster-Head* (CH) e a *Base-Station* (BS).

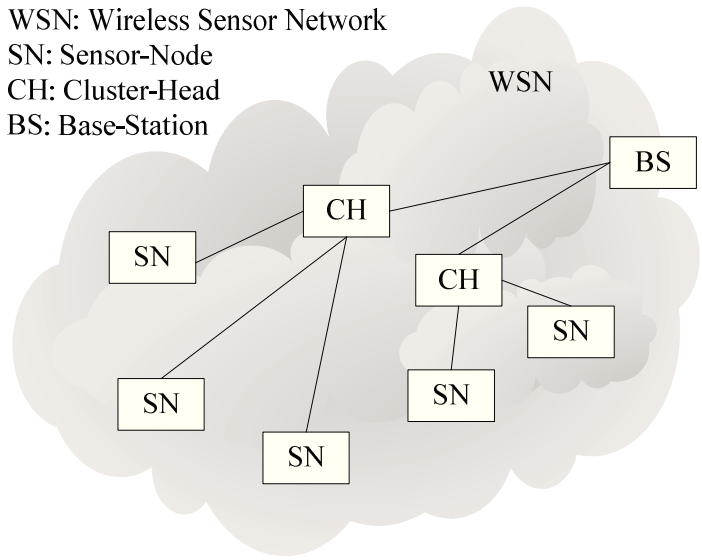

Figura 13 - Topologia Cluster

Os elementos SN têm como função a recolha ou analise de dados/eventos, a Base-Station (BS) é o ponto de saída da RSSF, enquanto que os CH atuam como um intermediário entre os SN e a BS, podendo desempenhar outras funções como por exemplo a agregação de dados. Pode-se dizer que os CH são o ponto de saída dos elementos SN e a BS é o ponto de saída dos elementos CH. A estrutura formada entre os vários elementos pode ser repetida quantas vezes forem necessárias, criando várias camadas hierárquicas na RSSF.

As classificações mais comuns para os vários tipos de Cluster são homogéneos/heterogéneos e estáticos/dinâmicos. Em redes homogéneas, todos os nós possuem as mesmas características e capacidades, podendo assim o papel de CH rodar entre os vários elementos do *cluster* de maneira a balancear a carga, equilibrando o consumo energético pelos vários elementos. Em redes heterogéneas, elementos com maiores capacidades são tipicamente usados para criar um *backbone* na RSSF, sendo estes designados por CH. Elementos com menores capacidades são usados para a aquisição de dados ou controlo de eventos. *Clusters* estáticos são de fácil implementação mas o seu uso é apenas indicado onde o ambiente e cenário de operação é pré-determinado, os alvos de monitorização são estáticos e onde as operações de manutenção (substituição de baterias) são de fácil execução. Por outro lado, os *clusters* dinâmicos podem fazer melhor uso dos nós-sensor, podendo estes pertencer a diferentes clusters em dados momentos. Durante o processo de criação dos *clusters* devem ser levados em conta vários aspetos, como por exemplo, o seu tamanho, forma, seleção de CH, entre outros.

Em todos os protocolos de *clustering*, podem ser identificadas três fases:

- Eleição do CH;
- Formação do *cluster*;
- Transmissão.

Existem várias abordagens para a implementação de cada uma destas três fases, por exemplo, pode ser usada uma distribuição estática de elementos SN e CH ou recorrer a algoritmos para a eleição de CH.

A formação do *cluster* pode ser feita de uma das seguintes formas [83]:

- Métodos probabilísticos. O protocolo *"Low Energy Adaptive Clustering Hierarchy"* (LEACH) [84] é um exemplo deste método;
- Eleição. Os vários nós divulgam a sua informação para toda a rede e com base nessa informação é selecionando um CH;
- Definido pela BS. Após a instalação dos vários nós, cada um deles comunica com a BS e com base na informação de cada nó, a BS tenta criar a melhor estrutura possível. Apesar da BS poder definir a melhor estrutura, este método é pouco usado uma vez que o custo de comunicação direta dos nós com a BS é elevado e por vezes impossível. O protocolo "*Base station Controlled Dynamic Clustering Protocol"* (BCDCP) [85] é um exemplo deste método.

As vantagens e desvantagens dos protocolos baseados em topologias Cluster são apresentadas de seguida:

- **Vantagens da topologia Cluster**
  - Escalabilidade;
  - Menor consumo energético em comparação com soluções baseadas em topologia Flat;
  - Eficiente capacidade de agregação de dados;

- Melhor utilização do meio de comunicação;
- Adequado aos paradigmas de comunicação *one-to-many* e *many-to-one*.

➢ **Desvantagens da topologia Cluster**

- A taxa de dissipação de energia poderá ser diferente entre SN, mesmo estando estes no mesmo *cluster*. Desta forma a distribuição de energia não é uniforme;
- A dissipação total de energia aumenta devido à longa distância de comunicação entre um SN e o seu CH, podendo causar o rápido desgaste dos nós e consequentemente reduzir o tempo de vida da RSSF;
- A conectividade da rede não é garantida.
- Os CH têm tendência para ficar sem energia mais rapidamente que um Cluster-Member (CM), sendo assim necessário ter em conta a possibilidade de atribuição das funcionalidades de CH a um CM.

3.1.3.3 Chain

Os protocolos baseados em topologias *Chain* criam uma cadeia de transmissão entre os vários nós que constituem a RSSF de maneira a reduzir a dissipação de energia na transmissão de mensagens. É selecionando um nó que atuará como o ponto de saída da rede e todos os nós comunicam através da cadeia de nós. A mensagem é enviada ao próximo nó da cadeia, chamado de nó sucessor até chegar ao nó líder. Com esta abordagem, todos os nós agregam a sua informação à medida que as mensagens passam de nó para nó. Este tipo de comunicação facilita a agregação de dados.

Existem vários protocolos baseados em topologias Single-*Chain* e *Multi-Chain*, como por exemplo o *"Power-Efficient. GAthering in Sensor Information Systems"* (PEGASIS) [86], *"Chain Oriented Sensor Network"* (COSEN) [87] e *"Chain-Based Hierarchical Routing Protocol"* (CHIRON) [88]. Usando como exemplo o PEGASIS, vê-se que cada nó da cadeia adquire dados, recebe dados do seu predecessor, agrega os seus dados e transmite a informação para o nó seguinte da cadeia. O objetivo deste protocolo visa prolongar o tempo de vida da RSSF, já que os nós apenas têm de comunicar com os seu vizinhos próximos e através deles chegar à estação-base. Estudos mostram que o PEGASIS permite uma poupança energética de cerca de 50% quando comparado com o protocolo LEACH. Isto deve-se ao

facto da inexistência do *overhead* causado pela formação dinâmica do Cluster e reduzindo o número de retransmissões recorrendo à agregação de dados. Apesar do *overhead* causado pela formação do Cluster não existir, o protocolo PEGASIS continua a necessitar de manutenção dinâmica da topologia, uma vez que cada nó necessita de saber qual o estado energético dos seus vizinhos de maneira a poder encaminhar as mensagens. Este tipo de gestão pode gerar algum *overhead*, especialmente em RSSF com elevado tráfego. Por outro lado, o protocolo PEGASIS introduz elevado atraso na entrega de mensagens entre nós afastados na cadeia. Além disso, o uso de apenas um líder pode provocar problemas de *bottleneck*. Na maioria dos casos o PEGASIS assume que os nós são estáticos e que se manterão imóveis durante o seu tempo de vida. Contudo, alguns nós poderão movimentar-se afetando assim a funcionalidade do protocolo.

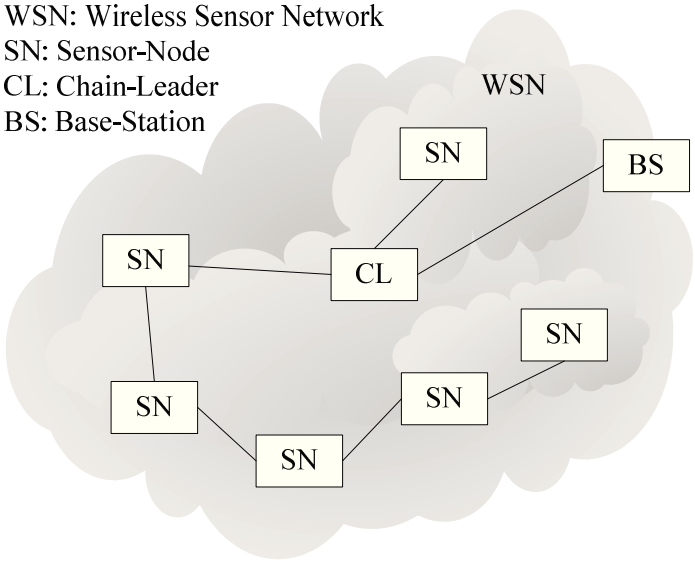

Figura 14 - Topologia Chain

As vantagens e desvantagens dos protocolos baseados em topologias *Chain* são apresentadas de seguida:

- **Vantagens da topologia Chain**

  - Devido à comunicação entre vizinhos próximos, as topologias *Chain* consomem menos energia que as topologias Cluster;

  - A dissipação energética numa topologia *Chain* é uniformemente distribuída pelos nós que constituem a cadeia;

- Devido à melhor gestão energética, as RSSF baseadas em topologias *Chain* acabam por ter mais tempo de vida.

➢ **Desvantagens da topologia Chain**

- Demasiado atraso na entrega e coleção de dados;
- O *overhead* exigido na gestão da topologia é elevado.

3.1.3.4 Tree

Neste tipo de topologia, os nós que constituem a RSSF formam um árvore lógica representando as rotas entre os vários nós. As mensagens são encaminhadas dos nós filhos para o seu nó pai, permitindo assim aos nós pai a agregação de dados à medida que recebem e reencaminham as mensagens dos seus nós filhos. O principal objetivo que se pretende atingir ao recorrer a uma topologia baseada em Tree está relacionado com a possibilidade de usar *unicast* e não *broadcast* para a transmissão de mensagens. Desta forma, pode ser poupada energia.

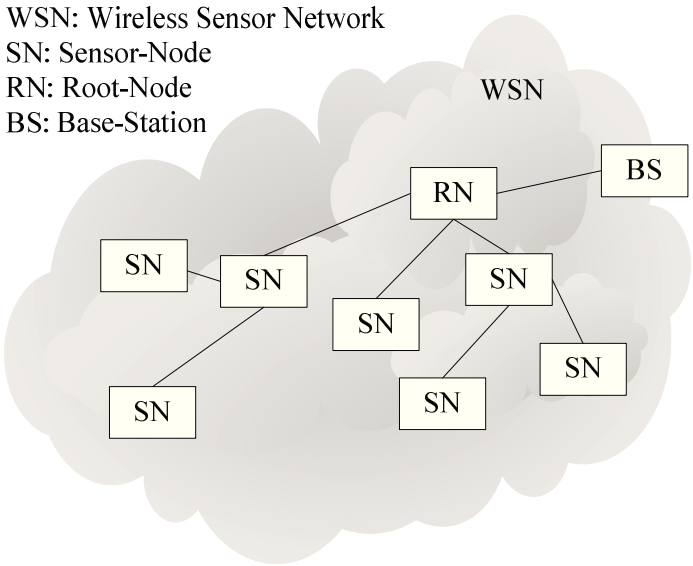

Figura 15 - Topologia Tree

Vários tipos de protocolo são baseados em topologias Tree, como protocolos de recolha de dados [89], protocolos de encaminhamento [90][91] ou protocolos de disseminação de dados [92][93].

As vantagens e desvantagens dos protocolos baseados em topologias Tree são apresentadas

de seguida:

- **Vantagens da topologia Tree**
  - Uma vez que a técnica de *flooding* não é necessária para a comunicação entre os nós da RSSF, o consumo energético acaba por ser inferior quando comparada com topologias Flat;
  - Em aplicações baseadas na aquisição de dados, o consumo energético pode ser ligeiramente inferior quando comparado com alguns protocolos baseados em topologias Cluster [94].
- **Desvantagens da topologia Tree**
  - A formação da árvore é um processo moroso e dispendioso;
  - Não resistente a falhas. Em caso de falha de um nó pai, toda a subárvore perde a conectividade com a estação-base;
  - O consumo de energia é desigual entre os nós da rede. Os nós mais próximos da estação-base consomem um nível de energia muito mais elevado em comparação com os nós mais afastados;
  - Longo atraso no envio de mensagens desde o nó filho até ao nó raiz;
  - A sobrecarga provocada pela manutenção da árvore é elevada.

Seguidamente, as várias topologias apresentadas (Flat, Cluster, Chain e Tree) são comparadas tendo em conta as métricas apresentadas anteriormente.

3.1.4 Avaliação

Tendo em conta as métricas definidas e as vantagens e desvantagens de cada topologia, é de seguida apresentada a avaliação da relação entre topologia e métrica.

3.1.4.1 Eficiência Energética

A eficiência energética é um das métricas mais importantes em RSSF dadas a limitações energéticas e impossibilidade de manutenção em determinadas aplicações. A comunicação é

a atividade que despende mais energia dos nós da RSSF. As topologias e protocolos desempenham um papel importante na otimização do consumo energético de toda a rede, prologando o tempo de vida da mesma.

A comunicação *multi-hop* usada na topologia Chain exigirá menor energia do que a comunicação *single-hop* de longa distância usada em topologias Cluster [86][95] já que a comunicação é feita através dos nós vizinhos de maior proximidade. O consumo energético dos nós deve ser distribuído de forma uniforme, de maneira a que o tempo de vida da RSSF seja o mais elevado possível.

Nas topologias Cluster e Tree, os nós CH e Pai, respetivamente, processam mais tráfego que o nó líder da topologia Chain. Como resultado, esses nós irão ficar sem energia mais rapidamente que o resto dos nós pondo em causa a conetividade com a estação-base. Esta é uma problemática conhecida como *"black-hole effect"*. Simulações mostram que tipicamente, a energia necessária para transmitir um bit pode ser entre 100 a 1000 vezes superiores à execução de uma instrução [96], tornando o processamento *"in-netwok"* um aspeto importante, efetuando agregação e compressão de dados.

- **Flat:** Numa topologia Flat, as rotas não são definidas previamente, limitando a possibilidade de processamento *"in-network"*.

- **Cluster:** Numa topologia Cluster, onde os elementos do *cluster* chegam ao CH através de comunicação *single-hop*, apenas os CH podem ser usados para agregação e pré-processamento.

- **Tree:** Numa topologia Tree, os vários nós acabam por ter mais oportunidade para agregação e pré-processamento dos dados.

- **Chain:** A topologia Chain, na sua essência, é orientada a processamento *"in-network"*. Neste tipo de topologia, cada nó pode ser usado para processamento dos dados enviados pelo seu precedente, reduzindo a necessidade de comunicação e o tráfego gerado através da agregação de dados.

Contrariamente aos CH da topologia Cluster, o líder da cadeia usado nas topologias Chain, não é responsável por toda a agregação de dados. Todos os nós da cadeia participam na agregação de dados, balanceando a carga pelos vários nós o que acaba por prolongar o tempo

de vida da RSSF.

Em resumo, em relação à eficiência energética temos o seguinte resultado, em que 1 é o mais eficiente e 4 o menos eficiente:

1) Chain;
2) Cluster;
3) Tree;
4) Flat.

3.1.4.2 Fiabilidade

A fiabilidade é um dos parâmetros mais importantes a ter em conta no desenho de RSSF [92]. A definição de fiabilidade pode ser feita como sendo a probabilidade do correto funcionamento da RSSF durante um determinado período de tempo [97].

- **Flat:** Em caso de falha de um nó, o uso da topologia Flat reduz a probabilidade de falha de toda a RSSF, já que, essa falha afeta apenas uma zona, deixando o resto da RSSF intacta. A topologia Flat é muito tolerante a falhas uma vez que apresenta diversos caminhos alternativos na rede. Em caso de falha de um nó ou de um *link* entre nós, a rede tem possibilidade de se reconfigurar facilmente, tirando partido da densidade de nós. Além disso, a adição de nós a uma topologia Flat acaba por aumentar a sua fiabilidade. A adição de mais nós acaba também por expandir o alcance da comunicação através de rotas *multi-hop*.

- **Tree:** A topologia Tree acaba por oferecer a menor fiabilidade uma vez que usa apenas um *link* entre a sucessão de níveis que definem a rede.

- **Chain:** A topologia Chain garante melhor fiabilidade que a topologia Tree uma vez que usa *links multi-hop* no mesmo nível hierárquico, desde a origem ao nó de saída.

- **Cluster:** A topologia Cluster oferece melhor fiabilidade do que as topologias Tree e Chain, uma vez que mantem rotas *multi-hop* mas pior em comparação com a topologia Flat visto que a comunicação entre nós de diferentes *clusters* necessita de ser encaminhada através de nós CH, daí a falha de um CH baixar significativamente a

fiabilidade.

Em resumo, em relação à fiabilidade temos o seguinte resultado, em que 1 é o mais eficiente e 4 o menos eficiente:

1) Flat;
2) Cluster;
3) Chain;
4) Tree.

3.1.4.3 Escalabilidade

As RSSF devem ser escaláveis, tentando manter a performance como um fator independente do número de nós em uso. Para além disso, a RSSF deve ser flexível aos requisitos da aplicação, levando em conta as possíveis alterações que podem ocorrer na rede. A capacidade de auto-organização ajuda a estender o tempo de vida da RSSF. Contudo, deve ser levado em conta as exigências energéticas e de tempo de resposta inerentes às ações de auto-organização, como por exemplo, a identificação/localização/recuperação de falhas. Por vezes, fazer com que alguns dos nós deixem de ser usados pode apresentar melhor desempenho energético [98].

- **Flat:** Numa topologia Flat, o uso de um elevado número de nós aumenta a carga da estação-base, resultando num elevado consumo energético e complexidade. Com o aumento do número de nós, o número de colisões será superior afetando assim a performance da RSSF. Para além disso, nem todos os nós têm poder de transmissão suficiente para comunicar com uma estação-base que se encontra a longa distância. Por todos estes fatores, torna-se difícil escalar uma RSSF baseada numa topologia Flat.

- **Tree:** Para topologias Tree, a auto-organização deve ser limitada até pré-determinados níveis da árvore. Depois dessa definição, os protocolos desenhados devem levar em conta o plano de transmissões e o agendamento de *duty-cycles*, de maneira a evitar os problemas de densidade de nós. Na prática, as topologias Tree escalam bem até redes de média dimensão mas apresentam problemas de degradação de performance em redes de elevada densidade [99].

- **Chain:** A escalabilidade e auto-organização de topologias Chain dependem do número de cadeias em uso na rede. Usando uma cadeia única (PEGASIS), a topologia Chain apresenta os mesmos problemas que os apresentados para a topologia Tree. O uso de topologias multi-cadeia melhoram as capacidades de escalabilidade em comparação com a topologia Flat, já que define nós líder para coordenar os nós vizinhos [92][93]. O uso de topologias Chain multi-cadeia permite o agendamento periódico de reorganizações, oferecendo adaptabilidade nas capacidades de auto-organização, tanto em eventos de falha de nós como na adição de novos nós.

- **Cluster:** Tal como a topologia Chain, a topologia Cluster apresenta também a existência de nós que coordenam os nós vizinhos. O uso de protocolos como o LEACH ou COSEN [92][93] usam coordenação localizada na tentativa de permitir a escalabilidade e a robustez em redes dinâmicas. As capacidades de auto-organização periódicas também são uma vantagem em caso de falha ou adição de nós.

O endereçamento é também uma das características que em muito influenciam a escalabilidade da RSSF. Por exemplo, o endereçamento livre de arquitetura [100], aproveita a localização espacial e temporal da RSSF para atribuir identificadores únicos para cada nova transação mas apenas escala com a densidade de transações da RSSF, enquanto que endereços estáticos, atribuídos globalmente, devem escalar com o número total de nós da RSSF. As topologias Tree e Cluster são inerentemente passíveis a uma estrutura de endereçamento escalável onde os vários nós da RSSF são endereçados tendo por base a sua posição na hierarquia. Este tipo de endereçamento permite o desenvolvimento de protocolos de encaminhamento simples.

Em resumo, em relação à escalabilidade temos o seguinte resultado, em que 1 é o mais eficiente e 4 o menos eficiente:

1) Cluster;

2) Tree;

3) Chain;

4) Flat.

3.1.4.4 Latência

Dependo da aplicação usada, a latência da RSSF pode ser um fator que define o sucesso ou insucesso de toda a aplicação. Uma estrutura de rede *single-hop* desde o nó origem até à estação-base seria a implementação ideal no que à latência diz respeito, uma vez que não será introduzido atraso provocado por encaminhamento. Contudo, esta implementação não é viável devido a questões energéticas e de colisão em redes com elevada densidade.

- **Flat:** Em comparação com uma estrutura *single-hop* até à estação-base, o uso de uma topologia Flat apresentará maior latência na entrega dos dados, mas a perda de dados será inferior uma vez que a baixa potência de transmissão necessária para transmitir para os nós vizinhos reduz as taxas de colisão significativamente. Dependendo do número de nós e da distância entre eles, a topologia Flat poderá apresentar problemas de sobrecarga nos nós mais próximos da estação-base. Esta sobrecarga poderá causar latência na comunicação, e no pior dos casos criar um *black-hole* de nós sobrecarregados em torno da estação-base.

- **Tree:** Em topologias Tree, uma vez que os dados se movem a partir de níveis inferiores para níveis superiores, acabam por percorrer uma distância maior, reduzindo assim a latência na entrega dos dados. Contudo, à medida que a distância dos níveis aumenta, a dissipação de energia aumenta. O uso de *clustering* tenta minimizar o consumo de energia e a latência na entrega dos dados.

- **Chain:** Na topologia Chain, todos os nós da cadeia executam tarefas de agregação. Em comparação com outras topologias, este processamento em cada salto da cadeia introduz uma maior latência na entrega dos dados.

- **Cluster:** Na topologia Cluster apenas os nós CH realizam tarefas de agregação, ao contrário da topologia Chain. Como resultado da arquitetura hierárquica da topologia Cluster, a latência da transmissão de dados será menor do que a latência introduzida pela topologia Chain.

Em resumo, em relação à latência temos o seguinte resultado, em que 1 é o mais eficiente e 4 o menos eficiente:

1) Cluster;

2) Tree;

3) Flat;

4) Chain.

3.1.4.5 Eficiência

A métrica relativa à eficiência da RSSF, relaciona o rácio entre os dados uteis e o *overhead* gerado para que a comunicação seja possível. O *overhead* pode ser criado por tarefas de encaminhamento ou por gestão da topologia.

- **Flat:** A topologia Flat produz o maior número de pacotes com necessidade de encaminhamento. Devido ao *flooding* usado na comunicação, um nó pode receber vários pacotes com a mesma informação vinda de vários nós vizinhos. Em termos de sobrecarga de mensagens de controlo, sendo esta sobrecarga a relação entre as mensagens de dados e as mensagens de controlo da rede, a topologia Flat apresenta os melhores resultados porque não necessita de manter a informação sobre toda a estrutura de rede, evitando assim a necessidade de envio de mensagens de controlo.

- **Cluster:** Nas topologias Cluster, os nós CH recebem dados dos vários membros do *cluster* (CM) e por isso apresenta *overhead* provocado por mensagens de controlo mas em número menos significativo em comparação com a topologia Tree.

- **Tree:** Na topologia Tree, devido à constante necessidade de manutenção da topologia, o número de mensagens de controlo é elevado.

- **Chain:** Numa topologia Chain, um nó da cadeia recebe dados apenas de um outro nó, o seu antecessor. Desta forma, em termos de sobrecarga de comunicação, a topologia Chain apresenta os melhores resultados.

Em resumo, em relação à eficiência temos o seguinte resultado, em que 1 é o mais eficiente e 4 o menos eficiente:

1) Flat;

2) Chain;

3) Cluster;

4) Tree.

3.1.5 Resumo e Conclusão

Após a avaliação das várias métricas para cada uma das topologias temos a seguinte tabela classificativa:

Métrica	Chain	Cluster	Flat	Tree
Eficiência Energética	1	2	4	3
Fiabilidade	3	2	1	4
Escalabilidade	3	1	4	2
Latência	4	1	3	2
Eficiência	2	3	1	4
Total	**13**	**9**	**13**	**15**

Tabela 4- Classificação de Topologias

Sendo que o menor valor total representa a melhor classificação geral, vemos que a topologia Cluster apresenta o melhor resultado no conjunto das métricas definidas.

Mediante o modelo conceptual definido, a topologia Cluster é a que melhor responde à generalidade dos requisitos apresentados, nomeadamente em resposta às seguintes necessidades:

- Escalabilidade da rede, permitindo a relativa fácil adição de novos nós a uma RSSF em funcionamento;
- Otimização do consumo energético, permitindo o uso de *duty cycles* e rotação de CH;
- Sendo a RSSF constituída por nós heterogéneos, os recursos com maiores capacidades de processamento e de energia podem assumir as tarefas de CH;
- Diferentes funções desempenhadas pelos vários CM (DE e ATE) com possibilidade de variação de *duty cycles*. Os CM com funções de ATE podem otimizar o seu consumo energético entrando em modo *idle*, atuando apenas quando necessário (periodicamente), sem afetar a capacidade de comunicação da RSSF;
- Baixa latência na entrega de mensagens. Uma vez que o encaminhamento e agregação

é feita apenas pelos CH.

Após a definição da topologia Cluster como a indicada para a solução pretendida, é apresentada de seguida uma análise referente às técnicas de *clustering* usadas em RSSF.

3.2 Classificação das Técnicas de Clustering

As técnicas de *clustering* propostas para uso em RSSF podem ser geralmente classificadas com base na arquitetura geral da rede, no modelo operacional ou no objetivo do processo executado pelos vários elementos que constituem o cluster [101]. Seguidamente serão apresentadas as diferentes classificações e taxonomia dos atributos de *clustering*.

3.2.1 Modelo de Rede

Diferentes objetivos e arquiteturas têm sido considerados para os diversos tipos de aplicações de RSSF. De seguida são apresentados alguns dos importantes parâmetros da arquitetura e os seus impactos na rede de *clustering*, tais como, a dinâmica de rede, o processamento na RSSF e a implementação e capacidades dos elementos da rede.

3.2.1.1 Dinâmica de rede

Uma RSSF consiste em três componentes principais: elementos sensor, estação-base e eventos/espaços monitorizados. A maioria das arquiteturas de rede pode assumir que os elementos sensor são estacionários [102][103][104]. A mobilidade dos diversos elementos que fazem parte da RSSF torna o processo de agrupamento (*clustering*) uma tarefa desafiante, uma vez que, a associação de um elemento a um determinado *cluster* pode ser alterada dinamicamente. Desta forma, os elementos que constituem um *cluster* serão obrigados a evoluir/adaptar-se com o tempo. Dependendo da aplicação, os eventos controlados por um elemento sensor podem ser intermitentes ou contínuos [101]. A monitorização de eventos intermitentes possibilita à rede trabalhar num modo reativo, gerando tráfego apenas quando determinado evento é verificado. Na maioria das aplicações, os eventos contínuos requerem relatórios periódicos, sendo assim gerada uma quantidade de tráfego mais significativa na comunicação com o elemento *gateway* (BS).

3.2.1.2 Processamento na RSSF

Uma vez que os elementos sensor podem gerar um volume significativo de dados

redundantes, a forma de diminuir a troca de mensagens e consequentemente otimizar o consumo energético da rede baseia-se na agregação de dados. A agregação de dados combina dados de diferentes fontes utilizando, por exemplo, funções que detetam dados duplicados [105]. Algumas destas funções podem ser realizadas parcialmente ou totalmente pelos elementos sensor que constituem a RSSF, sendo assim realizado o pré-processamento dos dados recolhidos na própria RSSF. Tendo em conta que o consumo energético é significativamente mais reduzido na computação do que na transmissão, o pré-processamento e agregação de dados acaba por permitir uma substancial poupança na energia consumida. Esta técnica tem sido aplicada em vários protocolos de encaminhamento, na tentativa de obter a melhor eficiência energética e otimização de tráfego. Tipicamente, as funções de agregação são atribuídas a elementos com maiores capacidades de processamento, como os elementos CH. Ao selecionar estes elementos, deve-se assegurar que não ficarão sobrecarregados com a tarefa de agregar os dados recolhidos por um elevado número de elementos sensor [106]. Dependendo da aplicação, por vezes pode ser necessário definir um CH de *backup* ou atribuir rotativamente o papel de CH pelos outros elementos do cluster [107][108]. A definição destas condições acaba por definir toda a arquitetura do *cluster*.

3.2.1.3 Implementação e capacidades do elemento

A topologia dos vários elementos aquando da sua implementação deve também ser considerada. Esta é uma característica dependente da aplicação e afeta todas as funcionalidades do *cluster*. A implementação pode ser determinística ou de auto-organização. Em situações deterministas, os elementos sensor são colocados manualmente e os dados recolhidos são encaminhados através de rotas predeterminadas. Em sistemas de auto-organização a localização dos elementos é aleatória, criando uma infraestrutura baseada numa topologia *Ad-hoc* [109][102][107]. Neste tipo de infraestrutura, a posição da estação base ou do CH é também crucial em termos de desempenho e eficiência energética. A distância de comunicação e a proximidade entre a estação base e os elementos CH pode causar restrições que devem ser consideradas. O alcance de transmissão é limitado e um CH pode não ser capaz de fazer chegar os dados transmitidos até à estação base. Contudo, mesmo que um elemento sensor consiga comunicar diretamente com a estação base, é melhor continuar a usar rotas *multi-hop* uma vez que, desta maneira, é provável que a distância de transmissão seja menor, exigindo assim menos energia não comunicação. Portanto, a comunicação *inter-CH* torna-se um fator importante que afeta o esquema de *cluster* [110][111].

3.2.2 Objetivos do Cluster

Os algoritmos de *clustering* variam nos seus objetivos. Muitas vezes, o objetivo de *clustering* é o de facilitar o cumprimento dos requisitos de aplicações. Seguidamente são apresentados alguns dos objetivos de redes de *clusters*:

3.2.2.1 Balanceamento de Carga

A equitativa distribuição de elementos sensor entre os *clusters* é geralmente um objetivo de configuração onde os elementos CH realizam processamento de dados ou executam tarefas de gestão *intra-cluster* [106].

Dadas as funções atribuídas aos elementos CH, é importante equilibrar a carga entre os vários CH de maneira a que possam atingir o desempenho esperado [112].

3.2.2.2 Tolerância a Falhas

Em várias aplicações, a RSSF opera em ambientes hostis onde a possibilidade de avaria ou dano físico de um elemento pode ser provável. A tolerância a falhas de um elemento CH é necessária neste tipo de aplicações, para que não sejam perdidos os dados recolhidos pelos elementos sensor. A forma mais simples para recuperar de uma falha de CH poderá ser reagrupar a rede (*recluster*). Contudo, o reagrupamento da rede é um processo pesado para os elementos da RSSF, devendo assim ser utilizadas técnicas mais eficazes para suplantar a falha de um CH. A definição de um CH de *backup* é uma técnica eficaz para responder a uma falha de CH. O método de seleção de um CH de *backup* pode variar se, por exemplo, os CH possuem um poder de alcance adequado, o CH vizinho poderá suportar os elementos sensor do CH em falha [108]. Alternar o papel de CH entre os vários elementos do *cluster*, além de efetuar o balanceamento de carga entre os vários elementos do *cluster*, pode também responder à falha do CH [107].

3.2.2.3 Conectividade e Redução do Atraso

Caso o CH não tenha uma elevada capacidade de alcance, a ligação *inter-CH* é essencial. Esta situação ocorre mais frequentemente quando o CH é escolhido aleatoriamente entre os elementos sensor disponíveis. O objetivo pretendido pode ser apenas a ligação entre os elementos CH de maneira a chegar até à estação base [110] ou mais restritiva, definindo, por exemplo, uma qualidade de ligação mínima [113]. Quando a latência dos dados é uma preocupação, a conectividade *intra-cluster* torna-se um objetivo ou uma restrição. O atraso é

geralmente controlado definindo um número máximo de saltos [114][115][116].

3.2.2.4 Máxima Longevidade de Vida da Rede

Dado que os elementos sensor são limitados em termos energéticos, o tempo de vida da rede é uma grande preocupação. Quando os CH possuem mais recursos que os elementos sensor, é imperativo minimizar a energia consumida na comunicação *intra-cluster* [117]. Se possível, os CH devem ser colocados próximo dos elementos sensor que são membros do *cluster* [118][119]. Por outro lado, quando os CH são elementos sensor normais, o seu tempo de vida pode ser estendido usando uma das técnicas de balanceamento de carga referidas anteriormente.

3.2.3 Atributos de Clustering

Seguidamente são apresentados os atributos que podem ser usados para diferenciar e categorizar os algoritmos de *clustering* para RSSF. Levando em conta as ideias apresentadas anteriormente, são identificados os seguintes atributos:

- Propriedades do *cluster*;
- Capacidades do *cluster*;
- Processo de *clustering*.

Este conjunto de atributos é apresentado de seguida

3.2.3.1 Propriedades do Cluster

As características do esquema de *cluster* podem ser relacionadas com a estrutura interna do *cluster* ou como esta se relaciona com os outros *clusters*. De seguida são apresentados os atributos relevantes:

- **Número de Clusters:** Em algumas abordagens o conjunto de CH é pré-determinado e a escolha aleatória do CH pode gerar um número variável de *clusters*;
- **Estabilidade:** Quando o número de *clusters* varia e a associação de elementos sensor cresce com o passar do tempo, diz-se que o esquema de *clustering* é adaptativo [101]. No caso contrário, considera-se que o número de elementos sensor associados a um dado CH não varia e o número de *clusters* permanece o mesmo durante toda a vida da

RSSF;

- **Topologia *Intra-Cluster*:** Alguns esquemas de *clustering* são baseados na comunicação *single-hop* entre o elemento sensor e o seu CH. No entanto, a comunicação *multi-hop* entre o elemento sensor e o CH é por vezes necessária, especialmente quando o alcance de transmissão do elemento sensor é limitado ou quando o número de CH é baixo;

- **Conectividade *Inter-CH*:** Quando o CH não possui capacidade de comunicação de longa distância, a conectividade com a estação-base terá de ser provisionada. Nestas situações, o esquema de *clustering* terá de garantir a possibilidade de comunicação entre CH, de maneira a que seja possível fazer chegar os dados até à estação-base.

3.2.3.2 Capacidades do Cluster-Head

Como referido, o modelo de rede influência a abordagem de *clustering*, particularmente as capacidades e âmbito do processamento *"in-network"*. Os seguintes atributos do CH são fatores distintivos entre os esquemas de *clustering*:

- **Mobilidade:** Quando um CH é móvel, a associação de elementos sensor pode mudar dinamicamente e a manutenção do *cluster* tem de ser efetuada continuamente. Por outro lado, os CH estacionários têm tendência a manter a estabilidade na associação de elementos sensor facilitando assim a gestão de rede *intra-cluster* e *inter-cluster*;

- **Tipo de Elemento:** Como referido anteriormente, em determinadas configurações um elemento sensor pode ser designado como CH enquanto noutras configurações os CH são elementos previamente definidos com maiores capacidades de comunicação e computação;

- **Função de um CH:** Um CH pode simplesmente agir como um intermediário para o tráfego gerado pelos elementos sensor do seu *cluster* ou executar também tarefas de agregação de dados.

3.2.3.3 Processo de *Clustering*

A coordenação de um *cluster* e as características dos algoritmos variam significativamente consoante a arquitetura de *clustering*. Os seguintes atributos são considerados relevantes:

- **Metodologia:** Quando os CH são elementos sensor regulares, o *clustering* terá de ser feito de uma forma distribuída e não coordenada. Algumas abordagens recorrem a uma autoridade centralizada que particiona os elementos e controla as associações ao *cluster*. Esquemas híbridos são também usados, especialmente quando os CH possuem elevada capacidade de processamento. Neste caso, a coordenação *inter-CH* é realizada de uma maneira distribuída, enquanto cada CH se encarrega de formar o seu próprio *cluster*;

- **Objetivo do *Clustering*:** Como discutido anteriormente, são vários os requisitos a ter em conta aquando da formação de um *cluster*. Exemplos disso são a tolerância a falhas, balanceamento de carga, conectividade de rede, etc.;

- **Seleção do CH:** A seleção de um CH pode ser feita previamente ou escolhendo aleatoriamente um dos elementos sensor disponíveis;

- **Complexidade de Algoritmo:** Dependendo do objetivo e da metodologia, inúmeros algoritmos de *clustering* podem ser usados. A taxa de convergência e a complexidade desses algoritmos pode ser constante ou em função do número de CH e/ou elementos sensor.

Tendo em conta as características da topologia *cluster* e o modelo conceptual definido, é apresentada a solução proposta no capítulo seguinte.

3.3 Arquitetura Proposta

Nesta secção é apresentada a arquitetura proposta, apresentando e definindo as várias entidades presentes na arquitetura e também a forma de interação entre essas mesmas entidades, de maneira a que possam atingir o objetivo pretendido.

3.3.1 Visão Global e Entidades da Arquitetura

Na Figura 16 é apresentada a arquitetura genérica da solução proposta, onde podem ser vistas as entidades essenciais para o funcionamento da solução.

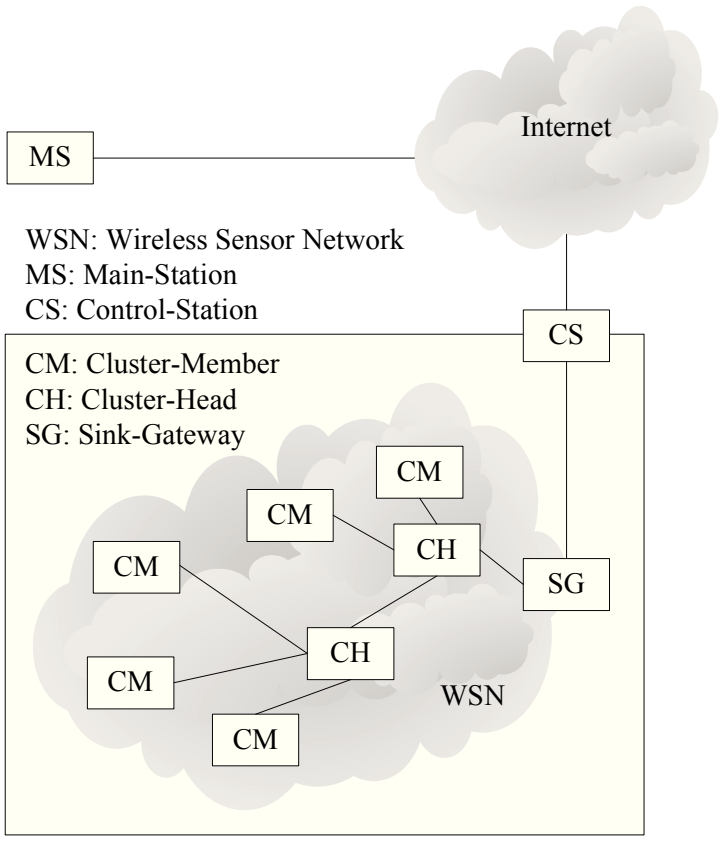

Figura 16 - Arquitetura Geral

Como apresentado na Figura 16, a arquitetura é composta por diversas entidades, sendo que, cada uma delas terá funções únicas e de elevada importância. Seguidamente serão apresentadas em detalhe, as funções e particularidades de cada uma das entidades envolvidas na arquitetura.

3.3.1.1 Pressupostos

A solução apresentada baseia-se numa RSSF *single-app*. Contudo, a arquitetura proposta permite a customização de parâmetros aplicacionais, tornando possível a personalização do comportamento dos vários elementos que constituem a RSSF.

O funcionamento deste método é explicado em maior detalhe no capítulo 0.

3.3.1.2 Cluster-Member

A entidade *Cluster-Member* (CM) é essencial para o fim pretendido por uma dada aplicação de RSSF. O CM tem como objetivo adquirir dados ou atuar de determinada forma, recorrendo a sensores e/ou atuadores, tal como pretendido pela aplicação.

Seguidamente podem ser vistas as particularidades que definem um CM.

- **Funções Base**

A função do CM é a aquisição de dados sensoriais e/ou atuação no meio envolvente. Dadas as funções de aquisição de dados sensoriais, será esta a entidade a gerar as mensagens que contêm a maioria dos dados pretendidos por uma dada aplicação. O tipo de dados recolhidos são dependentes da aplicação podendo estes ser, por exemplo, dados de elevado grau critico onde a urgência e garantia de entrega são vitais ou dados de recolha periódica usados para posterior análise, tendo por base grandes amostras de dados. Em qualquer dos casos, o CM é responsável pela criação e envio da mensagem.

Em resumo, as funções primárias de um CM são:

- Aquisição de dados e/ou atuação;
- Criação e envio de mensagens contendo informação útil à aplicação;
- Capacidade para receção de parâmetros de configuração da aplicação ou outros parâmetros úteis ao funcionamento da rede.

- **Associação e Presença no Cluster**

Para respeitar a topologia *Cluster Single-Hop*, um CM deverá sempre fazer parte de um grupo (*Cluster*), sendo esse conjunto gerido por um elemento designado para o efeito (*CH - Cluster-Head*). O CM é um elemento terminal da rede e o envio de mensagens deve ter como destino o CH do *Cluster* a que o CM pertence.

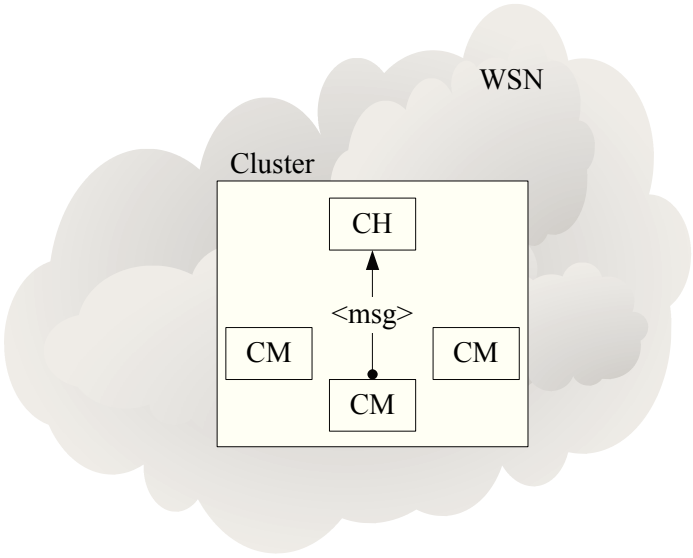

Figura 17 – Mensagem de CM para CH

Aquando da sua iniciação, um CM deverá perguntar à rede qual será o seu CH. Ao receberem um pedido de associação, os vários CH devem avaliar o pedido e responder positivamente ou negativamente ao CM. Após o sucesso da associação a um dado CH, as mensagens produzidas pelo CM terão como destino o CH atribuído.

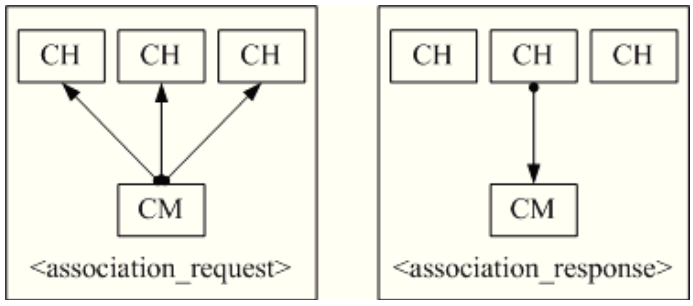

Figura 18 - Pedido de associação ao Cluster

- **Modo de Operação (Idle/Active)**

Mediante o tipo de dados recolhido e os requisitos da aplicação, podem ser definidos períodos de atividade e inatividade (*idle*) do CM. Seguindo esta abordagem, o tempo de vida da rede é estendido uma vez que, durante os períodos de inatividade o consumo energético é mínimo. Usando o exemplo de uma aplicação que tem como finalidade a recolha da temperatura de um determinado espaço, não será necessária a constante aquisição de dados e o seu envio, dado que os valores recolhidos não irão variar de forma significativa num curto espaço de tempo. Podem então ser assim definidos períodos de atividade para o CM, onde

este recolhe e envia a informação periodicamente.

- **Entrega de Mensagens**

Tal como referido anteriormente, o CM irá enviar as suas mensagens ao CH do *cluster* a que pertence. O envio destas mensagens pode seguir uma política baseada em eventos, situações onde um determinado evento crítico é detetado e é imediatamente comunicado ou, poderá seguir uma abordagem de envio periódico, onde o CM recolhe os dados pretendidos, podendo ou não efetuar algum tipo de pré-processamento, e envia as mensagens quando o tempo definido para o ciclo de envio se esgotar. O uso de políticas na entrega de mensagens ao CH (imediatas ou periódicas) pode minimizar o tráfego gerado pelo CM e consequentemente otimizar o consumo energético.

- **Pilha-Protocolar**

Na Figura 19 pode ser vista a pilha protocolar de um CM, definindo as componentes presentes nas várias camadas da pilha.

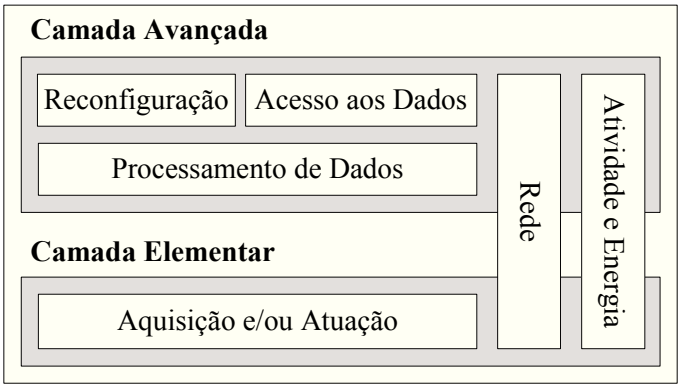

Figura 19 - Cluster-Member - Pilha Protocolar

- **Camada Avançada**

  - **Reconfiguração:** Fornece a inteligência necessária para a redefinição de parâmetros aplicacionais ou de rede, podendo assim a aplicação da RSSF ser customizada e adaptada a determinadas particularidades;

  - **Acesso aos Dados:** Capacidade para responder imediatamente a pedidos de consulta ou à execução de ações;

- **Processamento de Dados:** Análise e tratamento dos dados recolhidos, filtrando duplicações ou dados inválidos e aplicando tarefas de agregação. Dependendo do tipo de dados e severidade de mensagens poderá obedecer a períodos na entrega de mensagens ou enviar imediatamente os dados recolhidos.

- **Camada Elementar**

  - **Aquisição e/ou Atuação:** Responsável pelas tarefas de aquisição de dados e/ou execução de tarefas de atuação.

- **Componentes Transversais**

  - **Rede:** Responsável pelas tarefas de acesso ao meio, presença no *cluster*, endereçamento, encaminhamento e aplicação de políticas de QoS (filas prioritárias e balanceamento de carga).

    - Para além das funcionalidades de rede necessárias para a presença no *cluster*, a pilha-protocolar presente nos elementos CM deve suportar as capacidades de gestão de cluster, efetuando tarefas de encaminhamento e agregação dos dados enviados por outros CM. Em caso de falha do *Cluster-Head* (CH) a que o CM se encontra associado, deverá existir uma nova eleição de CH. Na impossibilidade de associação a um outro CH, um dos CM do *cluster* deverá ativar as funcionalidades necessárias e assumir as tarefas de CH de maneira a não por em causa todo o *cluster*. A este processo de reeleição dá-se o nome de *reclustering*. Visto que os nós CM em teoria serão nós com menores capacidades de processamento e energia, fica ao critério da aplicação a continuidade das tarefas de aquisição e/ou atuação.

  - **Atividade e Energia:** Controlo de períodos de atividade/*idle* e de informação relacionado com o estado energético.

3.3.1.3 Cluster-Head

A entidade *Cluster-Head* (CH) assume as tarefas essenciais para o correto funcionamento de toda a rede, nomeadamente, a gestão da associação e dissociação dos vários CM a um dado *cluster*.

As particularidades que definem um CH serão apresentadas de seguida.

- **Funções Base**

O nó CH tem como principais funções a gestão do *cluster*, agregação, disseminação e encaminhamento de mensagens, aplicação de políticas de segurança e qualidade de serviço, entre outras. No que diz respeito à gestão do *cluster*, o CH é responsável pelas tarefas de associação de um novo CM ao *cluster* pelo qual é responsável. A estas tarefas podem ser aplicadas políticas de segurança, permitindo ou não a associação do CM. Dependendo do tipo de dados recolhidos pelo CM, podem ser definidas também políticas de qualidade de serviço (QoS) para as mensagens enviadas pelo CM. Usando este tipo de mecanismo, onde são definidas regras de prioridade para um dado tipo de dados, o encaminhamento realizado pelo CH poderá ser otimizado, cumprindo assim os requisitos necessários a uma dada aplicação. Resumindo, as funções base da entidade CH, são:

- Tarefas de associação e dissociação de CM ao *cluster*;
- Aplicação de políticas de segurança e QoS;
- Agregação, filtragem e pré-processamento de dados de maneira a minimizar o tráfego de rede e consequentemente otimizar a performance de toda a RSSF;
- Encaminhamento de mensagens entre os diversos CH.

- **Gestão do Cluster**

As tarefas de gestão do *cluster* ficam a cargo do CH. Estas tarefas baseiam-se sobretudo no controlo e manutenção da informação acerca do estado dos CM pertencentes ao *cluster*. O CH deve permitir a associação e dissociação dos CM e comunicar esses eventos à entidade *Control-Station* (CS), de maneira a que seja mantido em tempo-real o conhecimento dos elementos presentes na rede.

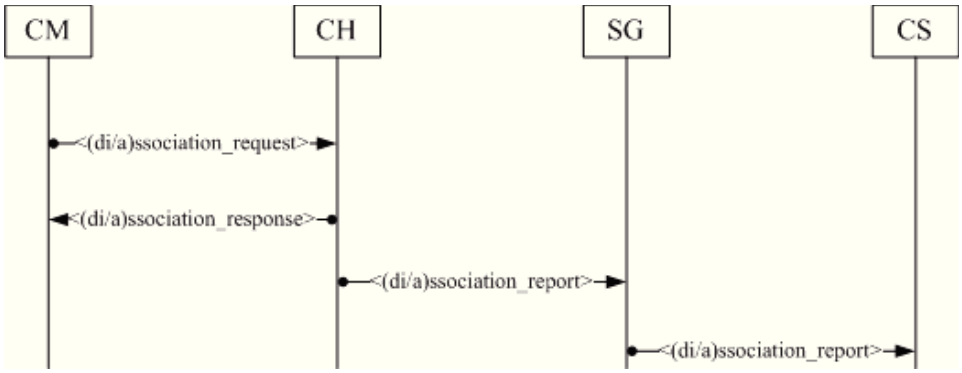

Figura 20 - Associação/Dissociação do CM ao Cluster

- **Eleição do Cluster-Head**

A eleição do CH pode ser realizada previamente, definindo estaticamente o elemento como CH, ou dinamicamente aquando da iniciação da rede. A escolha correta dos elementos CH pode diminuir a taxa de *reclustering* e consequentemente a sobrecarga da rede. Existem diversos algoritmos que podem ser utilizados para a eleição de um CH, nomeadamente o *"Lowest ID Clustering Algorithm"* [120], *"Highest Connectivity Clustering Algorithm"* [114], *"Least Cluster Change Algorithm"* [121] e *"Weighted Clustering Algorithm"* [122]. A capacidade de *reclustering* é fundamental para suprimir as falhas de CH provocadas por falhas de *hardware*, esgotamento natural da energia ou por perda de conectividade. Visto que o CH será um dos nós mais ativos na RSSF, a possibilidade de falha energética é superior quando comparado com os elementos CM.

- **Agregação e Filtragem de Dados**

Uma vez que o tráfego dos vários CM pertencentes ao *cluster* passam pelo CH, existe a possibilidade de aplicar algoritmos de agregação, evitando assim que dados não válidos ou duplicados se propaguem pela RSSF. Este tipo de filtragem irá facilitar o pós-processamento e minimizar o tráfego na rede.

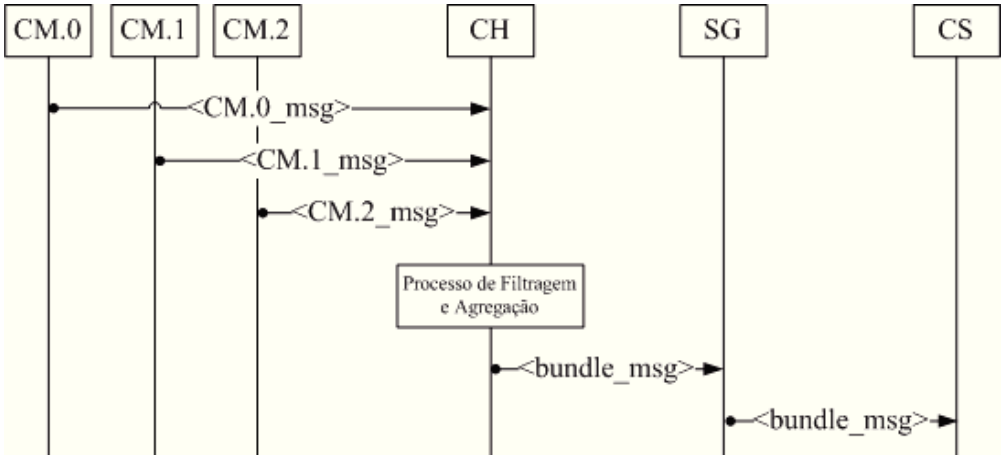

Figura 21 - Processo de Filtragem e Agregação

- **Funções Sensoriais**

Consoante as necessidades da aplicação ou o estado da rede num dado momento, um CH poderá acrescentar às suas funções as capacidades de aquisição de dados e/ou atuação no meio, tornando-se assim um CH com as funções extra de CM.

Esta particularidade pode ocorrer por diversos motivos, nomeadamente, a impossibilidade de associação de um CM a um dado *cluster* ou por falha do CH ativo no *cluster*. Nestas situações, um CM deverá ser capaz de assumir as tarefas de um CH. Esta ação obriga a que a tarefa de *reclustering* seja efetuada, reelegendo novos CH, de maneira a não colocar em causa o funcionamento do *cluster*.

Visto que a RSSF é composta por nós heterogéneos, idealmente os nós com maiores capacidades de processamento e energia devem assumir o papel de CH, contudo, no limite, todos os nós CM podem assumir as tarefas de CH visto que possuem a pilha-protocolar necessária para desempenhar as funções de gestão do *cluster*.

- **Problemática de Single-Point-Of-Failure**

Os elementos CH têm um papel essencial na RSSF, desempenhando tarefas de agregação e encaminhamento. Sendo o CH o ponto central e único de um dado *cluster* a falha deste põe em causa a entrega dos dados recolhidos ou gerados pelos CM, podendo assim comprometer toda a aplicação da RSSF.

De maneira a solucionar a problemática de *"Single-Point-Of-Failure"* (SPOF), em caso de

falha de *hardware* ou falha iminente devida a esgotamento energético, as funcionalidades de um CH poderão ser atribuídas a um CM existente no *cluster* caso o algoritmo de *reclustering* não consiga encontrar um CH alternativo para associar os CM órfãos.

Como referido anteriormente, a eleição de um novo CH poderá ser definida tendo em conta um nível de prioridade pré-definida pela aplicação ou recorrendo, por exemplo, a estatísticas da rede, selecionando o CH/CM com menos carga de processamento e/ou transmissão.

Aquando da eleição de um novo CH, o algoritmo de *reclustering* deve atualizar a topologia da RSSF e comunicar as alterações à CS.

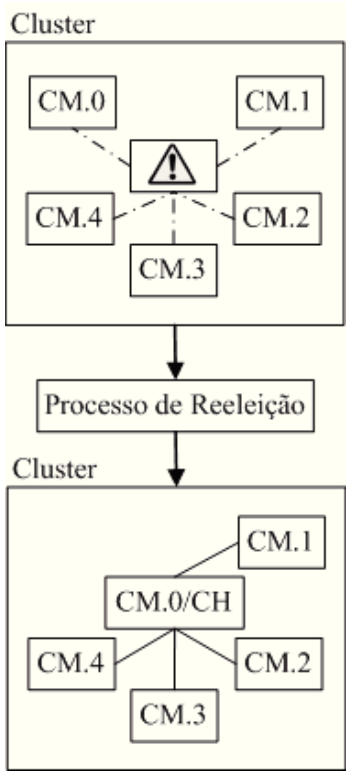

Figura 22 - Reeleição do Cluster-Head

- **Pilha-Protocolar**

Na Figura 23 pode ser vista a pilha protocolar de um CH, definindo as componentes presentes nas várias camadas da pilha.

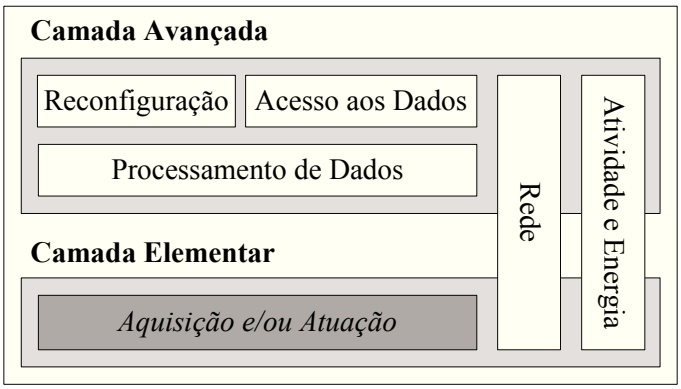

Figura 23 - Cluster-Head - Pilha Protocolar

- **Camada Avançada**

  - **Reconfiguração:** Fornece a inteligência necessária para a redefinição de parâmetros aplicacionais ou de rede, podendo assim a aplicação da RSSF ser customizada e adaptada a determinadas particularidades;

  - **Acesso aos Dados:** Capacidade para responder imediatamente a pedidos de consulta ou à execução de ações, diretamente no CH em questão ou nos vários CM que fazem parte do *cluster*;

  - **Processamento de Dados:** Análise e tratamento dos dados recolhidos pelos CM, filtrando duplicações ou dados inválidos e aplicando tarefas de agregação. Dependendo do tipo de dados e severidade de mensagens poderá obedecer a períodos na entrega de mensagens ou enviar imediatamente os dados recolhidos.

- **Camada Elementar**

  - **Aquisição e/ou Atuação:** Esta camada não é essencial para desempenhar as funções de CH mas será necessária para efetuar as tarefas de aquisição de dados e/ou execução de tarefas de atuação nos casos em que um CH é previamente instalado com capacidades de aquisição e/ou atuação.

- **Componentes Transversais**

  - **Rede:** Responsável pelas tarefas de acesso ao meio, gestão do *cluster*

(associação, dissociação, etc.), endereçamento, encaminhamento e aplicação de políticas de QoS (filas prioritárias e balanceamento de carga);

- **Atividade e Energia:** Controlo de períodos de atividade/*idle* dos CM do *cluster* e de informação relacionado com o estado energético.

3.3.1.4 Sink-Gateway

A entidade *Sink-Gateway* (SG) poderá ser um elemento independente ou fazer parte da *Control-Station* (CS). A sua principal função será interligar (*Bridge*) a RSSF com a CS, uma vez que, normalmente a RSSF usa uma tecnologia de transmissão não compatível com a CS, por exemplo, o uso de IEEE 802.15.4 na RSSF e Ethernet na CS.

De seguida são apresentadas as particularidades associadas a um SG.

- **Funções Base**

Os dados gerados na RSSF e encaminhados pelos CH têm como destino o SG, para que a lógica de tratamento de dados possa, posteriormente, ser aplicada pela CS. Tipicamente, o SG não recorre a baterias, contudo podem ocorrer outro tipo de falhas, tornando assim o uso de vários SG uma opção a ter em conta.

Em resumo, as funções base da entidade SG são as seguintes:

- Interface entre a RSSF e a CS, fazendo ponte entre (*Bridging*) as mensagens trocadas entre a RSSF e a CS;
- Interligação a outras redes;
- Tradução de endereços entre os dois segmentos de rede.

- **Ligação a Outras Redes**

Uma vez que o SG será a interface de ligação entre a RSSF e a CS, terá como responsabilidade a interligação a outras redes, como por exemplo, uma rede IP. Os endereços usados na RSSF podem ser mapeados para um endereço IP, fazendo com que os dados saídos da RSSF tenham uma representação IP e assim, facilitar a integração da RSSF com outras redes. Nesse sentido, o SG é o elemento responsável pelo mapeamento, mantendo para isso,

uma tabela de mapeamento devidamente atualizada. Tal como referido anteriormente, os CH devem reportar o estado das associações ao SG/CS para que seja possível manter a tabela de mapeamento (SG) e para, por exemplo, executar tarefas de reconfiguração e análise (CS).

Endereçamento Interno da RSSF	Endereçamento Externo (IP)
3.4	192.168.3.4
127.15	192.168.127.15
3.12.4	10.3.12.4

Tabela 5 - Exemplo de endereçamento Interno e Externo à RSSF

- **Problemática de Single-Point-Of-Failure**

Tal como os elementos CH, os SG sofrem também da problemática SPOF. A falha de um SG é ainda mais prejudicial para a RSSF, uma vez que, em caso de falha, os dados não poderão ser encaminhados até à CS. Para contornar este problema podem ser seguidas várias abordagens, como por exemplo, o uso de dois ou mais SG, podendo estes seguir o paradigma *Active-Standy* ou *Active-Active*.

No caso do paradigma *Active-Active*, poder-se-á tirar partido dos vários SG e recorrer a técnicas de balanceamento de carga, equilibrando assim o tráfego da rede pelos vários *gateways*. O uso de vários SG irá otimizar também o consumo energético dos CH, uma vez que, ao existirem vários SG, o tráfego encaminhado pelos CH pode ter como destino um dos SG mais próximos, minimizando os problemas de transmissão causados pela longa distância entre os CH e o SG.

- **Pilha-Protocolar**

Na Figura 24 pode ser vista a pilha protocolar de um SG, definindo as componentes presentes nas várias camadas da pilha.

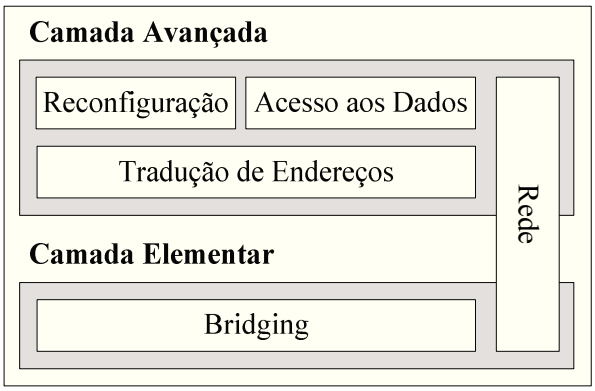

Figura 24 - Sink-Gateway - Pilha Protocolar

- **Camada Avançada**

  - **Reconfiguração:** Fornece a inteligência necessária para a redefinição de parâmetros aplicacionais ou de rede. Podendo assim a aplicação da RSSF ser customizada e adaptada a determinadas particularidades;

  - **Acesso aos Dados:** Capacidade para responder imediatamente a pedidos de consulta (por exemplo, estatísticas) ou à execução de ações (por exemplo, ativação/desativação de interfaces);

  - **Tradução de Endereços:** Mapeamento do endereçamento usado na RSSF (por exemplo, endereçamento hierárquico – "3.1") para um outro esquema de endereçamento (por exemplo, endereçamento IPv4 – 192.168.3.1).

- **Camada Elementar**

  - **Bridging:** Capacidade para interligação de vários segmentos de rede, como por exemplo, IEEE 802.15.4 e IEEE 802.3. Através desta interligação será possível comunicar com a CS.

- **Componentes Transversais**

  - **Rede:** Responsável pelas tarefas de acesso ao meio, complementação das tarefas de *bridging*, endereçamento, encaminhamento e aplicação de políticas de QoS (balanceamento de carga).

3.3.1.5 Control-Station

A entidade *Control-Station* (CS) é fundamental nas tarefas de pós-processamento, na apresentação de dados e também nas tarefas de reconfiguração da RSSF. Para além disso, a CS possui capacidades para compilar a informação útil à aplicação e posteriormente enviar esses dados já compilados a várias entidades centrais que possam existir.

As particularidades que definem a CS são apresentadas de seguida:

- **Funções Base**

Tal como referido anteriormente, a CS comunica diretamente com o SG, sendo este último a interface de comunicação com a RSSF. Os dados encaminhados pela RSSF, com destino ao SG, terão como destino final a CS.

Uma vez que o poder de processamento da CS é tipicamente elevado, poderão ser realizadas tarefas computacionais mais exigentes, efetuando, por exemplo, cálculos matemáticos sobre os dados recolhidos da RSSF. A comunicação com o SG poderá ser realizada usando apenas endereçamento IP, caso o SG possua as capacidades de mapeamento referidas anteriormente.

A CS é uma entidade local, geograficamente próxima da RSSF, podendo enviar os dados recolhidos e calculados para uma estação de controlo central, localizada, por exemplo, na Internet. Outra importante tarefa da entidade CS será a capacidade para efetuar consultas e disseminação de dados.

Resumindo, as funções base da entidade CS são:

- Manutenção e visualização de dados estatísticos/estado da RSSF, como por exemplo, o número de elementos ativos na rede e suas particularidades (*uptime*, estatísticas de transmissão de mensagens, etc.);

- Consultas à RSSF, onde poderão ser recolhidos dados de configuração, dados sensoriais, entre outros. Consulta de valores a um elemento específico ou a todos os elementos que fazem parte de um dado *cluster*;

- Disseminação de dados, particularmente dados de parametrização, ativação/desativação de funcionalidades e dados de reconfiguração pelos vários elementos da RSSF, individualmente ou por *cluster*;

- Comunicação com a estação de controlo central. Envio dos dados recolhidos pela RSSF, por exemplo, para fins de *Data Warehousing*. Receção de parâmetros enviados pela estação central, efetuando, por exemplo, a ativação/desativação de funcionalidades;

- Interligação com outro tipo de aplicações, como por exemplo, aplicações de alarmística.

➢ **Ligação a Outras Redes e Serviços**

A CS pode tirar partido das suas capacidades de processamento e expansibilidade para interligar a RSSF com outras redes, podendo até interligar várias RSSF. O principal exemplo de ligação a outras redes poderá ser a ligação à estação de controlo central, na Internet. Existindo a capacidade de interligação a outras redes, onde se destaca a Internet, o uso de vários tipos de serviço são uma possibilidade, como por exemplo, comunicação com serviços de emergência.

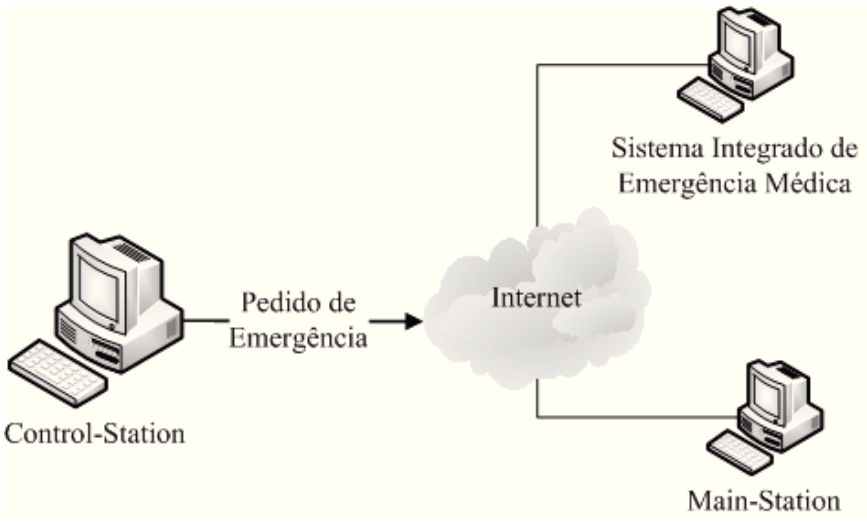

Figura 25 - Ligação da Control-Station a Outras Redes e Serviços

➢ **Pilha-Protocolar**

Na Figura 26 pode ser vista a pilha protocolar de uma CS, definindo as componentes presentes nas várias camadas da pilha.

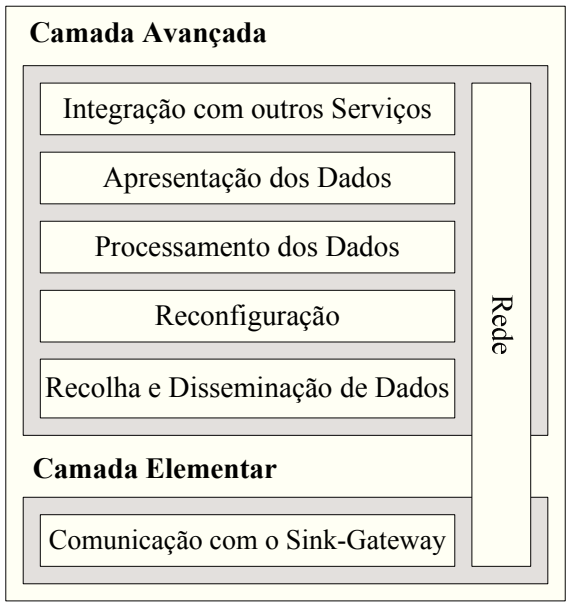

Figura 26 - Control-Station - Pilha Protocolar

- **Camada Avançada**

  - **Integração com outros Serviços:** Capacidade de interligações a uma ou mais *Main-Station* (MS) ou a serviços de terceiros, como por exemplo, serviços de alarmística;

  - **Apresentação dos Dados:** Responsável pela apresentação dos dados numa forma legível. Dependendo da aplicação da RSSF pode ser necessária a aplicar de conversões/adaptações aos dados recolhidos para posterior apresentação na aplicação da CS;

  - **Processamento dos Dados:** Tratamento dos dados recolhidos, filtrando e agregando os dados gerados na RSSF. Dada a maior capacidade de processamento da CS, podem ser aplicadas tarefas de agregação e filtragem mais exigentes;

  - **Reconfiguração:** Fornece a interface e comunicação necessárias para a redefinição de parâmetros aplicacionais ou de rede, podendo assim a aplicação da RSSF ser customizada e adaptada a determinadas particularidades;

  - **Coleção e Disseminação de Dados:** Execução de tarefas de

disseminação de novos parâmetros aplicacionais/configuração e tarefas de coleção de dados, como por exemplo, consulta a um determinado nó ou *cluster*.

- **Camada Elementar**

  - **Comunicação com o Sink-Gateway:** Receção e envio de mensagens através da interface de comunicação com o/os SG. Esta interface poderá por exemplo ser baseada em IEEE 802.3.

- **Componentes Transversais**

  - **Rede:** Disponibilização de uma pilha-protocolar TCP/IP para comunicação com a MS ou outros serviços. Manutenção do estado da RSSF, podendo ou não recorrer ao mapeamento do esquema de endereços usados nos segmentos de rede na RSSF e na CS. Uma vez que a CS mantem o estado da rede, pode recorrer às estatísticas estado da rede para aplicar políticas de QoS, encaminhado o tráfego através de um SG ou CH menos congestionado por exemplo.

3.3.1.6 Main-Station

A entidade *Main-Station* (MS), apesar de não ser uma entidade essencial para o funcionamento de uma dada aplicação, poderá ser utilizada em aplicações que recorrem a várias CS, sendo assim um ponto central entre as várias CS.

As particularidades que definem a MS são apresentadas de seguida.

- **Funções Base**

Sendo um ponto central entre as CS distribuídas por diversos locais, será importante a MS estar disponível através de uma rede comum e facilmente acessível por parte das CS. Essa rede é a Internet. Os serviços disponibilizados pela MS podem ir desde a disponibilização de bases de dados até serviços de *"Authentication, Authorization and Accounting"* (AAA).

- **Disponibilização de Serviços**

A implementação dos serviços disponibilizados pode tirar partido das diversas ferramentas

utilizadas nas típicas aplicações para *Desktops* ou *Smartphones*, onde se destacam os *Web-Services*.

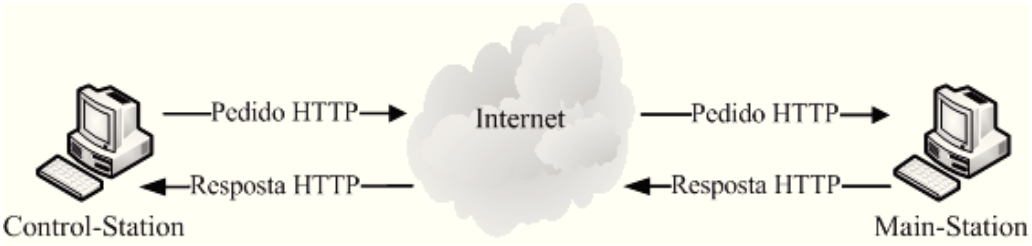

Figura 27 - Comunicação entre a CS e MS

- **Controlo e Agregação de várias Control-Station**

Um dos possíveis serviços disponibilizados pela MS poderá ser a capacidade de agregação dos dados recolhidos nas RSSF pelas quais as várias CS são responsáveis. Os dados recolhidos pela MS poderão ser agregados, calculados e guardados de forma permanente. Uma vez que as várias CS possuem capacidades de comunicação constante com a MS, será possível verificar o estado de cada CS e consequentemente das suas RSSF, podendo assim controlar diversos parâmetros das RSSF, bem como determinar qual o estado dos vários elementos participantes na solução.

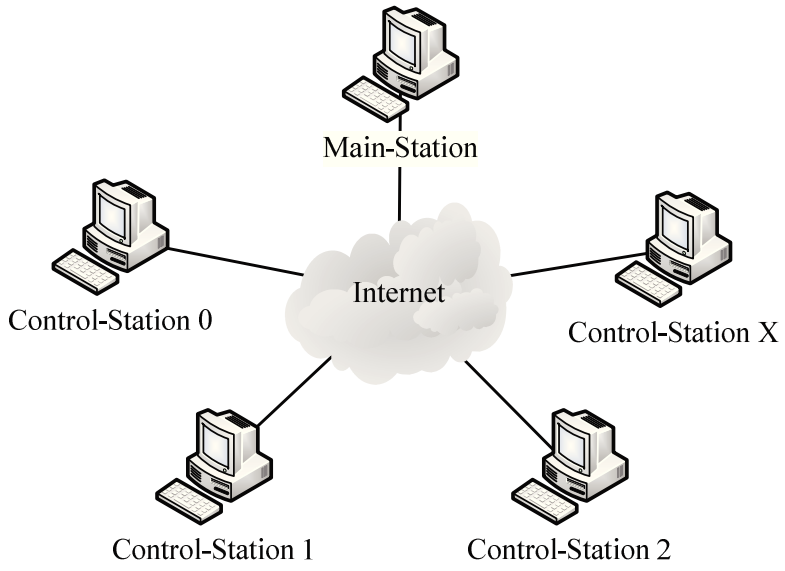

Figura 28 - Interligação de várias Control-Station

- **Pilha-Protocolar**

Na Figura 29 pode ser vista a pilha protocolar de uma MS, definindo as componentes presentes nas várias camadas da pilha.

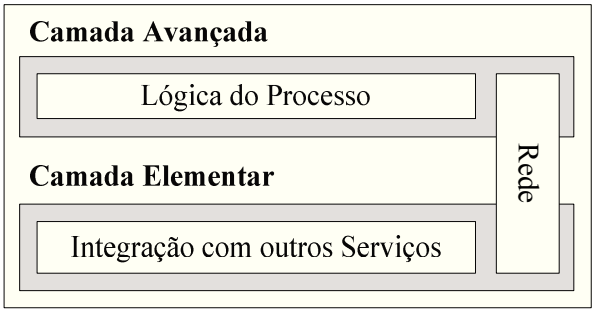

Figura 29 - Main-Station - Pilha Protocolar

- **Camada Avançada**
  - **Lógica do Processo:** Dependendo do tipo de dados recolhidos pela RSSF será aplicada a lógica necessária para posterior tratamento desses dados. Esta lógica de processo pode, por exemplo, guardar os dados recolhidos para efeitos estatísticos e posterior análise ou executar ações de suporte como, por exemplo, comunicar com serviços de emergência.
- **Camada Elementar**
  - **Integração com outros Serviços:** Capacidade de interligações a uma ou mais *Control-Station* (CS).
- **Componentes Transversais**
  - **Rede:** Disponibilização de uma pilha-protocolar TCP/IP para comunicação com as CS ou outros serviços.

3.3.2 Transporte e QoS

Após definida a topologia, capacidades e funções de cada elemento da solução, é importante aplicar as técnicas de transporte e QoS adequadas às necessidades da aplicação. Tal como nas típicas redes de computadores, as técnicas usadas em RSSF dependem do teor da aplicação,

diferenciando, por exemplo, aplicações onde o tempo de resposta é determinante de aplicações onde a integridade dos dados é fundamental.

Seguidamente serão apresentadas várias características encontradas em RSSF com influência direta no desenho dos protocolos de rede/transporte e na implementação de políticas de QoS.

3.3.2.1 Agregação de Dados

Mediante o tipo de mensagens, o CH poderá agregar as mensagens recebidas dos vários CH e dos CM presentes no seu *cluster*, fazendo uma pré-análise dos dados recebidos, removendo possíveis duplicações e posteriormente encaminhar até ao SG as várias mensagens agregadas. Esta técnica, quando aplicada corretamente, irá reduzir o *overhead* resultante da troca de mensagens. A agregação de dados deve ser analisada e aplicada por uma camada de *software* dedicada, olhando ao tipo de dados, à duplicação de informação, à origem/destino das mensagens, entre outras características. Esta camada de agregação poderá, por exemplo, aplicar, quando possível, cálculos matemáticos de maneira a encontrar a média de um valor recolhido de um vasto conjunto de CM, fazendo com que se reduza a informação de milhares de valores para apenas um. A lógica de agregação é dependente da aplicação, devendo ser desenhada de acordo com os dados recolhidos e gerados pela RSSF. Os protocolos de rede e transporte devem levar em conta este fluxo de tráfego de forma a otimizar o uso da rede.

A Figura 30 mostra um fluxo típico de mensagens e a agregação efetuada pelos vários CH.

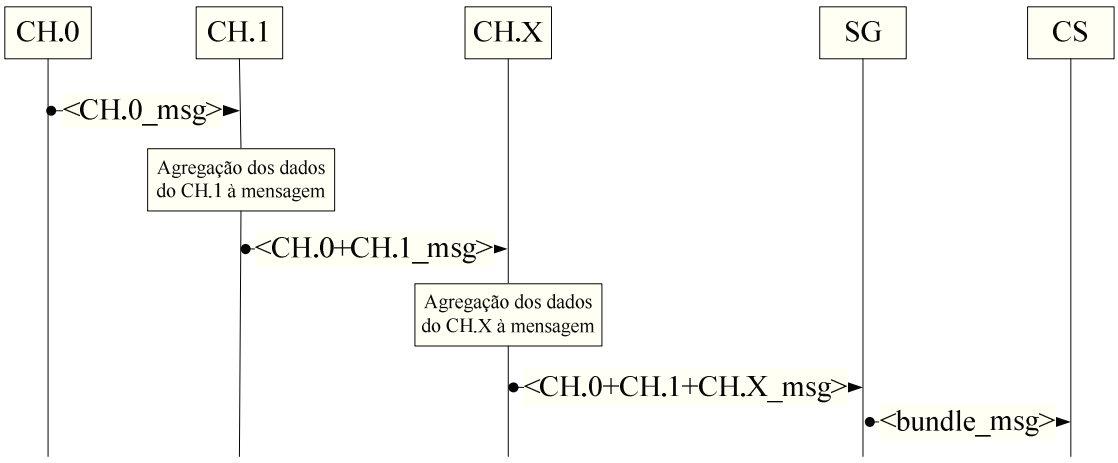

Figura 30 - Agregação de dados

3.3.2.2 Disseminação de Dados

Tal como a agregação, a disseminação de dados é também importante para várias aplicações usadas em RSSF. Contudo, a disseminação de dados pode não ser usada em aplicações que não necessitam, por exemplo, de recolher informação *ondemand* ou de reconfigurar parâmetros de execução.

A teoria de agregação pode ser aplicada de forma inversa neste tipo de fluxo de dados, fazendo com que, à medida que os dados disseminados são aplicados a um dado elemento ou conjunto de elementos, se suspenda a propagação das mensagens para outros elementos, evitando assim a transmissão de conteúdo desnecessário.

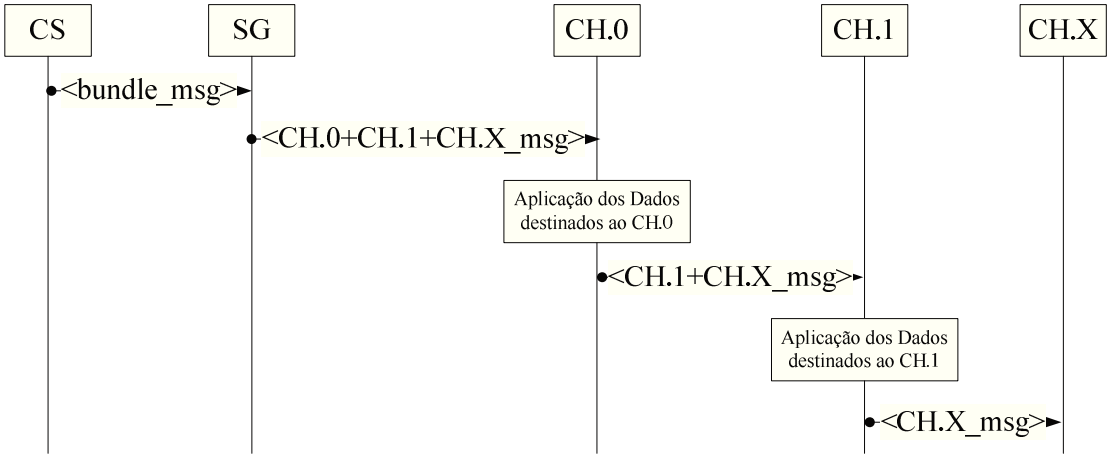

Figura 31 - Disseminação de Dados

- **Reconfiguração**

A solução proposta é baseada numa abordagem *single-app*, contudo, a possibilidade de reconfiguração da aplicação permite a flexibilização de determinadas características acabando por permitir uma maior longevidade no tempo de vida da RSSF.

A reconfiguração e customização da aplicação são possíveis recorrendo à disseminação de dados pelos elementos que constituem a RSSF. Vários parâmetros podem ser redefinidos sem que seja necessária a paragem de um dado elemento para atualização com um novo *firmware*. A inteligência necessária para a reconfiguração deve ser aplicada por uma camada aplicacional, recorrendo sempre que possível a dados/configurações genéricas.

A atualização de dados de parametrização, como por exemplo, variáveis aplicacionais ou

características de rede, é possível recorrendo à disseminação de novos parâmetros de execução. Estes dados podem ser distribuídos seguindo os sistemas *unicast*, *multicast* ou *broadcast*.

A consulta e redefinição destes valores é efetuada recorrendo à CS, onde é possível consultar os valores em uso por um determinado elemento, bem como, redefinir um novo valor para um dado parâmetro.

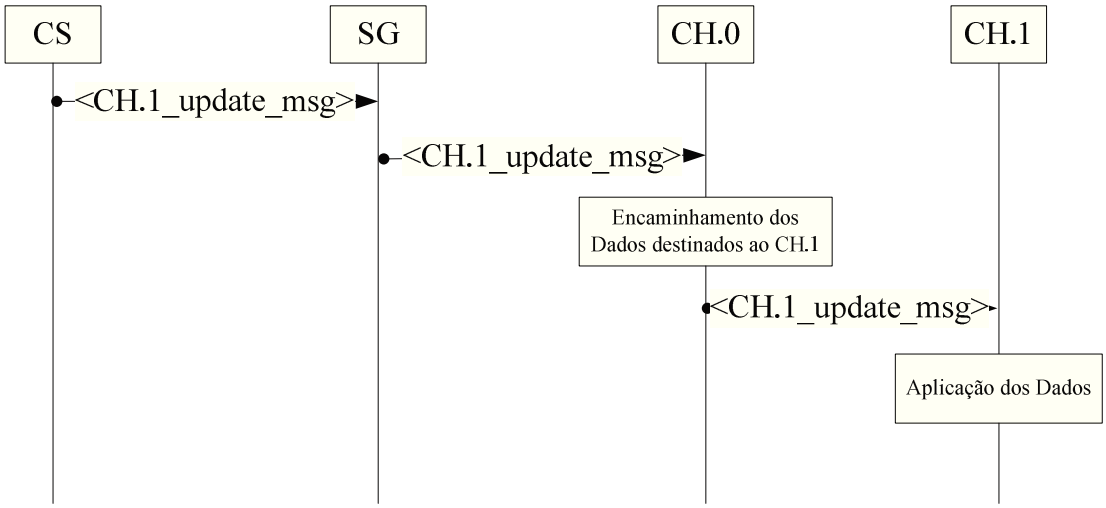

Figura 32 - Disseminação de Dados de Configuração

3.3.2.3 Consultas

Recorrendo à agregação e disseminação de dados poderão ser efetuadas consultas complexas à RSSF, efetuando, por exemplo, uma consulta onde é especificado qual o elemento ou CH e quais os dados que se pretendem consultar. Esta capacidade permite à CS a utilização de inteligência no tratamento dos dados recebidos, maximizando a eficácia da RSSF. Dando como exemplo um caso onde um dos CM deteta um valor alarmante e comunica esse valor à CS. Poderá fazer sentido a consulta de determinados valores atuais a todos os CM que constituem o *cluster*, ou mesmo de CM associados a outro *cluster*. Desta forma, a carga de processamento e análise fica a cargo da CS que dispõe de maior poder computacional, libertando assim os limitados elementos da RSSF deste tipo de tarefas.

3.3.2.4 Filas Prioritárias

Tal como nas redes de computadores convencionais, o uso de técnicas de QoS são também importantes em RSSF. De maneira a otimizar a eficiência da rede e como consequência,

otimizar o tempo de vida da mesma, as mensagens enviadas pelos CM e CH devem obedecer a um escalão de prioridades. Desta forma, podem ser definidas, por exemplo, mensagens críticas e não críticas, facilitando a agregação de mensagens e otimizando o uso do canal de transmissão. No caso de mensagens onde a rapidez de entrega é fundamental, pode ser usada uma fila de mensagens prioritárias, fazendo com que estas sejam encaminhadas para a CS logo que possível. No caso de mensagens não críticas, o CM pode obedecer aos ciclos de transmissão definidos e os CH podem agregar a informação, obedecendo também a ciclos de transmissão.

O tratamento distintivo das mensagens enviadas provocará um aumento na eficiência da transmissão de dados, garantindo a rápida entrega de dados críticos e a agregação de dados não críticos.

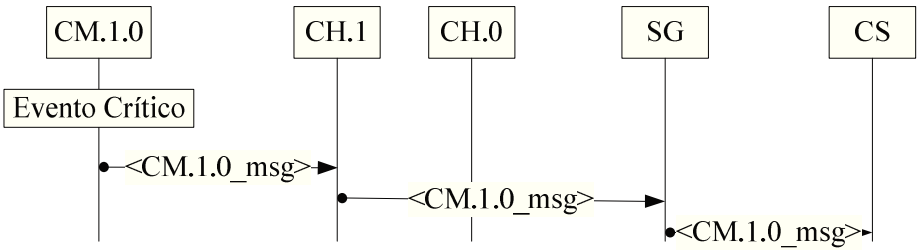

Figura 33 - Envio de Mensagem Crítica

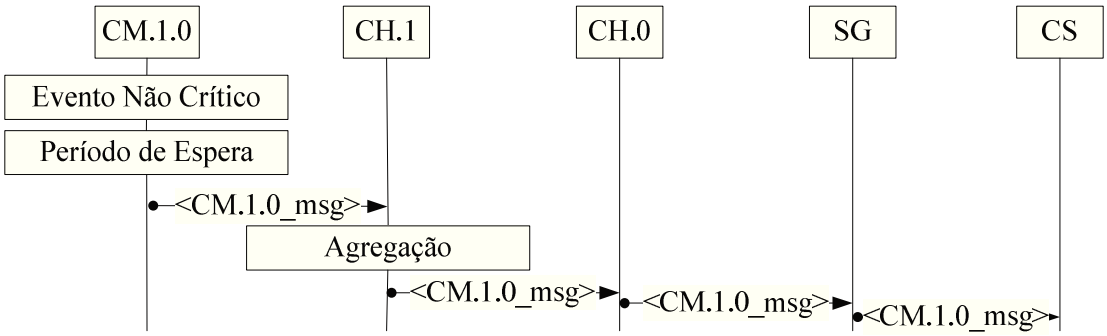

Figura 34 - Envio de Mensagem Não Crítica

3.3.3 Encaminhamento

Os CH presentes na RSSF serão responsáveis pela definição das rotas para o envio de mensagens e pela gestão do *cluster*, podendo estes comunicar diretamente com o SG ou fazer chegar as mensagens através de outros CH. Vários protocolos de encaminhamento e de

gestão de *cluster* têm sido propostos para RSSF baseadas em topologias *cluster single-hop* e/ou *multi-hop*, onde se destacam:

- The Adaptive Threshold sensitive Energy Efficient sensor Network protocol (APTEEN) [123];
- Algorithm for Cluster Establishment (ACE) [124] ;
- Base-Station Controlled Dynamic Clustering Protocol (BCDCP) [85];
- Concentric Clustering Scheme (CCS) [125];
- Distributed Weight-based Energy-efficient Hierarchical Clustering protocol (DWEHC) [103];
- Energy Efficient Clustering Scheme (EECS) [77];
- Energy-Efficient Uneven Clustering (EEUC) algorithm [126];
- Hybrid Energy-Efficient Distributed clustering (HEED) [102];
- Hierarchical Geographic Multicast Routing (HGMR) [127];
- Low-energy Adaptive Clustering Hierarchy (LEACH) [84];
- Position-based Aggregator Node Election protocol (PANEL) [128];
- Power-Efficient Gathering in Sensor Information Systems (PEGASIS) [86];
- Energy Efficient sensor Network protocol (TEEN) [82];
- Two-Level Hierarchy LEACH (TL-LEACH) [129];
- Two-Tier Data Dissemination (TTDD) [130];
- Unequal Clustering Size (UCS) model [131].

3.3.3.1 Equilíbrio no Transporte

Outra importante técnica de otimização que pode ser aplicada pelos protocolos de rede e transporte é a distribuição balanceada do tráfego pelos CH disponíveis. A maioria dos protocolos baseia-se na qualidade da ligação para seleção e criação de rotas. Apesar de esta seleção ser a mais adequada à primeira vista, pode facilmente levar ao rápido desgaste dos elementos presentes na rota preferencial. Tendo como exemplo um CH que dispõe de duas rotas (duas opções para próximo salto) para fazer chegar as suas mensagens à CS, mas onde uma dessas rotas dispõe de uma qualidade de ligação superior à outra, o protocolo de rede e transporte deverá estar consciente de que, apesar de a segunda rota apresentar uma ligação de menor qualidade, será importante balancear o tráfego entre as duas possíveis rotas, usando o algoritmo de *Round-Robin* por exemplo. Desta forma, o desgaste energético da RSSF será mais uniforme.

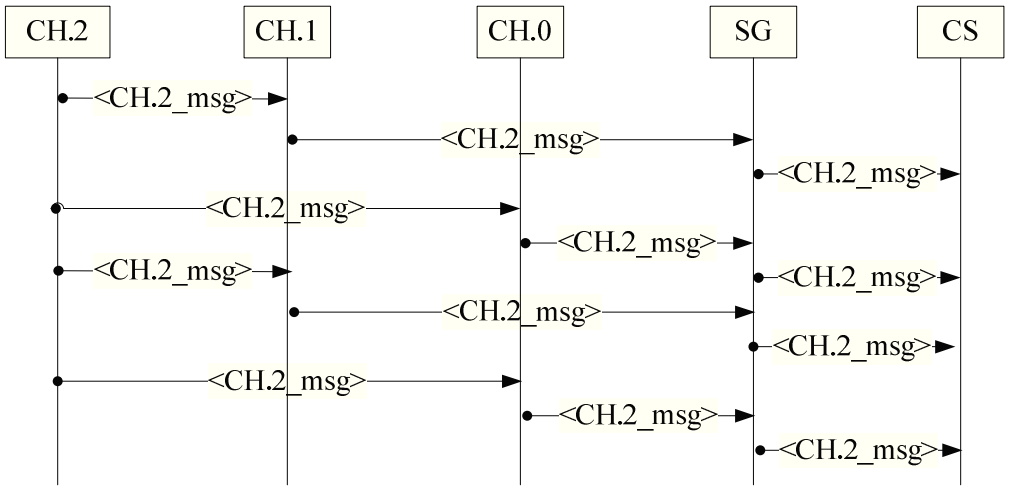

Figura 35 - Balanceamento de Carga

3.3.3.2 Endereçamento

O esquema de endereçamento utilizado depende do protocolo de rede/transporte utilizado. Os esquemas de endereçamento utilizados em RSSF baseadas em *cluster* podem seguir a abordagem de endereçamento hierárquico, onde cada CH é o "pai" da sub-rede e os CM que fazem parte do *cluster* serão "filhos" dessa mesma sub-rede. A abordagem hierárquica facilita a criação das árvores de *clustering* que representam a RSSF.

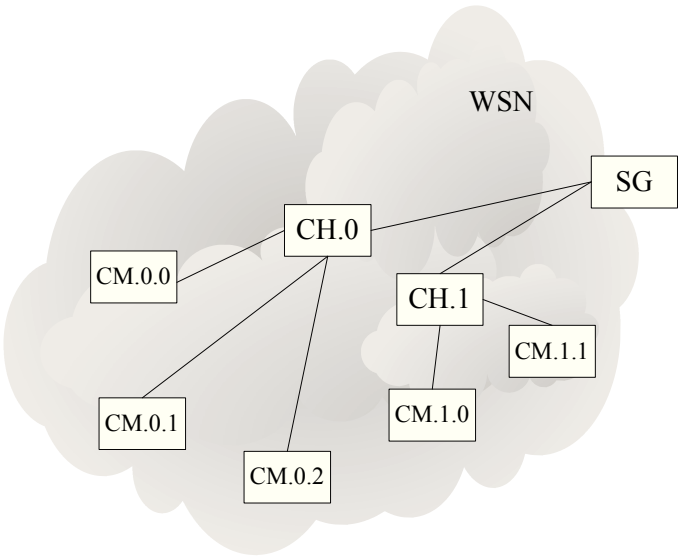

Figura 36 - Endereçamento Hierárquico

O uso de endereços sem olhar à hierarquia é também uma possibilidade, contudo apresentam a desvantagem de não representar diretamente quais os CM associados a um dado CH,

exigindo que essa associação seja feita recorrendo a mecanismos adicionais.

3.3.4 Segurança

Geralmente, as RSSF implementadas em ambientes abertos, onde os vários elementos não dispõem de proteção física, tornam-se vulneráveis a uma variedade de ataques maliciosos [132][133]. Os mecanismos de segurança convencionais não são diretamente aplicáveis às RSSF, uma vez que não consideram as restrições apresentadas pelas RSSF.

Para proteger um dado elemento de ataques, é fundamental conceber mecanismos de segurança eficazes ou protocolos que considerem tanto a eficiência energética como a segurança. O controlo do acesso, impedindo elementos ilegítimos de se associarem e participar nas tarefas da RSSF, pode preservar o correto funcionamento da RSSF na maioria dos cenários.

Os mecanismos criptográficos baseados na distribuição de chaves são as soluções mais comuns para garantir o controlo no acesso à RSSF. No entanto, pode não ser possível aplicar diretamente os sistemas de distribuição de chaves mais comuns na literatura a uma RSSF devido às seguintes consequências [134]:

- Exigência de elevadas capacidades processamento/memória para o tratamento de técnicas baseadas em chaves públicas;
- Vulnerabilidades de segurança com esquemas de codificação globais;
- Elevado consumo energético e ineficiência causados por uma distribuição de chaves centralizada.

Técnicas para proteger uma RSSF, com a pré-distribuição de chaves aleatórias, têm sido intensamente estudadas no contexto das RSSF baseadas numa arquitetura *flat* [135][136][137].

A maior parte dos esquemas de distribuição de chaves propostos na literatura assumem que cada elemento apenas interage com um conjunto estático de elementos vizinhos, predefinidos na implantação, o que torna a implementação não adequada em RSSF baseadas em *cluster*.

Numa RSSF baseada em *cluster*, os elementos CH são os alvos mais importantes para os

ataques, uma vez que são estes elementos que efetuam a agregação e encaminhamento de mensagens. Um CH comprometido pode ser usado para atacar a RSSF recorrendo a encaminhamento seletivo, criando o que se chama de *Sink-Hole* [138] ou propagando informação prejudicial para os outros elementos da RSSF.

Os algoritmos de *clustering* baseados na seleção rotativa de CH, como o *"Low Energy Adaptive Clustering Hierarchy"* (LEACH), tornam a identificação e posterior comprometimento dos CH uma tarefa mais difícil [132].

3.3.5 Eficiência energética

A aplicação das várias técnicas apresentadas anteriormente leva à otimização do consumo energético da RSSF acabando por prolongar o tempo de vida da RSSF.

Recorrendo à filtragem e agregação de dados por parte dos CH, minimizar-se-á a transmissão de dados não úteis à aplicação.

Tirando partido das técnicas de encaminhamento e transporte podem ser aplicados mecanismos que possibilitam a distribuição de carga pelos vários CH, evitando o rápido desgaste dos elementos presentes na rota preferencial.

Para além disso, dependendo da aplicação, os estados de *active/idle* e a definição de filas de prioridades podem otimizar significativamente o consumo energético de toda RSSF.

3.4 Análise da Solução Proposta

Com a aplicação das métricas de avaliação ao modelo conceptual definido, foram especificadas as entidades e funcionalidades necessárias para atingir os objetivos da solução.

Com recurso à topologia Cluster a escalabilidade da rede é flexível, permitindo a adição de elementos CM causando o menor impacto possível na rede. A recolha de dados sensoriais e posterior entrega desses dados ao elemento CH permite a aplicação de técnicas de agregação, possibilitando a otimização no uso do meio de transmissão e consequentemente a obtenção de uma melhor eficácia na comunicação e na gestão do consumo energético.

Ao tirar partido da entidade CS, além das possibilidades de reconfiguração dos elementos da RSSF, é também possível refinar os dados recolhidos e interligar a RSSF com outros tipos de

serviço através do uso da entidade MS, permitindo assim uma elevada aplicabilidade dos dados recolhidos.

3.5 Síntese

Neste capítulo foram delimitados os requisitos da solução e avaliadas as características que poderão dar resposta a esses mesmos requisitos. Tendo por base a apreciação realizada através das métricas de avaliação, foram identificadas as entidades e funcionalidades necessárias para a implementação da arquitetura proposta.

Além da definição das entidades da arquitetura foram apresentadas técnicas que poderão otimizar a performance da solução, onde se destacam o uso de filas prioritárias e o balanceamento de carga pelos vários elementos da rede.

O capítulo foi finalizado com a avaliação da solução proposta, onde se conclui que recorrendo às capacidades oferecidas pelas várias entidades se pode usufruir de uma arquitetura genérica e escalável, permitindo a aplicação de técnicas de agregação e ainda a interligação da RSSF a outros serviços.

4 Implementação

De maneira a testar a implementação e integração das várias entidades propostas, foi desenvolvido um protótipo que demonstra algumas das possíveis funcionalidades para cada uma das entidades.

Seguidamente são enumerados os requisitos do protótipo e alguns pressupostos que influenciaram o seu desenvolvimento. Para além dos requisitos são também apresentados o *hardware*, *software* e outras ferramentas utilizadas.

4.1 Requisitos do Protótipo

O principal objetivo do protótipo é testar a comunicação entre as várias entidades e o desenvolvimento de um conjunto de *software* que possa ser usado como base em projetos futuros. Nesse sentido, foram estabelecidos os seguintes pressupostos:

- Uso de *hardware* disponível para testes;
- Uso de *software open-source* sempre que possível;
- Comunicação entre as várias entidades;
- Poupança de energia.

4.2 Hardware Utilizado

Uma vez que um dos requisitos para o desenvolvimento é a utilização de *hardware* disponível para testes na Escola Superior de Tecnologia e Gestão (ESTG), as escolhas realizadas levaram em conta a disponibilidade de equipamento.

De seguida é apresentado o *hardware* utilizado para desempenhar as funções das várias entidades da solução proposta.

4.2.1 Cluster-Member e Cluster-Head

Para desempenhar as funções de Cluster-Member (CM) e Cluster-Head (CH), foram considerados três tipos de nós, sendo eles o Mica2, MicaZ e iMote2. Tendo em conta estas possibilidades a escolha recaiu sobre o MicaZ uma vez que este, tal como o Mica2, apresenta compatibilidade com uma maior variedade de sistemas operativos quando comparado com o iMote2 e fornece um *chip* rádio compatível com IEEE 802.15.4. Um dos fatores limitativos para o desenvolvimento foi o facto de existirem apenas dois MicaZ disponíveis para utilização.

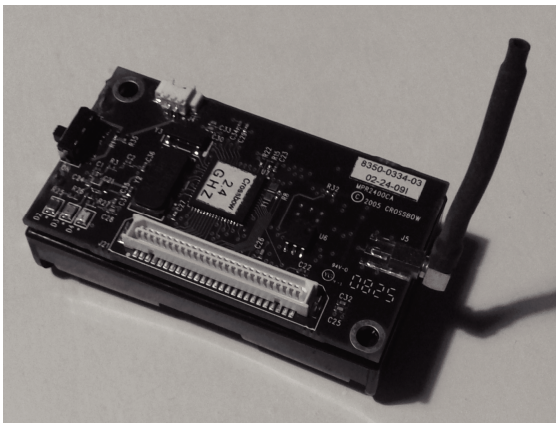

Figura 37 - MicaZ usado no Protótipo

4.2.2 Placa-Sensorial

Sendo usados MicaZ, a escolha da placa sensorial é limitada, já que apenas existem duas séries compatíveis, aMTS300/310 e a MTS400/420. Estando a placa MTS310 disponível para utilização, foi esta a placa sensorial utilizada para recolha de dados sensoriais.

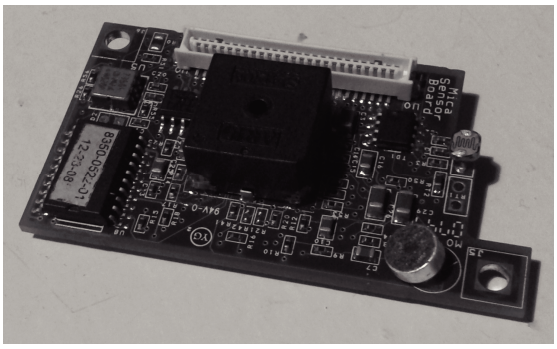

Figura 38 - Placa MTS310 usada no Protótipo

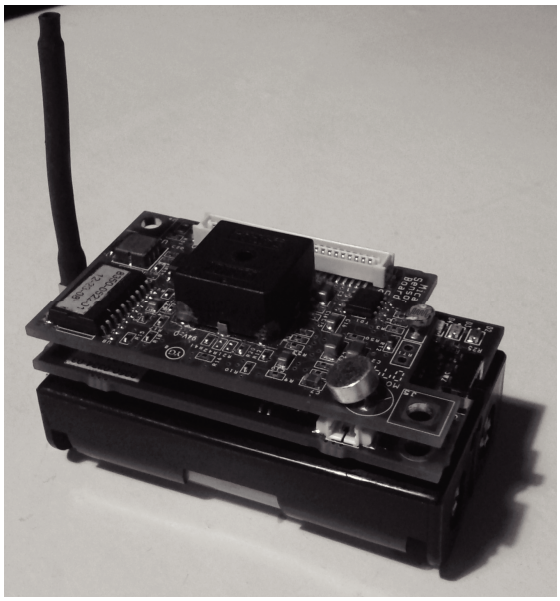

Figura 39 - MicaZ/MTS310 usados no Protótipo

4.2.3 Sink-Gateway

Para interface de ligação da Rede Sensorial Sem Fios (RSSF) à Control-Station (CS) as escolhas possíveis para utilização são as placas MIB510 e MIB600 [139]. Quando comparada com a MIB510, a placa MIB600 tem a vantagem de permitir conectividade TCP/IP sobre Ethernet.

De maneira a fazer a *bridge* entre a RSSF (Radio Frequência) e a CS (Ethernet) existe a necessidade de usar um dos MicaZ em conjunto com a placa MIB600. Visto que existem apenas dois MicaZ disponíveis, existe a necessidade de simular as tarefas das entidades CM e CH recorrendo apenas a um MicaZ.

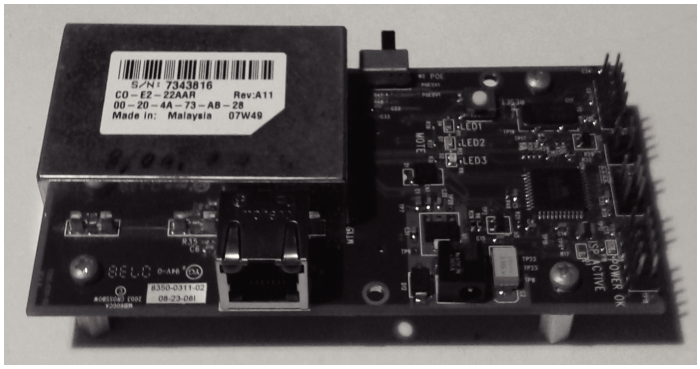

Figura 40 - MIB600 usada no Protótipo

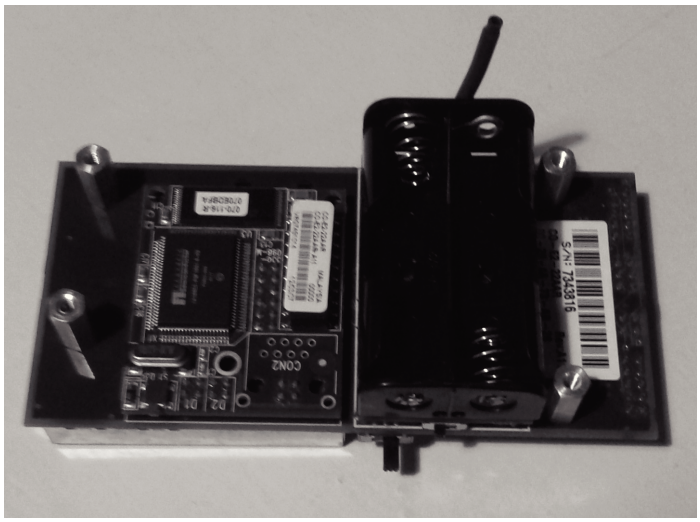

Figura 41 - MIB600/MicaZ usados no Protótipo

4.2.4 Control-Station

Para desempenhar as funções de CS foi usado um normal Personal Computer (PC) com capacidades para executar um sistema operativo (SO) moderno, com interface Ethernet para comunicação com o Sink-Gateway (SG) e ligação à Internet para permitir a comunicação com a entidade Main-Station (MS).

4.2.5 Main-Station

A entidade MS será um elemento na Internet que disponibilizará uma variedade de serviços. No caso deste protótipo, para efeitos meramente demonstrativos, o *hardware* necessário para representar os serviços da MS é fornecido gratuitamente por terceiros.

4.3 Software Utilizado

Um dos aspetos importantes para se dar inicio ao desenvolvimento do protótipo foi a escolha do SO a ser usado pelos elementos da RSSF (CM, CH e SG). Analisando os requisitos, levando em conta o uso de *software open-source* e a preocupação com as questões de poupança de energia, o TinyOS foi o SO escolhido para desenvolvimento das funcionalidades apresentadas.

No que diz respeito ao SO usado nos PC responsáveis pela execução das funcionalidades da CS, foi feito um esforço para manter a aplicação compatível com as plataformas Linux e

Windows. Tendo a portabilidade como um fator de decisão, a escolha recaiu sobre a linguagem Java.

4.4 Visão Global do Protótipo

Na Figura 42 é apresentado o cenário de implementação do protótipo, onde todas as componentes, em conjunto, permitirão a recolha de dados e posterior envio de alertas até à MS.

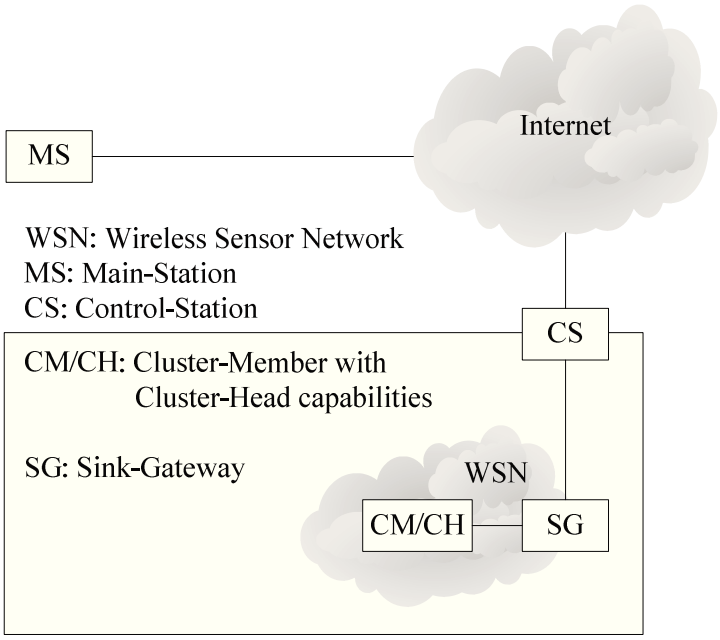

Figura 42 - Arquitetura Geral do Protótipo

A arquitetura do protótipo é constituída pelos seguintes elementos:

- O elemento CM/CH é representado pelo conjunto MicaZ/MTS310. Tem como principais tarefas a coleção e análise de dados sensoriais, geração de alertas e comunicação com o elemento SG;
- O elemento SG é representado pelo conjunto MIB600/MicaZ, efetuando as tarefas necessárias para a comunicação entre a RSSF e a CS;
- O elemento CS é representado por um PC. Através da comunicação com o SG pode receber e enviar mensagens para a RSSF e através da Internet comunicar com a MS;
- O elemento MS é representado por um serviço disponível na Internet. Neste caso é

usado um servidor de *email*.

Na Figura 43 pode ver-se o diagrama de sequência de mensagens caso um alerta seja detetado:

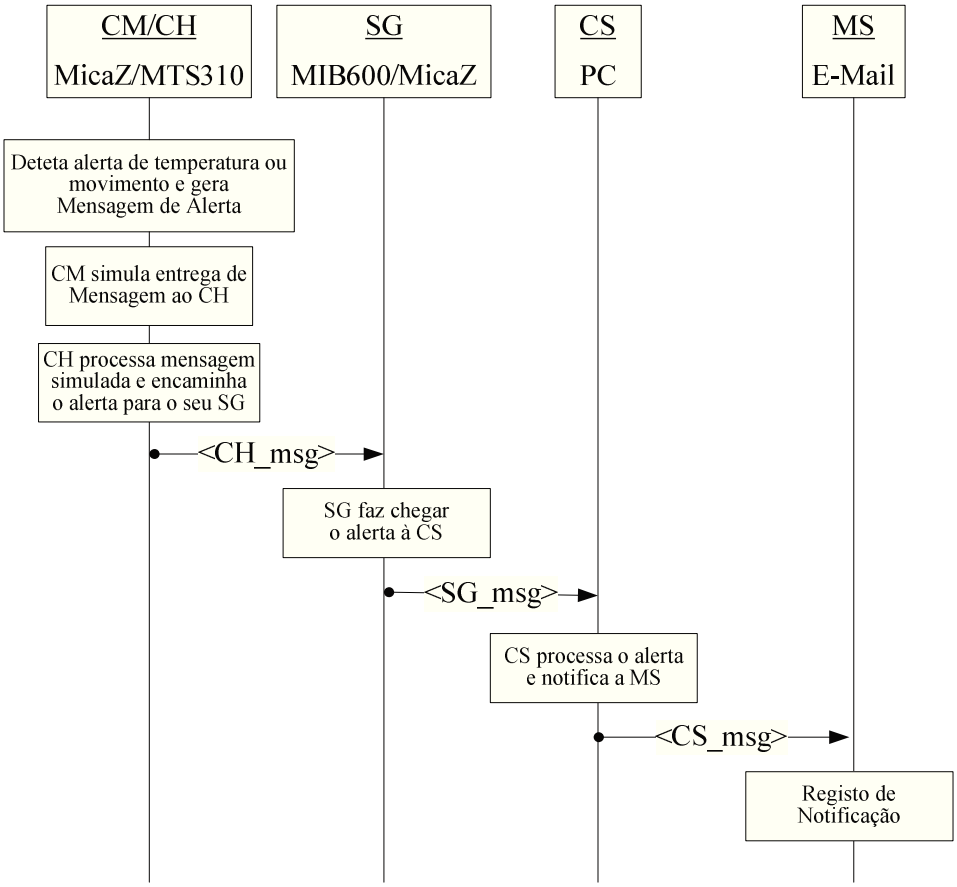

Figura 43 - Fluxo de mensagens de um Alerta no Protótipo

4.4.1 Aplicações da Rede Sensorial Sem Fios

Após a definição do *hardware* e *software* a usar no protótipo deu-se início ao desenvolvimento das três aplicações necessárias (CM/CH, SG e CS), cada uma delas com funcionalidades específicas.

Em conjunto, as três aplicações permitem a aquisição de dados referentes ao movimento e temperatura, fazendo chegar a informação a partir do CM/CH até à CS através do SG.

De seguida serão analisadas as funcionalidades desenvolvidas para cada aplicação.

4.4.1.1 Aplicação Cluster-Member e Cluster-Head

Tal como referido anteriormente, visto que apenas um dos MicaZ está disponível para executar as tarefas de CM e CH, a aplicação desenvolvida representará um CM com funcionalidade de CH, efetuando aquisição de dados sensoriais e comunicação com o SG.

Uma vez que a escolha de SO para os elementos da RSSF foi o TinyOS, o desenvolvimento das aplicações para os elementos CM/CH e SG foi realizado recorrendo à linguagem nesC. A aplicação desenvolvida para os elementos CM/CH é, a par da aplicação desenvolvida para a CS, a mais complexa e onde foi investido mais tempo de trabalho. Esta aplicação tem a importante função de recolher os dados sensoriais (placa sensorial MTS310), processá-los, executar tarefas de agregação e proceder ao envio das mensagens.

Em resumo, as funcionalidades presentes na aplicação CM/CH são as seguintes:

- Recolha de dados sensoriais relacionados com o movimento e temperatura;
- Envio e Receção de Mensagens;
- Agregação de Mensagens;
- Avaliação dos dados sensoriais recolhidos e geração de Alertas;
- Capacidade para atualização de parâmetros aplicacionais sem que a paragem da execução seja necessária;
- Simulação de eventos de associação e dissociação ao *cluster*;
- Simulação de ocorrência de alertas críticos e não críticos.

Tendo em conta as funcionalidades apresentadas, as mensagens processadas pelo elemento CM/CH são as seguintes:

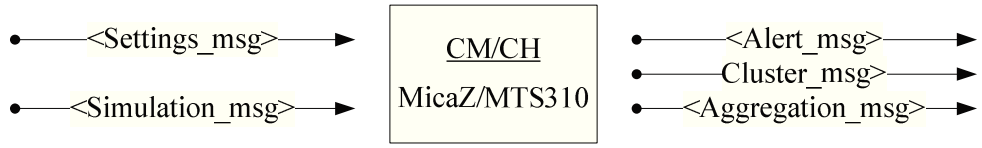

Figura 44 - Mensagens processadas pelo elemento CM/CH

Para implementar todas as funcionalidades apresentadas houve a necessidade de estudar

muitas das capacidades que o TinyOS oferece, tentando assim tirar o máximo partido dos limitados recursos oferecidos pelo MicaZ.

Seguidamente podem ser vistas as interfaces TinyOS usadas e as tarefas desenvolvidas para a aplicação CM/CH.

Interface	Função
Boot as Boot	Iniciar as componentes do TinyOS
Leds as Leds	Interagir com os três LED disponíveis
Timer<TMilli> as MainLoop	Definir o ciclo interno para recolha de dados
Read<uint16_t> as Temp	Leitura dos dados sensoriais referentes à temperatura
ReadStream<uint16_t> as Accel	Leitura dos dados sensoriais referentes ao movimento
Mts300Sounder as Sound	Interagir com o *buzzer* da placa MTS310
SplitControl as RadioControl	Iniciar as componentes de radiofrequência
AMSend as AlertRadioSend AMSend as AggregationRadioSend AMSend as SimulationRadioSend AMSend as ClusterRadioSend AMSend as SettingsRadioSend	Disponibilização dos métodos e estruturas necessárias para o envio de mensagens através de radiofrequência
Receive as AlertRadioReceive Receive as AggregationRadioReceive Receive as SimulationRadioReceive Receive as ClusterRadioReceive Receive as SettingsRadioReceive	Disponibilização dos métodos e estruturas necessárias para a receção de mensagens através de radiofrequência

Tabela 6 - Interfaces TinyOS usadas na aplicação CM/CH

Seguidamente são apresentadas as tarefas e funções desenvolvidas para que a aplicação CM/CH possa efetuar as funcionalidades indicadas previamente.

-	F: Nome da Função; D: Descrição da Função;
F:	`void amberLed(bool on)`
D:	Ativar/Desativar Led laranja
F:	`void greenLed(bool on)`

D:	Ativar/Desativar Led verde
F:	`void redLed(bool on)`
D:	Ativar/Desativar Led vermelho
F:	`void check(error_t ok)`
D:	Ativar Led vermelho em caso de erro
F:	`void beep(uint16_t time_ms)`
D:	Ativar *buzzer*
F:	`void initClusterHead()`
D:	Inicializar variáveis de controlo do CH
F:	`void initClusterMember()`
D:	Inicializar variáveis de controlo do CM
F:	`void initClusterInfo()`
D:	Inicializar variáveis de controlo do *Cluster*
F:	`void initAggregationInfo()`
D:	Inicializar variáveis de Agregação
F:	`event void Boot.booted()`
D:	Inicializar Sistema
F:	`void handleAggregation(alert_t alert)`
D:	Tratamento de Agregações
F:	`void handleAlert(node_t cmInfo, uint8_t alertLevel, uint8_t alertType, uint8_t alertValue)`
D:	Tratamento de Alertas
F:	`void handleCluster(node_t member, uint8_t action)`
D:	Gestão do *Cluster*
F:	`void handleSettings(settings_t *settings)`
D:	Tratamento de Atualizações
F:	`void handleSimulation(simulation_t *simulation)`
D:	Tratamento de pedidos de Simulação
F:	`event void RadioControl.startDone(error_t error)`
D:	Tarefas pós inicialização das componentes de radiofrequência
F:	`event void RadioControl.stopDone(error_t error)`
D:	Tarefas pós paragem das componentes de radiofrequência

F:	`event void AggregationRadioSend.sendDone(message_t *msg, error_t error)`
D:	Tarefas pós envio de mensagem de Agregação
F:	`event void AlertRadioSend.sendDone(message_t *msg, error_t error)`
D:	Tarefas pós envio de mensagem de Alerta
F:	`event void ClusterRadioSend.sendDone(message_t *msg, error_t error)`
D:	Tarefas pós envio de mensagem de gestão do *Cluster*
F:	`event void SettingsRadioSend.sendDone(message_t *msg, error_t error)`
D:	Tarefas pós envio de mensagem de Atualização
F:	`event void SimulationRadioSend.sendDone(message_t *msg, error_t error)`
D:	Tarefas pós envio de mensagem de Simulação
F:	`event message_t *AggregationRadioReceive.receive(message_t* msg, void* payload, uint8_t len)`
D:	Tarefas pós receção de mensagem de Agregação
F:	`event message_t *AlertRadioReceive.receive(message_t* msg, void* payload, uint8_t len)`
D:	Tarefas pós receção de mensagem de Alerta
F:	`event message_t *ClusterRadioReceive.receive(message_t* msg, void* payload, uint8_t len)`
D:	Tarefas pós receção de mensagem de gestão do *Cluster*
F:	`event message_t *SettingsRadioReceive.receive(message_t* msg, void* payload, uint8_t len)`
D:	Tarefas pós receção de mensagem de Atualização
F:	`event message_t *SimulationRadioReceive.receive(message_t* msg, void* payload, uint8_t len)`
D:	Tarefas pós receção de mensagem de Simulação
F:	`void tempRead()`
D:	Leitura dos dados sensoriais relativos à temperatura
F:	`event void Temp.readDone(error_t error, uint16_t val)`

D:	Tarefas pós leitura dos dados sensoriais relativos à temperatura
F:	`void accelRead()`
D:	Leitura dos dados sensoriais relativos ao movimento
F:	`event void Accel.readDone(error_t error, uint32_t usActualPeriod)`
D:	Tarefas pós leitura dos dados sensoriais relativos ao movimento
F:	`event void Accel.bufferDone(error_t ok, uint16_t * buf, uint16_t count)`
D:	Tarefas pós preenchimento do *buffer* dos dados sensoriais relativos ao movimento
F:	`event void MainLoop.fired()`
D:	Tarefas de execução a cada ciclo de repetição

Tabela 7 - Tarefas e Funções da aplicação CM/CH

No anexo B.2 pode ser vista em maior detalhe a implementação das funções desenvolvidas para a aplicação CM/CH.

4.4.1.2 Aplicação Sink-Gateway

A aplicação desenvolvida para o SG tem como principal função a troca de mensagens entre a RSSF e a CS. Recorrendo à placa MIB600, em conjunto com um dos MicaZ, as mensagens recebidas via Radiofrequência (RF) são entregues à CS via Ethernet e o processo inverso é feito quando a CS pretende enviar dados para a RSSF.

Em resumo, as funcionalidades presentes na aplicação SG são as seguintes:

- Receção das mensagens enviadas pelos elementos CH e entrega dessas mesmas mensagens na CS;
- Receção das mensagens enviadas pela CS e entrega dessas mesmas mensagens aos CH da RSSF.

Seguidamente são apresentadas as interfaces TinyOS usadas e as tarefas desenvolvidas para a aplicação SG.

Interface	Função
Boot as Boot	Iniciar as componentes do TinyOS

Leds as Leds	Interagir com os três LED disponíveis
SplitControl as SerialControl	Iniciar as componentes de controlo Serial/Ethernet
AMSend as AlertSerialSend AMSend as AggregationSerialSend AMSend as SimulationSerialSend AMSend as ClusterSerialSend AMSend as SettingsSerialSend	Disponibilização dos métodos e estruturas necessárias para o envio de mensagens através de Serial/Ethernet
Receive as AlertSerialReceive Receive as AggregationSerialReceive Receive as SimulationSerialReceive Receive as ClusterSerialReceive Receive as SettingsSerialReceive	Disponibilização dos métodos e estruturas necessárias para a receção de mensagens através de Serial/Ethernet
SplitControl as RadioControl	Iniciar as componentes de radiofrequência
AMSend as AlertRadioSend AMSend as AggregationRadioSend AMSend as SimulationRadioSend AMSend as ClusterRadioSend AMSend as SettingsRadioSend	Disponibilização dos métodos e estruturas necessárias para o envio de mensagens
Receive as AlertRadioReceive Receive as AggregationRadioReceive Receive as SimulationRadioReceive Receive as ClusterRadioReceive Receive as SettingsRadioReceive	Disponibilização dos métodos e estruturas necessárias para a receção de mensagens

Tabela 8 - Interfaces TinyOS usadas na aplicação SG

Seguidamente são apresentadas as tarefas e funções desenvolvidas para que a aplicação SG possa efetuar as funcionalidades indicadas previamente.

-	**F: Nome da Função; D: Descrição da Função;**
F:	`void check(error_t ok)`
D:	Ativar Led vermelho em caso de erro
F:	`event void Boot.booted()`
D:	Inicializar Sistema

F:	`event void RadioControl.startDone(error_t error)`
D:	Tarefas pós inicialização das componentes de radiofrequência
F:	`event void RadioControl.stopDone(error_t error)`
D:	Tarefas pós paragem das componentes de radiofrequência
F:	`event void SerialControl.startDone(error_t error)`
D:	Tarefas pós inicialização das componentes de comunicação Serial/Ethernet
F:	`event void SerialControl.stopDone(error_t error)`
D:	Tarefas pós paragem das componentes de comunicação Serial/Ethernet
F:	`event void AggregationRadioSend.sendDone(message_t *msg, error_t error)`
D:	Tarefas pós envio de mensagem de Agregação através de radiofrequência
F:	`event void AlertRadioSend.sendDone(message_t *msg, error_t error)`
D:	Tarefas pós envio de mensagem de Alerta através de radiofrequência
F:	`event void ClusterRadioSend.sendDone(message_t *msg, error_t error)`
D:	Tarefas pós envio de mensagem de gestão do *Cluster* através de radiofrequência
F:	`event void SettingsRadioSend.sendDone(message_t *msg, error_t error)`
D:	Tarefas pós envio de mensagem de Atualização através de radiofrequência
F:	`event void SimulationRadioSend.sendDone(message_t *msg, error_t error)`
D:	Tarefas pós envio de mensagem de Simulação através de radiofrequência
F:	`event message_t *AggregationRadioReceive.receive(message_t* msg, void* payload, uint8_t len)`
D:	Tarefas pós receção de mensagem de Agregação através de radiofrequência
F:	`event message_t *AlertRadioReceive.receive(message_t* msg, void* payload, uint8_t len)`
D:	Tarefas pós receção de mensagem de Alerta através de radiofrequência
F:	`event message_t *ClusterRadioReceive.receive(message_t* msg, void* payload, uint8_t len)`
D:	Tarefas pós receção de mensagem de gestão do *Cluster* através de radiofrequência
F:	`event message_t *SettingsRadioReceive.receive(message_t*`

	`msg, void* payload, uint8_t len)`
D:	Tarefas pós receção de mensagem de Atualização através de radiofrequência
F:	`event message_t *SimulationRadioReceive.receive(message_t* msg, void* payload, uint8_t len)`
D:	Tarefas pós receção de mensagem de Simulação através de radiofrequência
F:	`event void AggregationSerialForward.sendDone(message_t *msg, error_t error)`
D:	Tarefas pós envio de mensagem de Agregação através de Serial/Ethernet
F:	`event void AlertSerialForward.sendDone(message_t *msg, error_t error)`
D:	Tarefas pós envio de mensagem de Alerta através de Serial/Ethernet
F:	`event void ClusterSerialForward.sendDone(message_t *msg, error_t error)`
D:	Tarefas pós envio de mensagem de gestão do *Cluster* através de Serial/Ethernet
F:	`event void SettingsSerialForward.sendDone(message_t *msg, error_t error)`
D:	Tarefas pós envio de mensagem de Atualização através de Serial/Ethernet
F:	`event void SimulationSerialForward.sendDone(message_t *msg, error_t error)`
D:	Tarefas pós envio de mensagem de Simulação através de Serial/Ethernet
F:	`event message_t *AggregationSerialReceive.receive(message_t* msg, void* payload, uint8_t len)`
D:	Tarefas pós receção de mensagem de Agregação através de Serial/Ethernet
F:	`event message_t *AlertSerialReceive.receive(message_t* msg, void* payload, uint8_t len)`
D:	Tarefas pós receção de mensagem de Alerta através de Serial/Ethernet
F:	`event message_t *ClusterSerialReceive.receive(message_t* msg, void* payload, uint8_t len)`
D:	Tarefas pós receção de mensagem de gestão do *Cluster* através de Serial/Ethernet
F:	`event message_t *SettingsSerialReceive.receive(message_t* msg, void* payload, uint8_t len)`
D:	Tarefas pós receção de mensagem de Atualização através de Serial/Ethernet

F:	`event message_t *SimulationSerialReceive.receive(message_t* msg, void* payload, uint8_t len)`
D:	Tarefas pós receção de mensagem de Simulação através de Serial/Ethernet

Tabela 9 - Tarefas e Funções da aplicação SG

No anexo B.3 pode ser vista em maior detalhe a implementação das funções desenvolvidas para a aplicação SG.

4.4.1.3 Aplicação Control-Station

De forma a gerir, tratar e apresentar os dados captados pela RSSF foi desenvolvida a aplicação CS. Esta aplicação conta com várias funcionalidades, entre as quais o tratamento dos dados recolhidos e a comunicação com a MS.

Como referido anteriormente, a aplicação CS foi desenvolvida recorrendo à linguagem Java. A escolha desta linguagem teve como principal objetivo disponibilizar a aplicação ao maior número de plataformas possível. Durante o desenvolvimento a execução foi testada nos SO GNU/Linux (Debian 7 32-bit) e Microsoft Windows (Windows 7 64-bit).

Em resumo, as funcionalidades presentes na aplicação CS são as seguintes:

- Comunicação com a RSSF através do SG;
- Comunicação com a MS através da Internet;
- Reconfiguração de parâmetros aplicacionais;
- Receção e tratamento de alertas;
- Envio de notificações de alerta através da MS;
- Pedido de simulação de geração de alerta;
- Pedido de simulação de associação/dissociação de elementos ao *cluster*;
- Apresentação e estatísticas acerca das mensagens de *Aggregation*, *Alert* e *Cluster* recebidas;
- Apresentação e estatísticas acerca das mensagens de *Simulation* e *Settings* enviadas;

- Customização e envio de *email*.

De seguida são apresentadas e analisadas as características e funcionalidades da aplicação CS.

- **Comunicação com a RSSF**

Para que a aplicação CS (Java) comunique de forma adequada com o SG (TinyOS) é necessário recorrer a uma das ferramentas disponibilizada pelo TinyOS, esta ferramenta é a aplicação SerialForwarder (SF) [140]. Recorrendo a esta ferramenta é possível criar um *socket* onde a aplicação CS poderá conectar-se via TCP/IP, ou seja, na realidade a aplicação SF assemelha-se a um *proxy* para escrita e leitura de pacotes.

Seguidamente pode ver-se um exemplo da criação do *socket* no porto 9002 do *localhost*:

```
java net.tinyos.sf.SerialForwarder -comm sf@localhost:9002
```

Outra das ferramentas necessárias para a correta codificação/descodificação das mensagens enviadas/recebidas pela CS é a aplicação Message Reader (MR).

Um exemplo de execução para a descodificação de mensagens do tipo S2sAlertMsg é o seguinte:

```
$ mig java -target=null -java-classname=AlertMsg s2s.h alert -o AlertMsg.java
$ javac AlertMsg.java
$ net.tinyos.tools.MsgReader S2sAlertMsg -comm network@10.0.0.100:10002
```

O IP 10.0.0.100 será o IP da MIB600 (SG) e 10002 o porto de escuta da MIB600. S2sAlertMsg refere-se ao tipo de mensagem analisada. Na secção "Estruturas de Informação usadas" são apresentadas as várias estruturas de mensagem usadas

Durante o desenvolvimento da aplicação CS houve o esforço para integrar estas ferramentas no código da aplicação CS, evitando assim a necessidade de execução comandos externos e facilitando o desenvolvimento de futuras funcionalidades sobre o código da CS.

Na Figura 45 é representada a integração da aplicação CS com a RSSF, recorrendo às aplicações SF e MR.

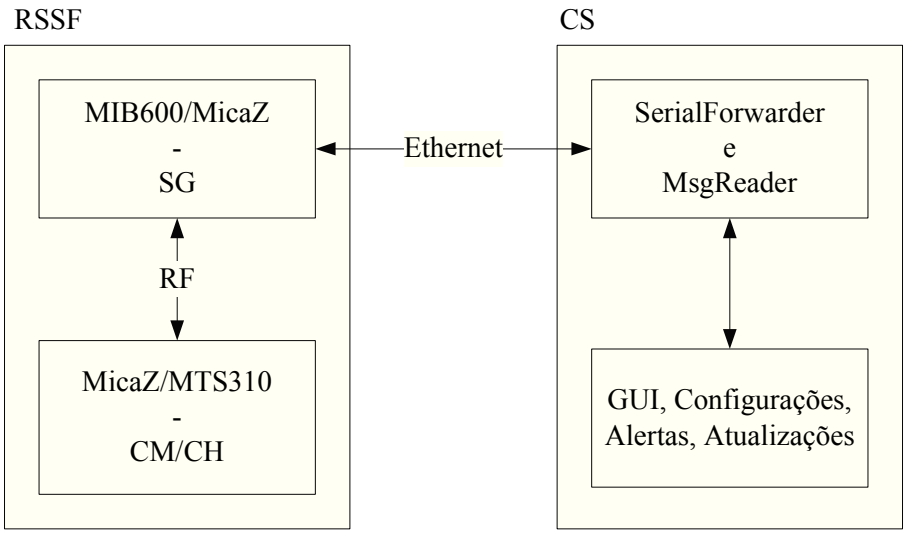

Figura 45 - Interação entre a CS e a RSSF

Aquando da inicialização da CS são pedidos ao utilizador os parâmetros para a execução do SF. Na Figura 46 é apresentado o painel de configuração do SF.

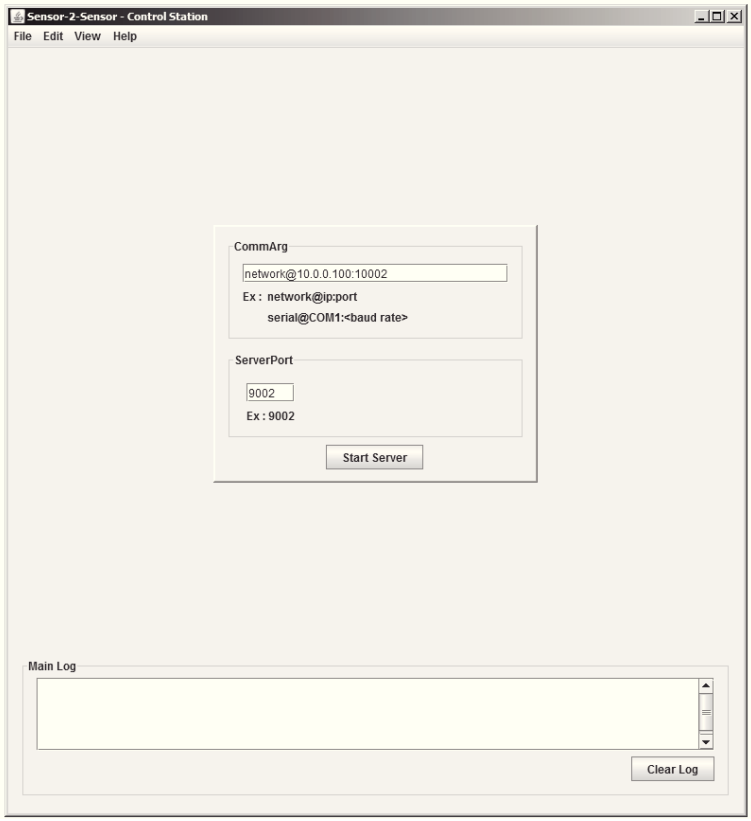

Figura 46 - Painel de configuração do SF

Como se pode ver na Figura 46, o SF estará à escuta no porto 9002 do *localhost* e que os argumentos para comunicação com a MIB600 são network@10.0.0.100:10002.

Durante o desenvolvimento, para depuração dos dados da rede, foi usada recorrentemente a aplicação MR sem recorrer à aplicação CS, ou seja, recorrendo apenas à consola. O *output* da informação tem o seguinte aspeto:

```
network@10.0.0.100:10002: resynchronising
1380140420257: Message <S2sAlertMsg>
        [msgDst=0x0]
        [nodeInfo.nodeId=0xa]
        [nodeInfo.type=0x3]
        [nodeInfo.parentId=0x5]
        [level=0x1]
        [type=0x1]
        [value=0xf4]
```

Estes são os dados relativos às mensagens do tipo S2sAlertMsg enviados pelos CH, contendo a informação que interessa tratar na aplicação CS.

- **Comunicação com a Main-Station**

Uma das principais funcionalidades da aplicação CS é a comunicação com a MS, um elemento externo que, à partida, se encontra na Internet. Na tentativa de demonstrar esta integração recorreu-se à implementação das capacidades necessárias para comunicação com um serviço de *email*.

Depois de um estudo acerca das interfaces de programação de aplicação (API) disponíveis para o envio de *email*, a escolha recaiu sobre a API JavaMail versão 1.4.3 [141]. A API JavaMail é independente da plataforma em que a aplicação CS está a ser executada e suporta os protocolos SMTP, POP3 e IMAP.

Na Figura 47 é apresentado o painel de configuração para o envio de notificações de alerta por *email*.

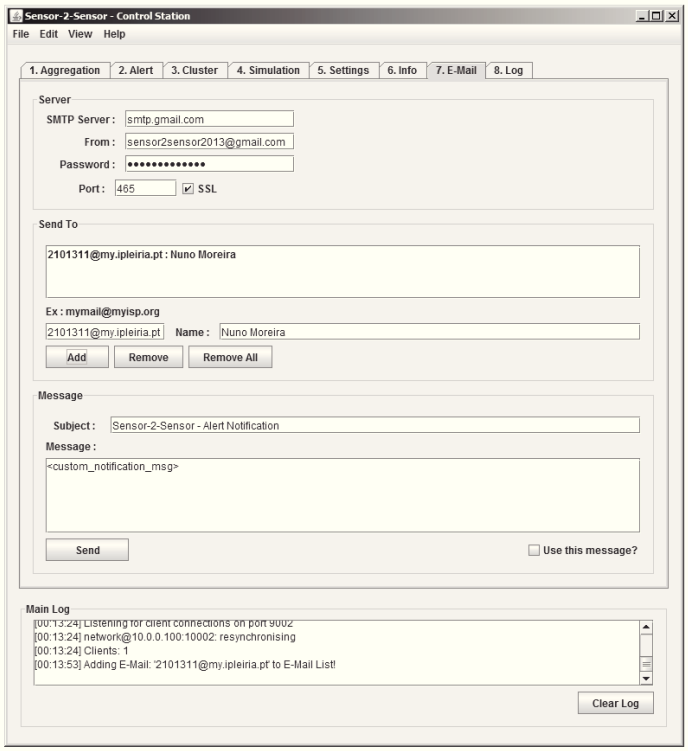

Figura 47 - Painel de configuração da notificação por *email*

Como se pode ver na Figura 47, é possível configurar os dados da conta usada para o envio (Server) e adicionar várias contas de *email* de destino (Send To).

- **Estruturas de Informação usadas**

Para gerir a informação necessária para o correto funcionamento da RSSF, desde o envio de mensagens até à receção de novos parâmetros de execução, foram criadas várias estruturas de dados. Este tipo de organização não só permite uma gestão de código mais simples mas também acaba por permitir o fácil crescimento da informação que circula na rede. As estruturas de dados usadas no protótipo são as seguintes:

- **node_t**

Tipo	Campo	Descrição
nx_uint8_t	nodeId	Indentificador do Elemento (Endereço)
nx_uint8_t	type	Indentificador do Tipo de Elemento (CM, CH, SG)
nx_uint8_t	parentId	Endereço do Elemento Pai (CH Id para CM e SG Id para CH)

Tabela 10 - Estrutura de dados node_t

- **alert_t**

Tipo	Campo	Descrição
nx_uint8_t	msgDst	Endereço de Destino do Alerta
node_t	nodeInfo	Informação do Elemento que criou o Alerta
nx_uint8_t	level	Nível do Alerta (Critico ou Não Critico)
nx_uint8_t	type	Tipo do Alerta (Movimento ou Temperatura)
nx_uint8_t	value	Valor do Alerta

Tabela 11 - Estrutura de dados alert_t

- **aggregation_t**

Tipo	Campo	Descrição
nx_uint8_t	msgDst	Endereço de Destino da Agregação
nx_uint8_t	alertCount	Número de Alertas presentes na Agregação.
node_t	nodeInfo	Informação do Elemento que criou a Agregação
alert_t	alert0	Estrutura de Alerta
alert_t	alert1	Estrutura de Alerta
alert_t	alert2	Estrutura de Alerta

Tabela 12 - Estrutura de dados aggregation_t

- **cluster_t**

Tipo	Campo	Descrição
nx_uint8_t	msgDst	Endereço de Destino da Mensagem
nx_uint8_t	infoCode	Código de Operação (Associação/Dissociação)
nx_uint8_t	memberNumber	Número de CM associados ao *Cluster*
node_t	nodeInfo	Informação do Elemento que criou a Mensagem
node_t	node0	Estrutura de Elemento (CM)
node_t	node1	Estrutura de Elemento (CM)
node_t	node2	Estrutura de Elemento (CM)

Tabela 13 - Estrutura de dados cluster_t

- **settings_t**

Tipo	Campo	Descrição
nx_uint8_t	msgDst	Endereço de Destino da Mensagem
nx_uint8_t	type	Tipo de dados a atualizar (Temperatura/Movimento)
nx_uint8_t	value	Novo valor de *Threshold* (Temperatura/Movimento)

Tabela 14 - Estrutura de dados settings_t

- **simulation_t**

Tipo	Campo	Descrição
nx_uint8_t	msgDst	Endereço de Destino da Mensagem
nx_uint8_t	type	Tipo de Simulação (Alerta ou Associação/Dissociação)
nx_uint8_t	value0	Valor de Simulação 0
nx_uint8_t	value1	Valor de Simulação 1
nx_uint8_t	value2	Valor de Simulação 2
nx_uint8_t	value3	Valor de Simulação 3

Tabela 15 - Estrutura de dados settings_t

- **Reconfiguração da Aplicação**

Outra das funcionalidades implementadas é a distribuição de novos parâmetros aplicacionais. Desta forma, os CM/CH podem usar novas configurações sem que haja a necessidade de parar o sistema e proceder ao *flash* do novo *firmware*. Recorrendo a esta funcionalidade é possível passar a usar, um novo *threshold* de temperatura/movimento imediatamente após a receção da atualização. A estrutura de dados recebida no *update* é a estrutura settings_t analisada anteriormente.

Na Figura 48 é apresentado o painel de atualização presente na aplicação CS.

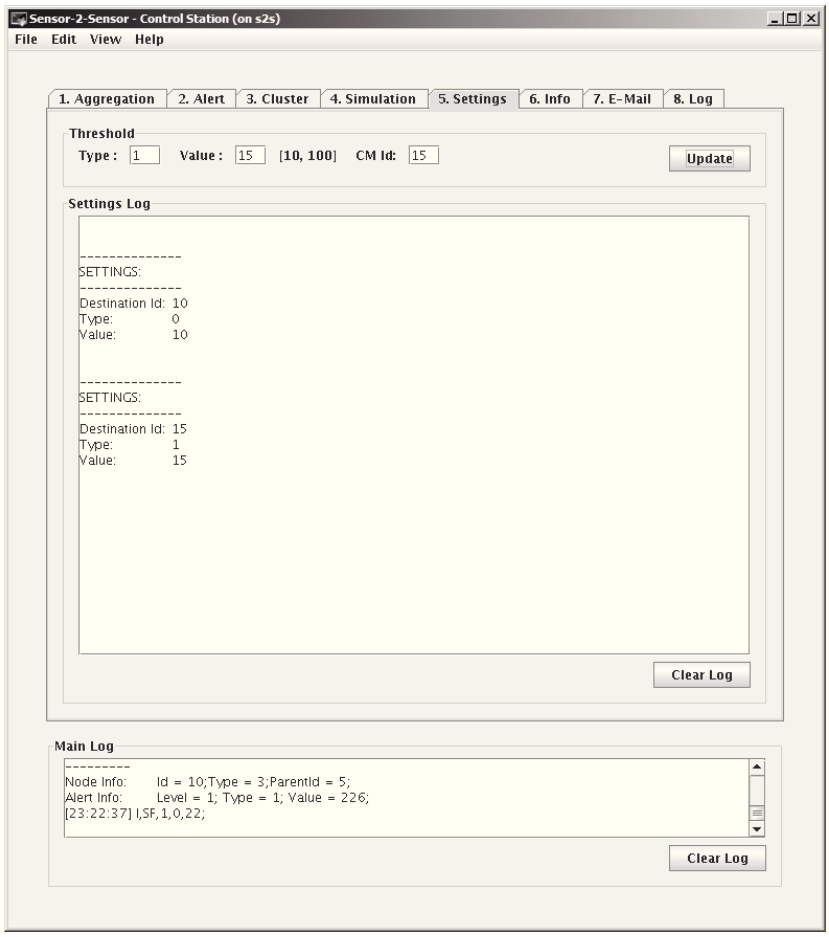

Figura 48 - Painel de Configurações

O mecanismo de distribuição de novos *settings* foi desenvolvido de maneira a que se possa indicar qual o elemento a que o *update* se destina.

Como se pode ver na Figura 48, foram realizadas duas ações de atualização, na primeira o CM com o identificador 10 (Destination Id) passou a usar o valor 10 (Value) como *threshold* de temperatura (Type 0) e na segunda ação de atualização o CM com o identificador 15 passou a usar o valor 15 como *threshold* de movimento (Type 1).

- **Receção de Alertas**

No painel Alert é possível visualizar os detalhes das mensagens de alerta recebidas e os números totais de alertas recebidos. Todos os alertas recebidos são apresentados, sejam eles críticos ou não críticos (Agregações).

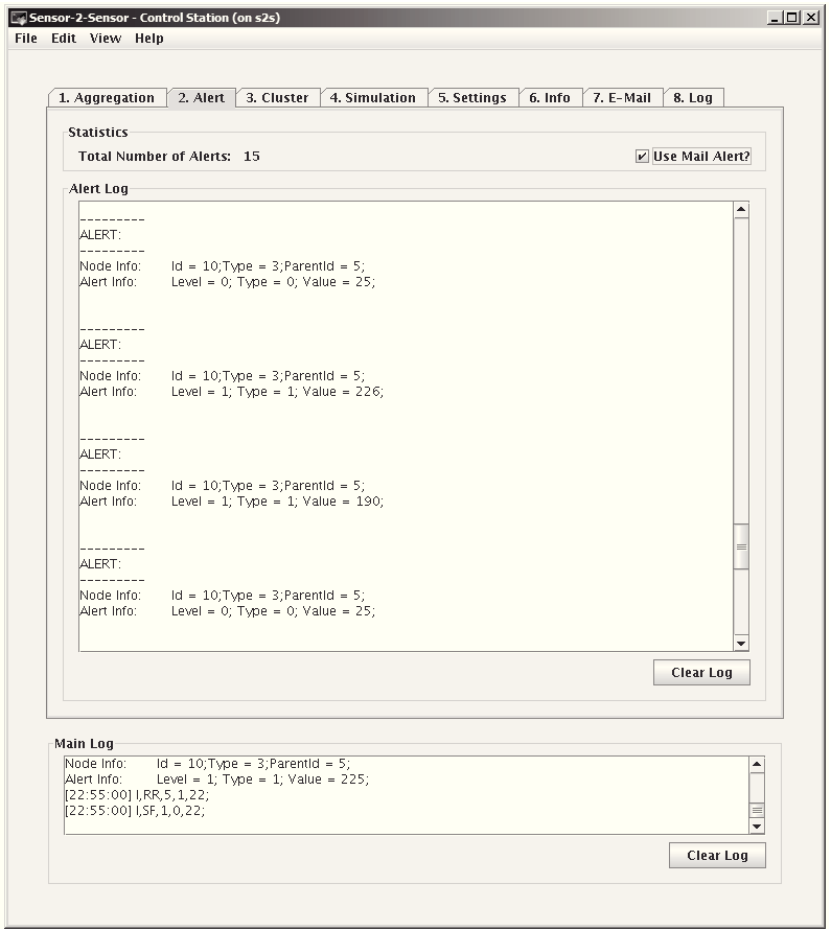

Figura 49 - Painel de Alertas

Olhando para o último registo presente na Figura 49 pode ver-se que foi recebido um alerta de temperatura não critico com o valor 25. O alerta foi gerado pelo CM (Type 3) com o identificador 10 (Id), associado ao CH com o identificador 5 (ParentId).

- **Receção de Agregações**

No painel Aggregation é possível visualizar os detalhes das mensagens de agregação recebidas e os números totais de alertas agregados.

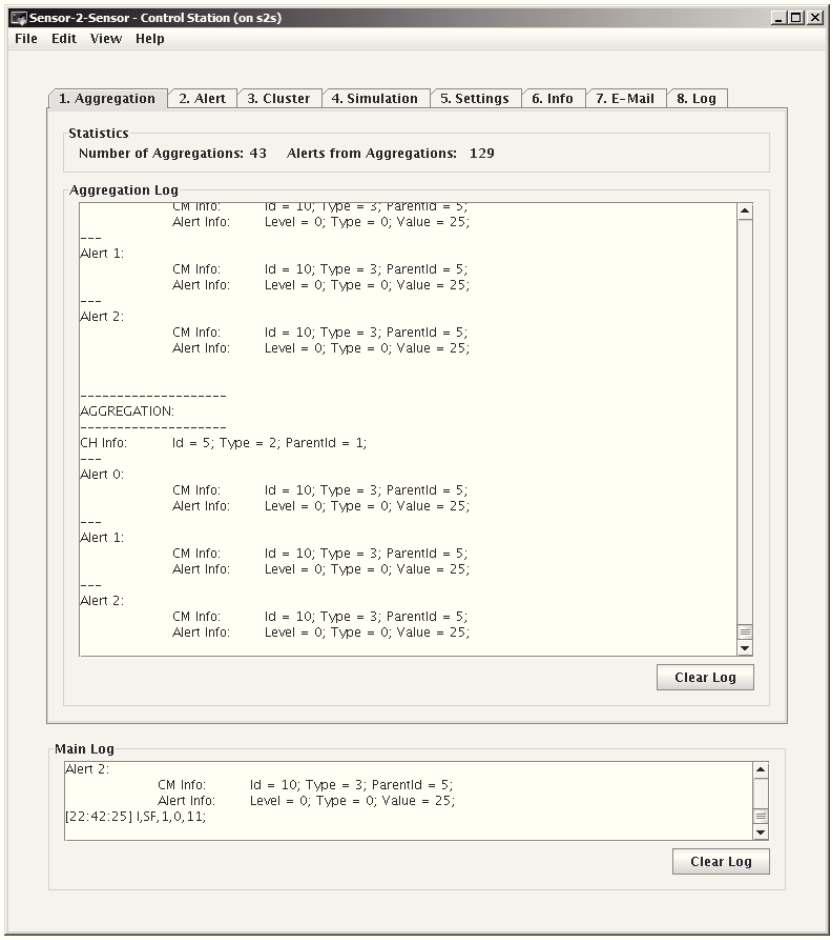

Figura 50 - Painel de Agregações

Na Figura 50 são apresentadas as agregações dos vários alertas não críticos (Level 0). Neste caso, os alertas de temperatura são designados como alertas não críticos e são agregados em blocos de três.

- **Eventos ocorridos no Cluster**

No painel Cluster é apresentada a informação relacionada com eventos de associação e dissociação a um dado CH.

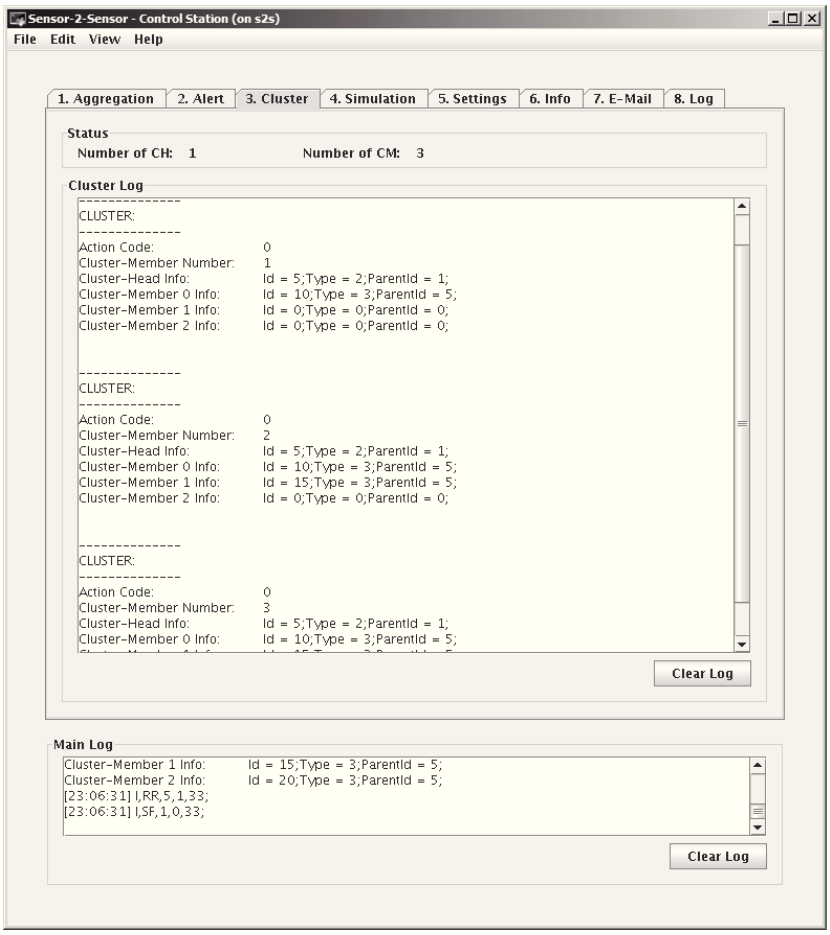

Figura 51 - Painel de Cluster

Olhando para o primeiro evento registado na Figura 51 pode ver-se que o CM com o identificar 10 se associou (Action Code 0) ao CH com o identificador 5.

- **Simulação de Eventos**

Dada a limitação no número de MicaZ disponíveis, foram desenvolvidas as funcionalidades necessárias para simular a geração de alertas e eventos de gestão do *cluster*.

Recorrendo ao painel Simulation é possível parametrizar a simulação pretendida. Tal como os detalhes da simulação invocada, é também visível o retorno da operação efetuada.

No exemplo apresentado na Figura 52, foi simulada a dissociação do CM com o identificador 20 ao CH com o identificador 5.

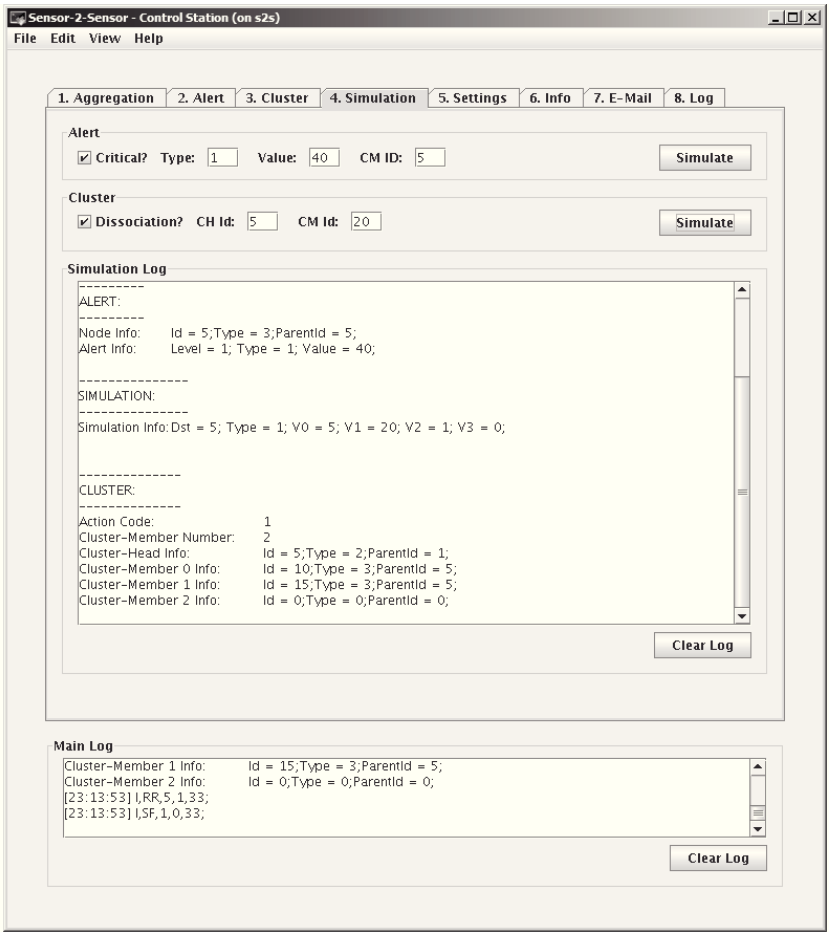

Figura 52 - Painel de Simulação

- **Estatísticas da Rede**

No painel Info são apresentadas as configurações do SerialForwarder e as estatísticas de rede referentes ao número de mensagens processadas pela CS.

Recorrendo a esta informação, é possível quantificar na globalidade o número de mensagens recebidas e enviadas pela CS.

Figura 53 - Painel de Informação

- **Visualização e Exportação de Registos**

No painel Log é possível visualizar o registo das várias operações efetuadas pela CS, como por exemplo a receção/envio de mensagens.

Com se pode ver na Figura 54, existe a possibilidade de exportar os registos apresentados para um ficheiro de texto.

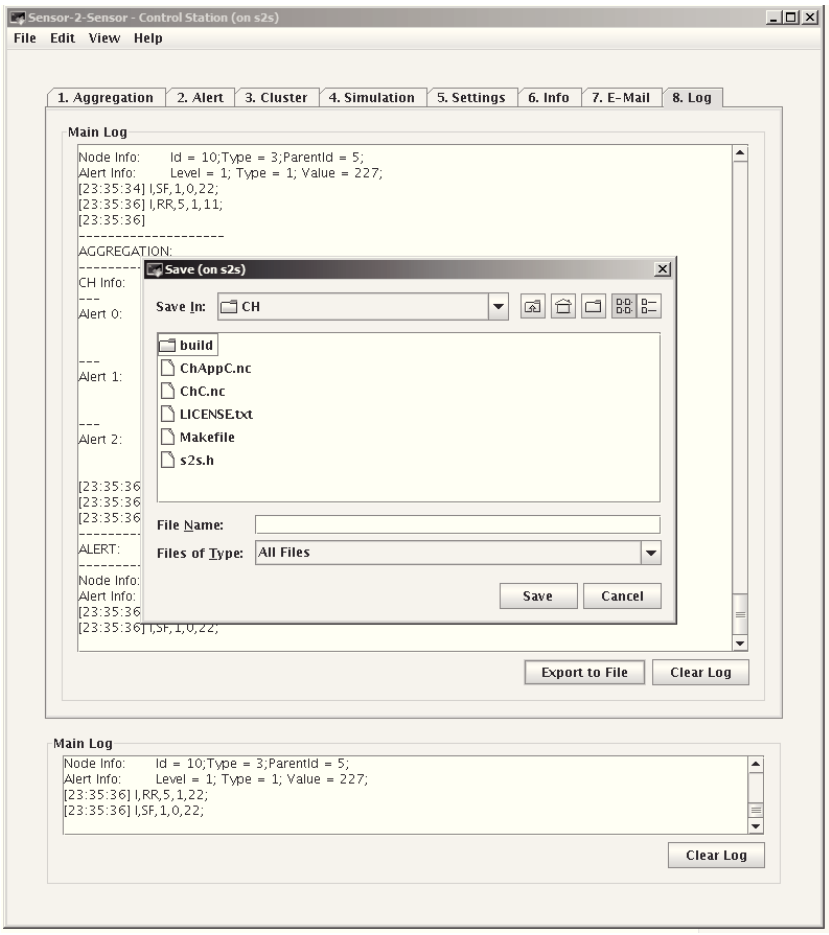

Figura 54 - Painel de Log

4.4.1.4 Recursos necessários ao Protótipo

Na Tabela 16 e Tabela 17 são apresentadas as listas de *hardware*, *software* e outros recursos necessários para o funcionamento do protótipo desenvolvido.

Hardware		
Item	**Quantidade**	**Função**
MicaZ	2	Execução das aplicações CM/CH e SG
MTS310	1	Aquisição de dados sensoriais (Elemento CM/CH)
MIB600	1	Execução da aplicação SG em conjunto com MicaZ
PC	1	Execução da aplicação CS e conectividade com a Internet
Baterias AA LR6	2	Alimentar o elemento CM/CH (MicaZ)
Adaptador AC MIB600	1	Alimentar a placa MIB600
Cabo RJ-45	1	Ligação Ethernet entre a MIB600 (SG) e PC (CS). (Caso sejam ligados diretamente, deve ser usado um cabo RJ-45 cruzado)

Tabela 16 - Requisitos de Hardware do Protótipo

Software	
Requisito	**Função**
TinyOS 2.x.x	Compilação e instalação do código das aplicações CM/CH e SG
Java 1.6	Compilação e execução da aplicação CS e ferramentas de controlo do TinyOS
Sistema Operativo GNU/Linux ou Windows	Execução da aplicação CS
Ligação à Internet	Comunicação entre a CS e a MS

Tabela 17 - Requisitos de Software do Protótipo

No anexo A são indicados os passos necessários para executar a máquina virtual (VM) que contém as ferramentas e aplicações necessárias para a execução do protótipo.

4.5 Síntese

Neste capítulo foi apresentada a implementação da arquitetura proposta. Começou-se por definir quais os requisitos do protótipo e qual o *hardware* e *software* utilizado na sua implementação.

Posteriormente foi apresentado o cenário de implementação do protótipo, onde são identificadas as várias entidades essenciais à arquitetura e quais as suas funcionalidades. Esta implementação permitiu efetuar um conjunto de testes às funcionalidades e mecanismos propostos.

Os resultados obtidos na avaliação dessas funcionalidades e mecanismos serão apresentados no capítulo seguinte.

5 Testes

O presente capítulo apresenta um conjunto de testes efetuados para avaliar a arquitetura e a comunicação entre as suas várias entidades. Inicialmente será apresentado o cenário de testes usado e posteriormente os objetivos, detalhes e resultados dos testes relevantes para a avaliação da solução. Por último, é apresentada uma síntese tendo por base a análise dos resultados obtidos.

5.1 Cenário de Testes

O cenário de testes usado vai de encontro ao cenário apresentado na secção "Visão Global do Protótipo", ou seja, todas as entidades especificadas estão presentes no cenário de testes apresentado.

Os testes efetuados têm o objetivo de verificar o comportamento da solução no tratamento de determinados eventos e ações. O comportamento testado com o protótipo apresentado engloba os seguintes eventos e ações:

- Alerta de Temperatura e Agregação de Alertas;
- Alerta de Movimento e comunicação com a Main-Station;
- Atualização de Configurações;
- Teste de Simulação:
    - Alerta de temperatura de nível Crítico;
    - Alerta de temperatura de nível Não-Crítico;
    - Alerta de movimento de nível Crítico;

- Alerta de movimento de nível Não-Crítico;
- Associação de um Cluster-Member (CM) ao Cluster-Head (CH);
- Dissociação de um Cluster-Member (CM) ao Cluster-Head (CH).

Em resumo, os recursos usados no cenário de testes foram:

Quantidade	Hardware
2	MicaZ
1	MTS310
1	MIB600
1	Dell E5420
1	Adaptador AC MIB600
2	Pilhas AA LR6
1	Cabo RJ-45 Cruzado

Tabela 18 - Hardware utilizado no cenário de testes

Software
Windows 7
VirtualBox 4.1.20 com o ambiente TinyOS 2.1.2 instalado no Debian 7
Comunicação com o servidor smtp.gmail.com (Internet)

Tabela 19 - Software utilizado no cenário de testes

A Figura 55 apresenta todo o equipamento usado no cenário de testes.

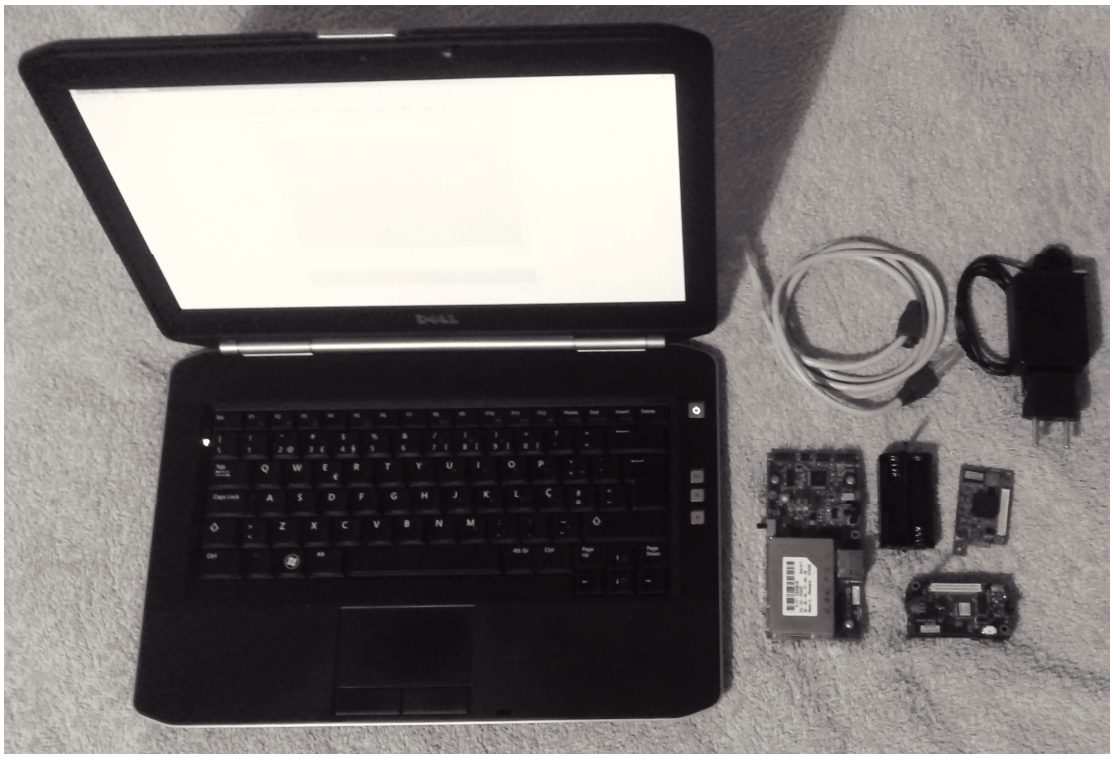

Figura 55 - Equipamento usado no cenário de testes

A Figura 56 representa a arquitetura do protótipo.

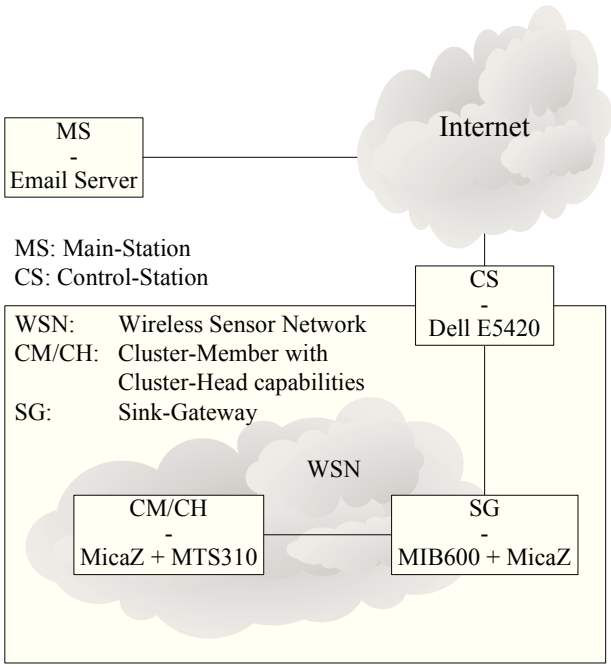

Figura 56 – Arquitetura do Protótipo

5.2 Alerta de Temperatura e Agregação de Alertas

O objetivo deste teste é o de verificar a deteção de temperatura elevada e posterior notificação à Control-Station (CS).

A sequência de tarefas necessárias para efetuar o teste pretendido é a seguinte:

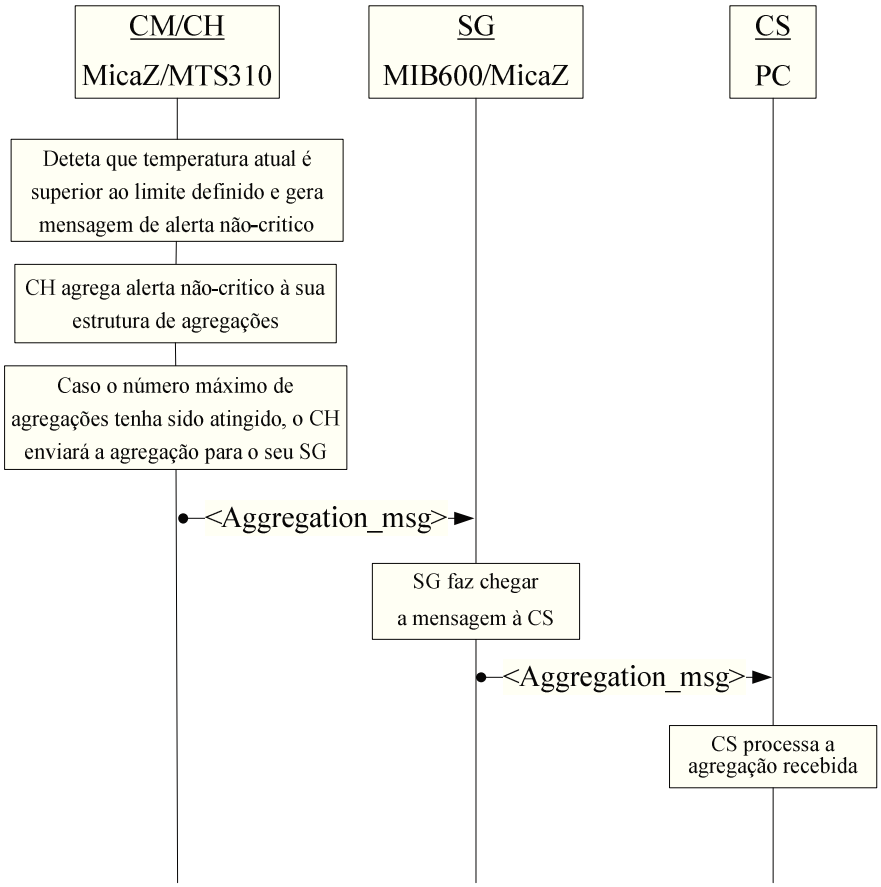

Figura 57- Teste de Alerta de Temperatura e Agregação de Alertas

Como é visível na Figura 57, o elemento CM deverá analisar a temperatura atual e verificar se o valor ultrapassa o *threshold* definido para a geração de alerta. À partida, para este teste, o *threshold* definido foi de 37 ºC.

Caso o valor de temperatura ultrapasse o limite definido, o CM deverá gerar um alerta não-crítico e enviar a mensagem para o seu CH. Sendo este um alerta não-critico, o CH deverá efetuar a agregação da mensagem recebida à sua estrutura de agregação.

Caso o número máximo de agregações tenha sido atingido, o CH deverá encaminhar imediatamente a agregação de alertas à CS. Neste caso são usados três alertas por agregação. A agregação das mensagens é aplicada diretamente pelo CH, juntando as várias mensagens numa estrutura de agregação.

5.2.1 Resultados dos testes ao Alerta de Temperatura e Agregação de Alertas

Ao executar o teste descrito, verificou-se que o valor da temperatura recolhido pelo CM é sempre igual a zero. De maneira a diagnosticar o problema foi feito um teste com uma aplicação simples, onde apenas eram recolhidos os dados de temperatura e eram usados os LED do MicaZ para notificação caso o valor de temperatura recolhido fosse diferente de zero. Após a execução do teste o erro na recolha do valor de temperatura continuou a manifestar-se.

Até ao momento da entrega da dissertação, não foi possível concluir o porquê desta situação, contudo, acredita-se que poderá estar relacionado com o facto de o sensor de temperatura presente na placa MTS310 não estar a recolher a informação corretamente ou com o facto de a versão do sistema operativo (SO) TinyOS usada apresentar alguma falha na leitura do *buffer* utilizado para a informação de temperatura.

Para contornar esta falha, permitindo a criação de alertas de temperatura, o valor devolvido pelo sensor foi definido estaticamente como sendo igual a 25 ºC. Depois disto, basta configurar um *threshold* abaixo de 25 ºC, para gerar alertas. Desta forma, a cada ciclo de verificação (1000 ms) o CM irá verificar que o valor de temperatura é superior ao limite definido e irá gerar um alerta de temperatura.

Ao receber um alerta não-critico, o CH irá proceder à agregação da mensagem recebida. Visto que o limite de agregações definido é igual a três, e que a cada segundo será gerado um alerta de temperatura pelo CM, o CH irá enviar para o seu Sink-Gateway (SG) uma mensagem de agregação a cada três segundos. O limite máximo de três alertas é aplicado independentemente do CM que gera os alertas, ou seja, a agregação será feita em caso de alertas gerados por diferentes CM e no caso em que um CM gera um conjunto de três alertas sequenciais. Para efeitos de diferenciação de comportamento da CS dependendo da criticidade dos alertas gerados na Rede Sensorial Sem Fios (RSSF), foi decidido que os

alertas não-críticos não notificariam a Main-Station (MS).

A Figura 58 representa a receção das mensagens de agregação na CS.

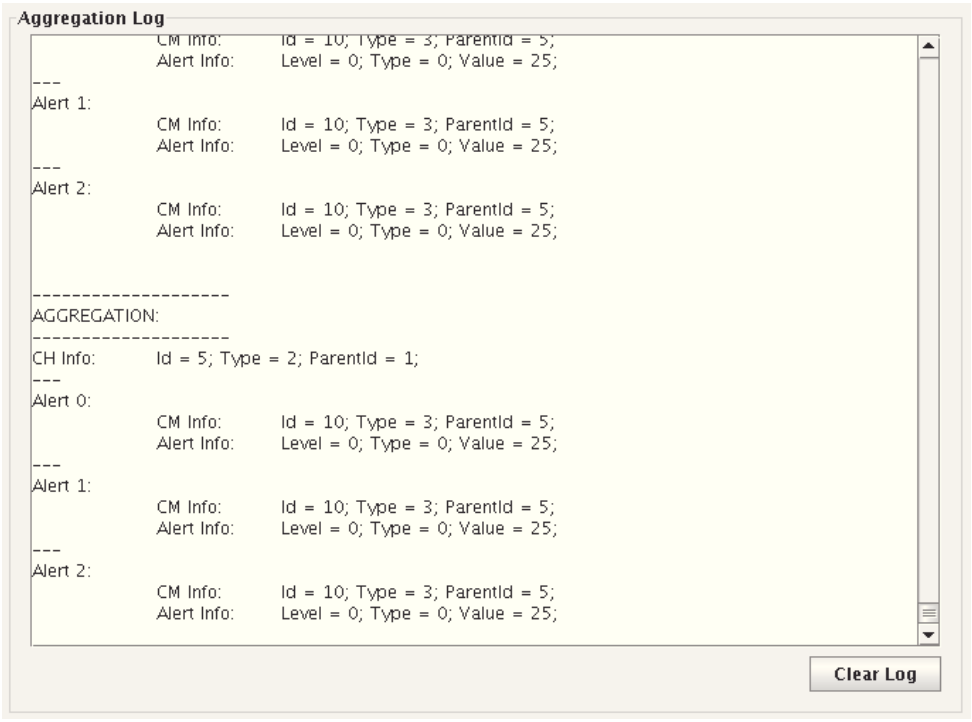

Figura 58 – Receção de Agregações de Alertas

Tendo como base de avaliação a receção e apresentação dos alertas agregados na CS, conclui-se o sucesso nas seguintes tarefas:

- Criação de um alerta de temperatura por parte do CM;
- Agregação de mensagens não-críticas pelo CH;
- Envio de mensagens de Agregação desde o CH até ao SG;
- Envio de mensagens de Agregação desde o SG até à CS;
- Processamento e apresentação dos alertas agregados na interface da CS.

5.3 Alerta de Movimento e comunicação com a MS

O objetivo deste teste é o de verificar a deteção de movimento, comunicação com a CS e posterior notificação à MS representada pelo uso de um servidor de *email*.

A sequência de tarefas necessárias para o teste pretendido é a seguinte:

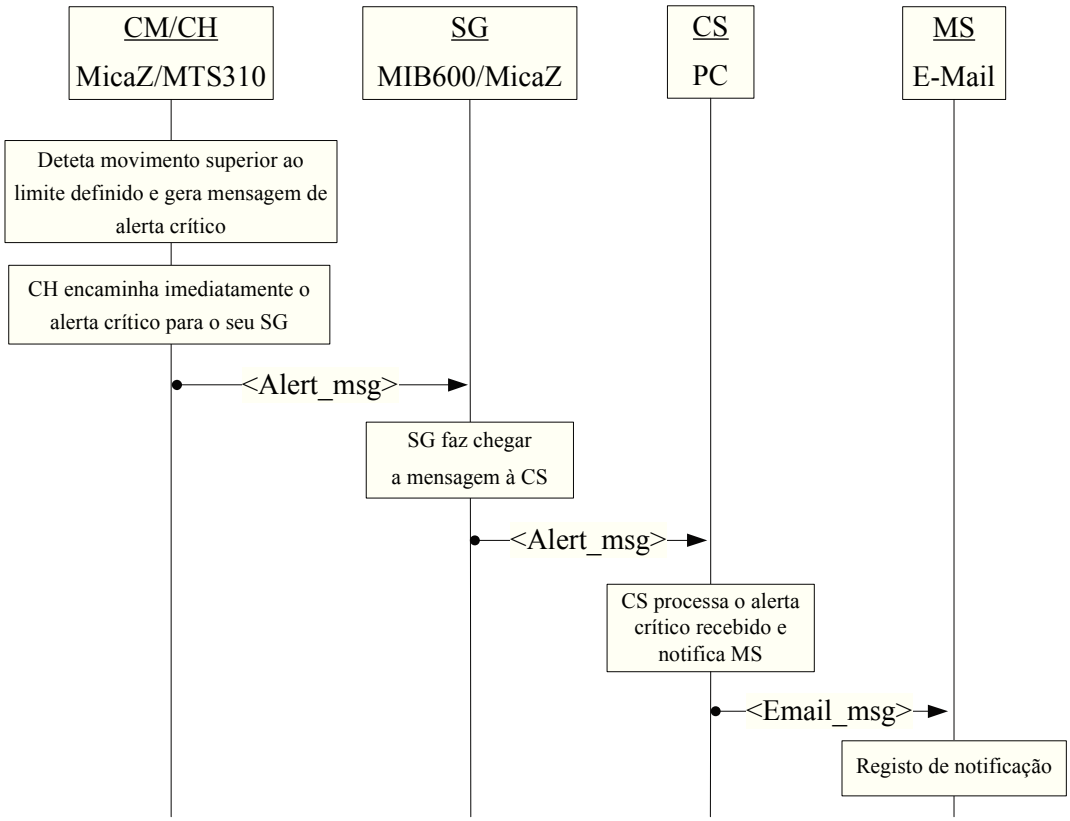

Figura 59- Teste de Alerta de Movimento e comunicação com a Main-Station

Como se pode ver na Figura 59, o elemento CM deverá detetar o movimento, neste caso o movimento do objeto composto pelo MicaZ/MTS310, e verificar se é ultrapassado o *threshold* definido para a geração de alerta.

A técnica utilizada para detetar o movimento baseia-se no cálculo da diferença do valor absoluto recolhido entre ciclos (1000ms). A placa MTS310 está equipada com um acelerómetro de dois eixos (x,y) mas para efeitos de demonstração foi apenas usado um dos eixos (x). Na Figura 60 está representada uma sequência de valores recolhidos e a geração de

um alerta caso a diferença entre ciclos ultrapasse o *threshold* definido (10).

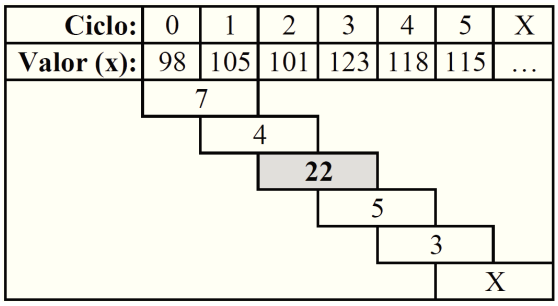

Figura 60 - Deteção de Movimento pelo CM

Como se pode ver na Figura 60, a diferença de valores entre os ciclos número três e número dois causou um alerta de movimento com o valor 22.

Ao ser registado um valor superior ao *threshold* definido, o CM deverá gerar um alerta de nível crítico e enviar a mensagem para o seu CH. Sendo este um alerta crítico, o CH deverá encaminhar imediatamente o alerta.

De maneira a provocar o valor de movimento necessário para que seja gerado um alerta, basta inverter a posição do objeto composto pelo MicaZ/MTS310.

5.3.1 Resultados dos testes ao Alerta de Movimento e comunicação com a MS

Ao provocar o movimento suficiente para que o *threshold* de alerta seja ultrapassado, verificou-se a correta geração de um alerta crítico por parte do CM e consequente encaminhamento da mensagem até à CS.

A Figura 61 representa a receção do alerta crítico na CS.

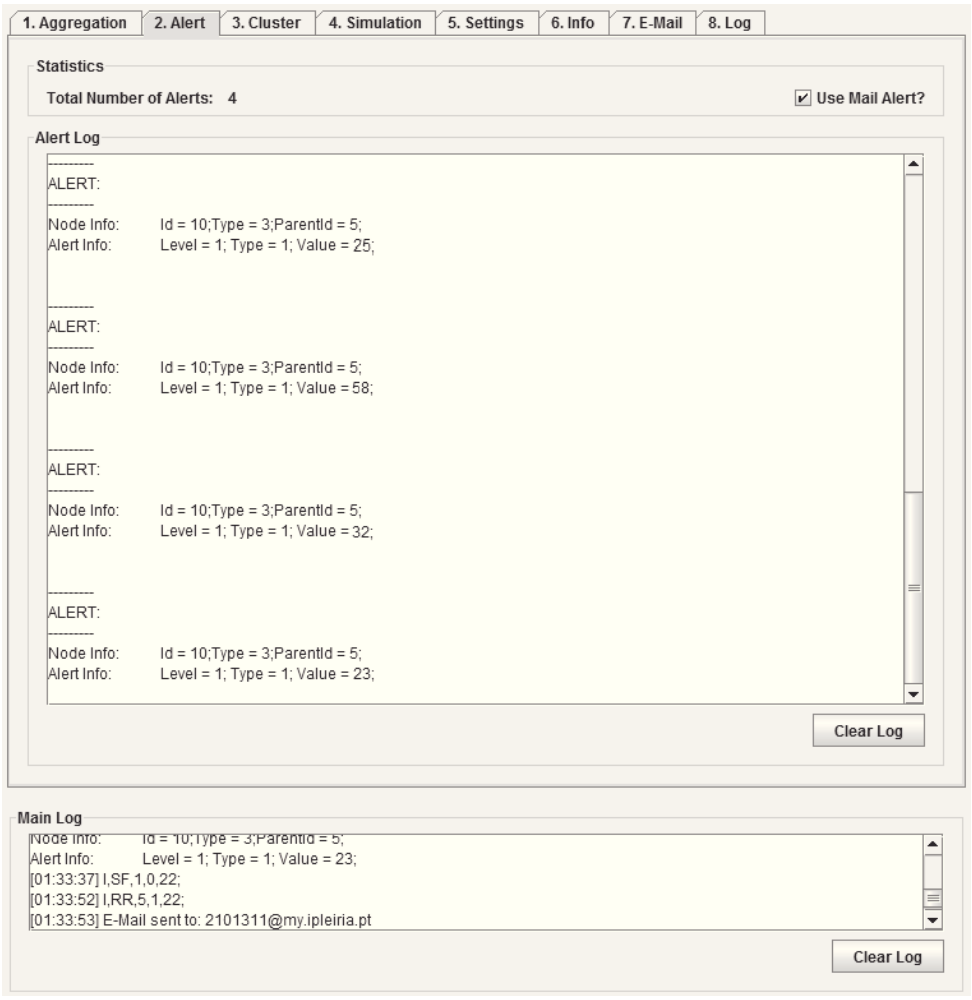

Figura 61 - Receção de Alerta Crítico na CS

Sendo este um alerta crítico, a CS deverá notificar a MS através do serviço de notificações por *email*. Após a receção do alerta na CS, é imediatamente desencadeado o processo de envio de *email*. Recorrendo a uma comum ligação à Internet, o tempo desde o envio da CS até à receção na MS é relativamente reduzido, como mostra a Tabela 20.

Teste	Valor do Alerta	Tempo até entrega na CS	Tempo até entrega na MS
1	21	1s	5s
2	25	1s	6s
3	58	1s	7s
4	32	1s	5s
5	23	1s	7s

Tabela 20 - Tempo despendido na entrega dos alertas críticos

Como se pode ver na Tabela 20, o tempo desde a deteção do evento pelo CM até a apresentação na CS ronda em todos os testes cerca de 1 segundo. Já o tempo de entrega desde a deteção até entrega na MS varia entre os 5 e 7 segundos.

Figura 62 - Receção de Notificação de Alerta na MS

Tendo como base de avaliação a deteção do movimento por parte do CM e entrega das notificações de alerta na MS, conclui-se o sucesso nas seguintes tarefas:

- Aquisição dos dados sensoriais fornecidos pelo acelerómetro presente na placa MTS310;

- Aplicação dos cálculos necessários para detetar o movimento;
- Criação de um alerta de movimento por parte do CM;
- Envio da mensagem de Alerta desde o CH até ao SG;
- Envio de mensagens de Alerta desde o SG até à CS;
- Processamento e apresentação dos alertas agregados na interface da CS;
- Interação entre a CS e a MS através do envio do *email* com notificação de alerta;
- Apresentação dos dados relativos ao alerta na interface da MS.

5.4 Atualização de Configurações

O objetivo deste teste é o de verificar a correta distribuição e implementação de novos valores de configuração no elemento CM. Desta forma, poderão ser definidos novos valores para as variáveis de *threshold* usadas na análise de alertas de temperatura e movimento.

A sequência de tarefas necessárias para o teste pretendido é a seguinte:

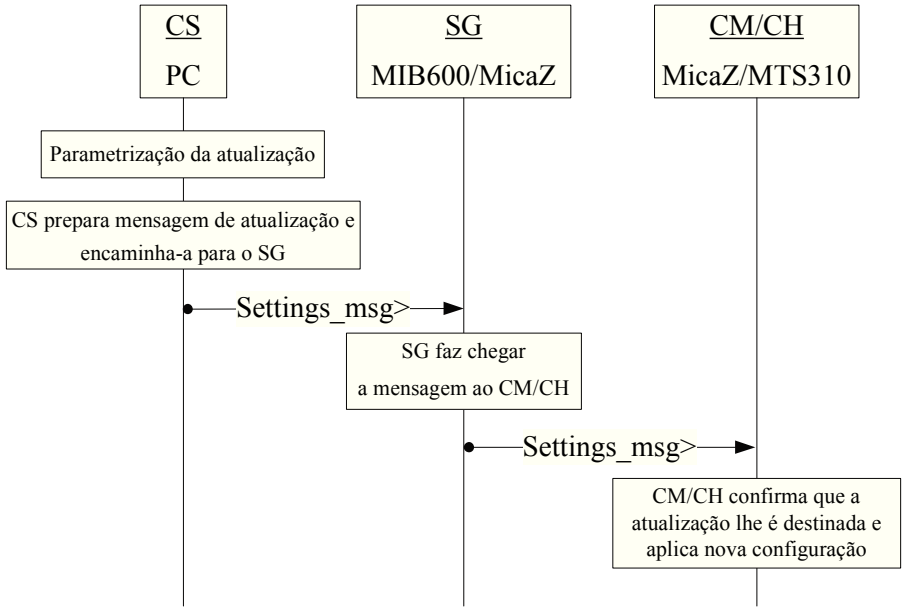

Figura 63 - Teste de Atualização de Configurações

Como se pode ver na Figura 63, é possível através da interface oferecida pela CS

parametrizar as mensagens de atualização. Aquando da invocação de envio, a CS recolhe a informação parametrizada e cria uma mensagem de atualização contendo o identificador do CM a que a atualização se destina, além do tipo de alerta e o seu novo *threshold*. Após a criação da estrutura, a mensagem é encaminhada para o SG que por sua vez a reencaminhará para o CM/CH.

Ao receber a mensagem de Atualização, o CM confirma que a atualização lhe é destinada, verifica qual o tipo de alerta e define o valor recebido como o novo *threshold* para o alerta em questão.

De maneira a testar a funcionalidade de atualização de configurações, decidiu-se usar o alerta de temperatura como exemplo. Os resultados desse teste são apresentados de seguida.

5.4.1 Resultados dos testes de Atualização de Configurações

Tirando partido do facto do valor de temperatura reportado pelo CM ser sempre igual a 25ºC, basta apenas definir um valor inferior a 25 como novo *threshold* para provocar a correta atualização da configuração.

Ao aplicar um valor de *threshold* inferior a 25, este provoca a geração de um alerta após a análise efetuada pelo CM. Visto que os alertas de temperatura são considerados de nível não-crítico, o CH irá proceder à agregação do alerta e proceder ao envio da agregação quando o limite de alertas for atingido, neste caso três alertas.

A Figura 64 demonstra a execução desde teste.

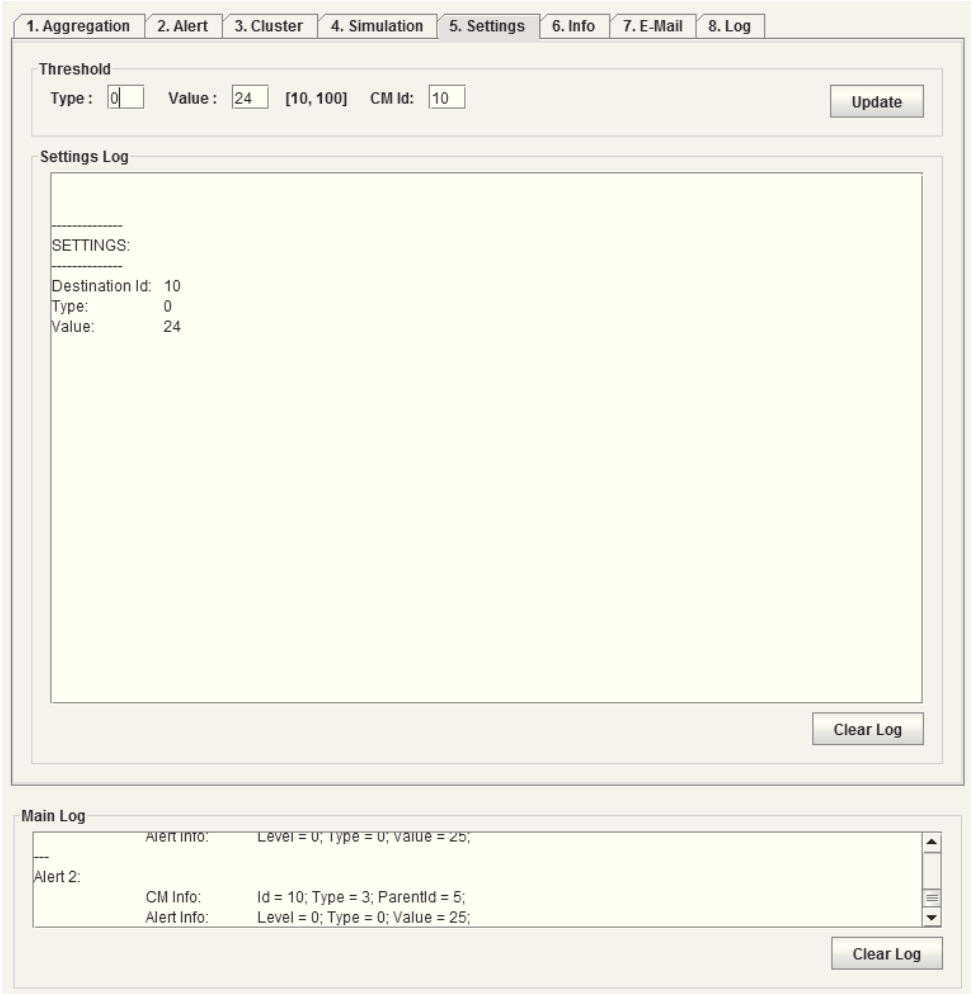

Figura 64 - Atualização de Configuração

Como se pode ver na Figura 64, os valores de parametrização indicam que deve ser feita uma atualização ao CM com o identificador 10 e que o novo valor de *threshold* para os alertas de temperatura deve ser igual a 24.

Os detalhes da mensagem enviada são apresentados e imediatamente depois verifica-se que o CH envia a agregação com os alertas de temperatura provocados pela redefinição do *threshold* para um valor inferior a 25.

Tendo como base de avaliação o envio da mensagem de atualização e a aplicação do novo valor de configuração por parte do CM, conclui-se o sucesso nas seguintes tarefas:

- Introdução de parâmetros de atualização através da interface da CS;

- Geração de mensagens de Atualização pela CS e o seu posterior envio até ao SG;
- Envio de mensagens de Atualização desde o SG até ao CH/CM;
- Aplicação e utilização imediata de novos valores de configuração no CM.

5.5 Teste de Simulação

O objetivo deste teste é o de verificar a correta execução dos pedidos de simulação invocados ao CM.

Tal como referido anteriormente, recorrendo ao painel Simulation é possível efetuar a geração de alertas personalizados de temperatura e movimento. Para além da geração de alertas é também possível simular eventos de associação e dissociação ao CH.

A Figura 65 apresenta, genericamente, a sequência de tarefas necessárias para efetuar a simulação de alertas.

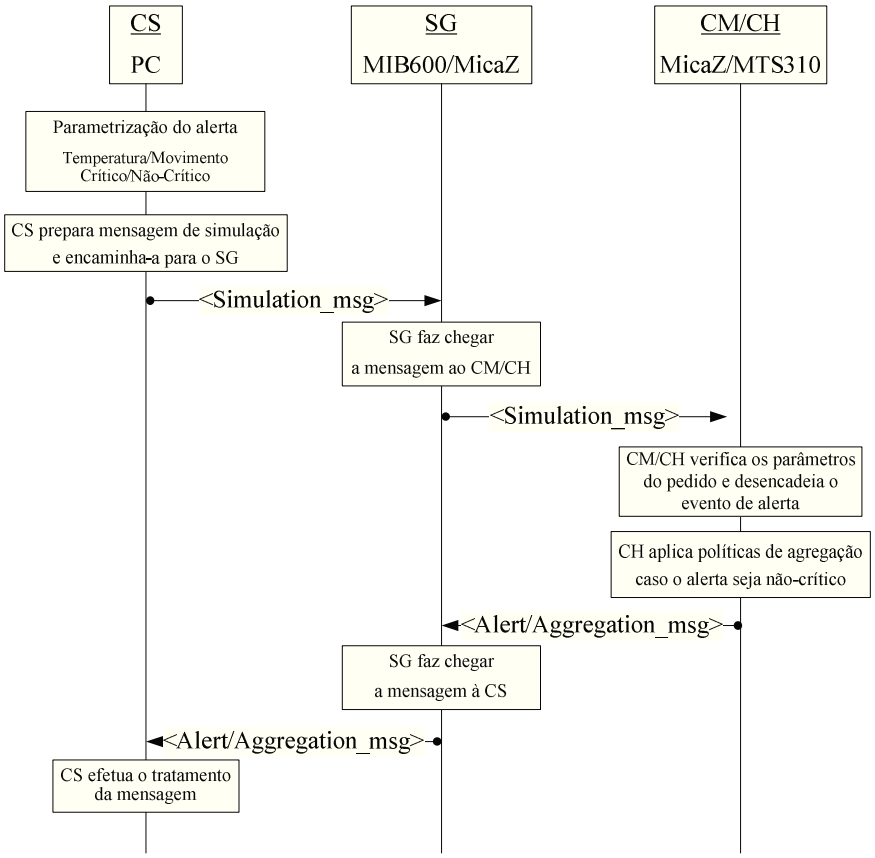

Figura 65 - Testes à Simulação de Alertas

Como se pode ver na Figura 65, as tarefas após a criação do alerta simulado são idênticas às tarefas executadas em caso de ocorrência de eventos de alerta reais.

No que diz respeito às tarefas de associação e dissociação ao *cluster* o processo de entrega da mensagem de simulação ao CH decorre da mesma forma que as simulações de alertas.

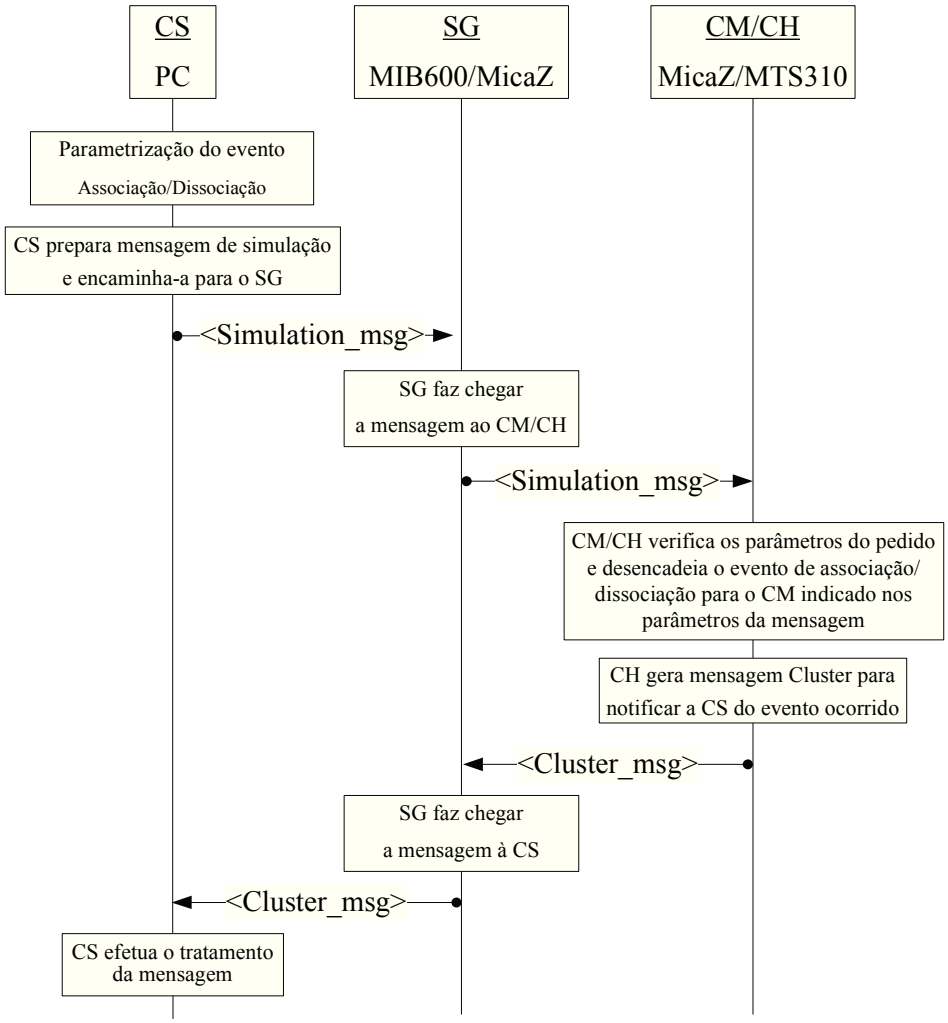

Figura 66 - Testes à Simulação de Associações e Dissociações ao CH

Como se pode verificar na Figura 66, após analisar os parâmetros presentes na mensagem, o CH efetua a associação/dissociação virtual do CM e notifica a CS usando a estrutura de mensagens Cluster.

Dadas as possibilidades de personalização da simulação pretendida, foram executados os seguintes testes de simulação:

5.5.1 Alerta de temperatura de nível Crítico

A Figura 67 representa o pedido de simulação de um alerta de temperatura de nível crítico. Esta parametrização irá provocar a geração de um alerta por parte do CM e posterior notificação à CS. Visto ser um alerta de nível crítico, a MS será também notificada.

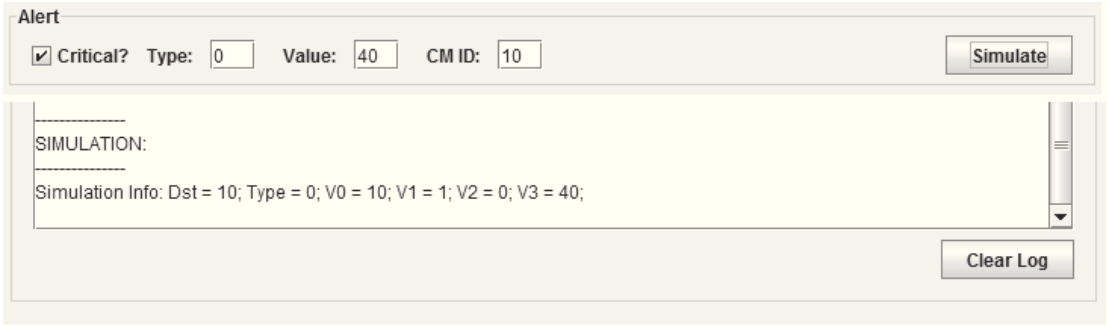

Figura 67 - Simulação de Alerta de Temperatura Crítico

5.5.2 Alerta de temperatura de nível Não-Crítico

Ao contrário do exemplo anterior, esta simulação irá gerar um alerta de temperatura de nível não-critico, estando assim o alerta sujeito a agregação.

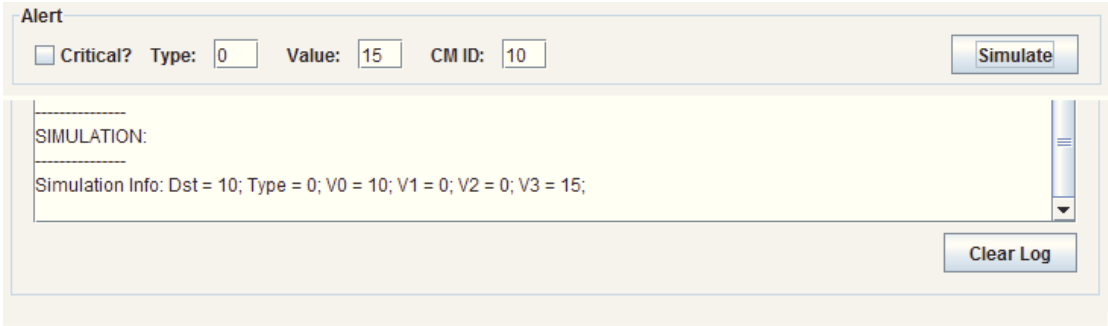

Figura 68 - Simulação de Alerta de Temperatura Não-Crítico

5.5.3 Alerta de movimento de nível Crítico

Na Figura 69 é apresentada a simulação executada com o objetivo de gerar um alerta de movimento de nível crítico. Sendo um alerta de nível crítico, a MS será também notificada.

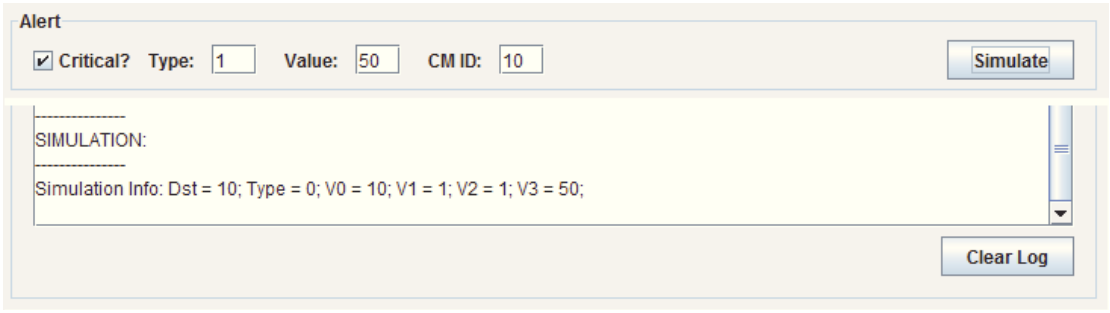

Figura 69 - Simulação de Alerta de Movimento Crítico

5.5.4 Alerta de movimento de nível Não-Crítico

Tal como os alertas de temperatura, também os alertas de movimento com nível não-crítico são agregados pelo CH.

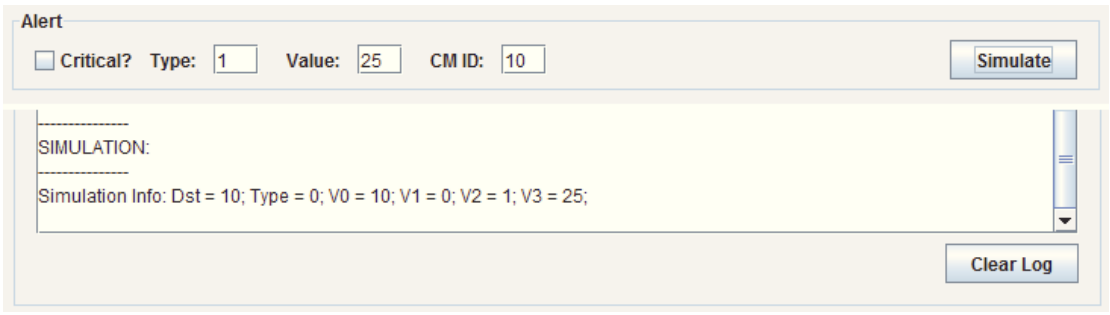

Figura 70 - Simulação de Alerta de Movimento Não-Crítico

5.5.5 Associação de um CM ao CH

A Figura 71 apresenta a simulação de uma associação ao *cluster*. Neste caso foi simulada a associação do CM com o identificador 15 ao CH com o identificador 5. Após a execução do pedido é recebida uma mensagem do CH com a indicação da operação e com o estado do *cluster*, onde se pode ver a presença do CM definido anteriormente. A Figura 66 demonstra as tarefas realizadas no processo de associação.

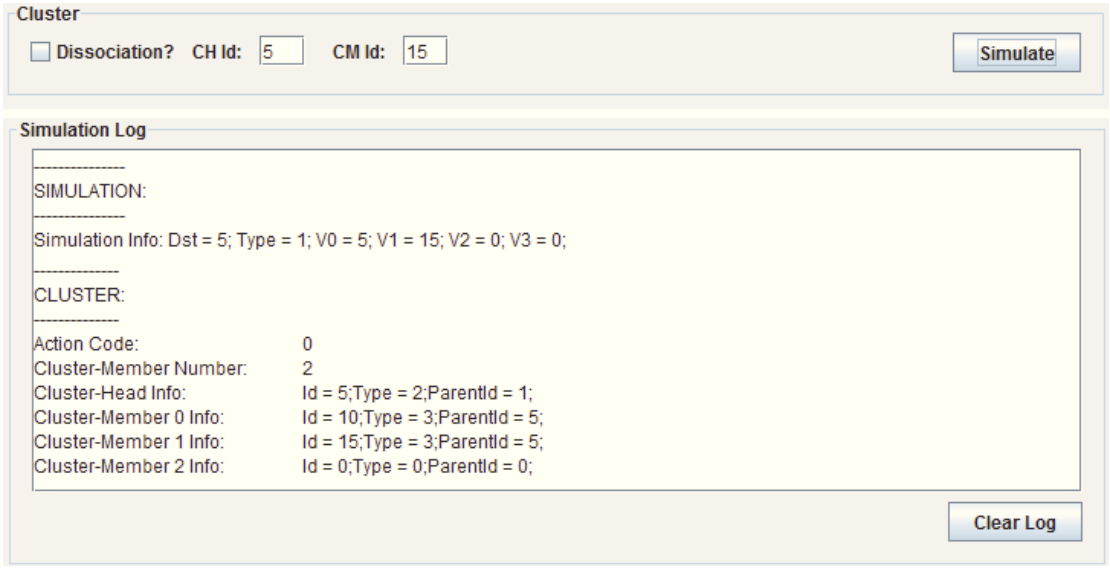

Figura 71 - Simulação de Associação ao Cluster

5.5.6 Dissociação de um CM ao CH

A simulação de uma dissociação ao *cluster* é similar à de uma associação, sendo que, em vez de adição de um novo CM, é neste caso subtraído. Como se pode ver na Figura 72, o CM com o identificador 15 foi removido do cluster e a CS foi notificada após a operação.

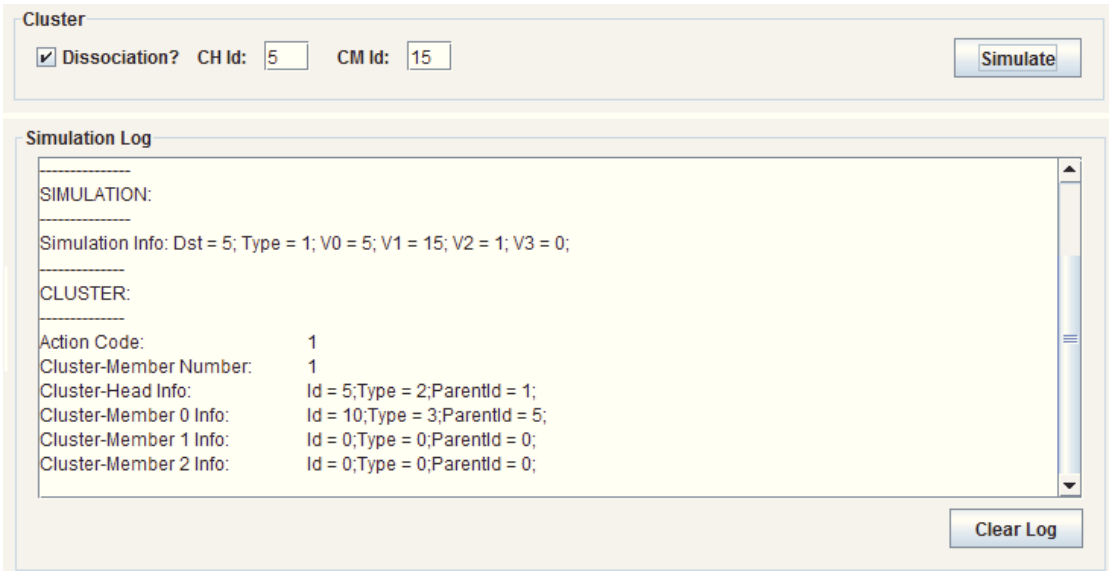

Figura 72 - Simulação de Dissociação ao Cluster

A Figura 66 demonstra as tarefas realizadas no processo de dissociação.

5.5.7 Resultados dos testes de Simulação

Ao executar as simulações apresentadas verificou-se a correta execução das operações pretendidas e posteriores notificações à CS, concluindo-se assim o sucesso nas seguintes tarefas:

- Introdução de parâmetros de simulação através da interface da CS;
- Geração de mensagens de Simulação pela CS e o seu posterior envio até ao SG;
- Envio de mensagens de Simulação desde o SG até ao CM/CH;
- Interpretação e execução da simulação pretendida por parte do CM/CH;
- Geração de mensagens de Cluster pelo CM/CH e o seu posterior envio até ao SG;
- Envio de mensagens de Cluster desde o SG até à CS.
- Apresentação na CS de qual o número de CH e CM presentes na RSSF.

5.6 Aplicações Possíveis

O protótipo apresentado pode ser definido como uma aplicação baseada em Deteção de Eventos (DE) para o caso da deteção de movimento, e como Análise Temporária de um Espaço (ATE) para o caso da análise da temperatura.

Recorrendo ao uso de outros tipos de sensores e atuadores, o protótipo desenvolvido pode servir de base para a criação de aplicações em áreas com finalidades similares, como por exemplo:

- Casas inteligentes (Domótica);
- Localização e rastreamento;
- Monitorização Industrial;
- Monitorização do ambiente envolvente;

5.7 Síntese

Neste capítulo foi apresentado um cenário de testes que tem como objetivo validar a implementação do protótipo e subsequentemente a solução proposta.

Do resultado dos testes efetuados, conclui-se que as entidades definidas pela arquitetura proposta executam com sucesso as principais funcionalizas anunciadas, onde se destacam a aquisição de dados sensoriais, a geração e agregação de alertas e a notificação das estações de recolha de dados (Control-Sation e Main-Station).

Foi possível constatar que o protótipo assenta numa arquitetura fiável e veio provar que desde a arquitetura, passando pela implementação do protótipo e testes efetuados, tornam a aplicação desenvolvida uma base consistente para futuros desenvolvimentos.

6. Conclusão

Neste capítulo são apresentadas as conclusões obtidas após o desenvolvimento da solução apresentada nos capítulos anteriores.

O principal objetivo desta dissertação foi analisar e identificar as principais características de uma Rede Sensorial Sem Fios (RSSF) e a partir dessa análise desenvolver uma especificação que define as principais entidades presentes numa arquitetura de rede que se pretende genérica, escalável e compatível com diversos tipos de aplicações.

Começou-se por definir os requisitos e modelo conceptual da arquitetura, desta forma foi possível a aplicação de um conjunto de métricas de avaliação na análise às diversas topologias possíveis e após avaliação, definir qual a que melhores resultados pode apresentar.

Após o estudo e avaliação das diversas topologias, verificou-se que a topologia *cluster* apresenta o maior número de vantagens nas métricas aplicadas ao modelo conceptual definido, nomeadamente nas métricas de escalabilidade e latência.

Já com a topologia em *cluster* definida como base para a arquitetura pretendida, foi feito um estudo acerca das particularidades da técnica de *clustering*, apresentando as características, os objetivos e as possibilidades, como por exemplo, o processamento na rede ou o balanceamento de carga. Com este estudo conclui-se que são muitas as vantagens que se podem retirar do uso de uma topologia *cluster*, principalmente em aplicações onde o principal objetivo é a recolha massiva de dados e onde o tempo de resposta é um fator importante no sucesso da aplicação.

Após o estudo das técnicas de *clustering* e suas particularidades foi possível identificar as entidades necessárias para a implementação da arquitetura pretendida, onde os elementos Cluster-Member (CM), Cluster-Head (CH), Sink-Gateway (SG) e Control-Station (CS) são a base de uma aplicação para RSSF, podendo em alguns casos recorrer-se ainda a um elemento

Main-Sation (MS) para a interligação de várias RSSF ou para a centralização dos dados.

Acrescentando à análise específica de cada entidade, foram apresentadas diversas tarefas que podem ser aplicadas à arquitetura com o intuito de otimizar a solução num todo, nomeadamente, o uso de filas prioritárias, a agregação de dados ou o balanceamento de carga.

Usando as análises e estudos realizados como guia, foi possível dar início ao desenvolvimento de um protótipo que demonstra a funcionalidade da arquitetura e a comunicação entre as várias entidades. Recorrendo a *hardware* disponível foi desenvolvida uma aplicação sobre o sistema operativo TinyOS onde é possível efetuar a recolha de dados sensoriais e a posterior entrega na CS e MS.

As principais dificuldades enfrentadas durante o desenvolvimento do protótipo foram a curva de aprendizagem necessária para o desenvolvimento de aplicações em TinyOS e o facto do número de MicaZ ser reduzido não possibilitando o uso de um CM dedicado. Para contornar esta situação foi usado um MicaZ com capacidades de CM/CH e desenvolvida uma camada de simulação com o intuito de simular determinados eventos no *cluster*. Apesar destas adversidades, foi possível verificar o sucesso na recolha de dados, na comunicação entre as várias entidades e na entrega dos dados no destino.

Tendo em conta a especificação apresentada, conclui-se que, recorrendo às entidades indicadas e às suas principais funções e capacidades, é possível desenvolver um conjunto de aplicações tendo como base uma arquitetura genérica e escalável, nomeadamente, aplicações na área da domótica, de localização e rastreamento ou de monitorização de espaços.

6.1 Principais Contribuições

Como resultado do trabalho a que esta dissertação diz respeito, destacam-se como principais contribuições a arquitetura modelo constituída pelas várias entidades apresentadas e o protótipo desenvolvido que poderá ser usado como base para o desenvolvimento de novas aplicações.

A primeira contribuição consiste na identificação, estudo e apresentação de diversos temas e opções relevantes para a otimização da arquitetura e performance das aplicações de RSSF. Foi feito o levantamento das questões relacionadas com as vantagens e desvantagens das

principais topologias usadas em RSSF e das principais características que definem as técnicas de *clustering*. Com base no estudo realizado, conclui-se que o uso de filas prioritárias, agregação de dados ou balanceamento de carga otimizam o consumo energético da RSSF e a performance da aplicação.

A segunda contribuição deste trabalho, constituída pela proposta de arquitetura genérica e escalável para RSSF, pode ser usada como diretriz na criação de novas aplicações. Tendo como referência as funções e capacidades de cada uma das entidades será possível planear e desenvolver aplicações de uma maneira mais linear.

Por último, a terceira contribuição consiste no protótipo desenvolvido que pode ser usado como base para o desenvolvimento de aplicações para RSSF. O uso de ferramentas e *software* aberto permitirá o crescimento ou adaptação do protótipo a outras aplicações.

Em resumo, a solução apresentada neste trabalho contribuiu para o fornecimento de uma arquitetura genérica e escalável e na apresentação de diversas características e particularidades que influenciam a performance de uma RSSF.

6.2 Trabalho Futuro

Dada a complexidade e elevado número de entidades presentes na arquitetura e na tentativa de apresentar uma especificação o mais genérica possível, poderá ser importante no futuro olhar mais concretamente para cada uma das entidades, ou a um conjunto de entidades, dependendo da aplicação pretendida.

Alguns dos temas e áreas de investigação importantes para o melhoramento da solução proposta são os seguintes:

- Algoritmos de *clustering*;
- Técnicas e algoritmos de agregação de dados;
- Aplicação de políticas de segurança na gestão do *cluster* e no transporte de dados;
- Uso de políticas de qualidade de serviço (QoS).

Identificando e definindo quais as soluções capazes de contemplar da melhor maneira os

pontos apresentados, será possível tornar a arquitetura proposta uma solução mais preparada para responder às exigências das diversas aplicações usadas em RSSF.

Bibliografia

[1] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[2] "Ember," *[online] www.ember.com (Acedido a 28 de Dezembro de 2012)*. 2012.

[3] "Shimmer," *[online] www.shimmer-research.com (Acedido a 28 de Dezembro de 2012)*. 2012.

[4] "MemSic," *[online] www.memsic.com (Acedido a 28 de Dezembro de 2012)*. 2012.

[5] K. Romer and F. Mattern, "The Design Space of Wireless Sensor Networks," *IEEE Wireless Communications*, 2004.

[6] J. T. Correll, "Igloo White," *Air Force Magazine*, vol. 87, no. 11, 2004.

[7] "Smart Dust," *[online] robotics.eecs.berkeley.edu/~pister/SmartDust/ (Acedido a 28 de Dezembro de 2012)*. 2001.

[8] "NEST," *[online] webs.cs.berkeley.edu/nest-index.html (Acedido a 28 de Dezembro de 2012)*. 2001.

[9] "Alternatives to landmines," *[online] www.scienceblog.com/community/older/2001/C/200113355.html (Acedido a 28 de Dezembro de 2012)*. 2001.

[10] "DARPA SensIT," *[online] basics.eecs.berkeley.edu/sensorwebs/29palms/ (Acedido a 28 de Dezembro de 2012)*. 2001.

[11] "MESSAGE - Mobile Environmental Sensing System Across a Grid Environment," *[online] www.commsp.ee.ic.ac.uk/~wiser/message/ (Acedido a 28 de Dezembro de 2012)*. 2006.

[12] "PermaSense," *[online] www.permasense.ch (Acedido a 28 de Dezembro de 2012)*. 2010.

[13] "GlacsWeb," *[online] glacsweb.org (Acedido a 28 de Dezembro de 2012)*. 2004.

[14] "Wireless Industrial I/O Applications," *[online] www.bannerengineering.com/en-US/wireless/surecross_web_appnotes (Acedido a 28 de Dezembro de 2012)*. 2012.

[15] "Using wireless sensors to monitor bridge safety," *[online] www.physorg.com/news154614946.html (Acedido a 28 de Dezembro de 2012)*. 2012.

[16] "Smart-Clothing," *[online] www.e-projects.ubi.pt/smart-clothing/project.html (Acedido a 28 de Dezembro de 2012)*. 2009.

[17] "Mercury," *[online] fiji.eecs.harvard.edu/Mercury (Acedido a 28 de Dezembro de 2012)*. 2009.

[18] "ITALH," *[online] buffy.eecs.berkeley.edu/PHP/resabs/resabs.php?f_year=2006&f_submit=one&f_absid=101094 (Acedido a 28 de Dezembro de 2012)*. 2005.

[19] "Vital Responder," *[online] www.vitalresponder.pt (Acedido a 28 de Dezembro de 2012)*. 2011.

[20] C. Herring and S. Kaplan, "Component - based software systems for smart environments," *IEEE Personal Communications*, vol. 7, no. 5, pp. 60–61, 2000.

[21] A. J. Goldsmith and S. B. Wicker, "Design challenges for energy - constrained ad hoc wireless networks," *IEEE Wireless Communications*, vol. 9, no. 4, pp. 8–27, 2002.

[22] N. Moreira, M. Venda, C. Silva, L. Marcelino, and A. Pereira, "@Sensor - Mobile Application to Monitor a WSN," in *2011 6th Iberian Conference on Information Systems and Technologies (CISTI)*, 2011, pp. 1–6.

[23] F. Koushanfar, M. Potkonjak, and A. Sangiovanni - Vincentelli, "Fault-Tolerance techniques for ad hoc sensor networks," *Proceedings of IEEE Sensors*, vol. 2, pp. 1491–1496, 2002.

[24] Y. Chong and S. P. Kumar, "Sensor networks: Evolution, opportunities, and challenges," *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1247–1256, 2003.

[25] G. E. Moore, "Cramming More Components onto Integrated Circuits," presented at the IEEE Commun. Mag., 1965, pp. 114–117.

[26] R. F. Pierret, *Introduction to Microelectronic Fabrication*. Menlo Park, CA: Addison - Wesley, 1990.

[27] D. Senturia, *Microsystem Design*. Norwell, MA: Kluwer Academic Publishers, 2001.

[28] F. Zhao and L. Guibas, *Wireless Sensor Networks: An Information Processing Approach*. San Francisco, CA: Morgan Kaufmann Publishers, 2004.

[29] A. Savvides and M. B. Srivastava, "A distributed computation platform for wireless embedded sensing," in *Proceedings of International Conference on Computer Design (ICCD' 02)*, Freiburg, Germany, 2002, pp. 220–225.

[30] A. Chandrakasan, R. Min, M. Bhardwaj, H. Cho, and A. Wang, "Power aware wireless microsensor systems," in *Proceedings of 32nd European Solid - State Device Research Conference (ESSDERC' 02)*, Florence, Italy, 2002, pp. 47–54.

[31] J. M. Khan, R. H. Katz, and K. Pister, "Next century challenges: Mobile networking for smart dust," in *Proceedings of 5th International Conference on Mobile Computing and Networking (MobiCom' 99)*, Seattle, WA, 1999, pp. 271–278.

[32] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proceedings of 36th Annual Symposium on Foundations of Computer Science (FOCS' 95)*, 1995, pp. 374–382.

[33] "Mica2," *[online] www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=149%3Amicaz-oem-edition (Acedido a 29 de Dezembro de 2012)*. 2012.

[34] "MicaZ," *[online] www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=148%3Amicaz (Acedido a 29 de Dezembro de 2012)*. 2012.

[35] “Epic,” *[online] www.cs.berkeley.edu/~prabal/projects/epic/ (Acedido a 29 de Dezembro de 2012)*. 2012.

[36] “Iris,” *[online] www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=135%3Airis (Acedido a 29 de Dezembro de 2012)*. 2012.

[37] “TelosB,” *[online] www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=152%3Atelosb (Acedido a 29 de Dezembro de 2012)*. 2012.

[38] “Sun SPOT,” *Sun SPOT World*. .

[39] “iMote2,” *[online] www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=139%3Aimote2-multimedia (Acedido a 29 de Dezembro de 2012)*. 2012.

[40] “MTS 310 Sensor Board,” *[online] www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=150%3Amts-mda (Acedido a 29 de Dezembro de 2012)*. 2012.

[41] “MTS 420 Sensor Board,” *[online] www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=174%3Amts400_420 (Acedido a 29 de Dezembro de 2012)*. 2012.

[42] “ITS 400 Sensor Board,” *[online] www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=139%3Aimote2-multimedia (Acedido a 29 de Dezembro de 2012)*. 2012.

[43] “IMB 400 Sensor Board,” *[online] www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=139%3Aimote2-multimedia (Acedido a 29 de Dezembro de 2012)*. 2012.

[44] “TinyOS,” *[online] www.tinyos.net (Acedido a 29 de Dezembro de 2012)*. 2012.

[45] “nesC,” *[online] nescc.sourceforge.net (Acedido a 29 de Dezembro de 2012)*. 2012.

[46] “TinyGALS,” *[online] ptolemy.eecs.berkeley.edu/papers/03/TinyGALS (Acedido a 13 de Janeiro de 2013)*. 2013.

[47] P. Levis and D. Culler, “Maté: A Tiny Virtual Machine for Sensor Networks,” San José, CA, 2002, pp. 85–95.

[48] V. R. Adi Mallikarjuna, A. V. U. Phani Kumar, D. Janakiram, and G. Ashok Kumar, “Operating Systems for Wireless Sensor Networks: A Survey,” *International Journal of Sensor Networks*, vol. 5, no. 4, pp. 236–255, 2009.

[49] J. Yannakopoulos and A. Bilas, “CORMOS: a communication-oriented runtime system for sensor networks,” *In The Second European Workshop on Wireless Sensor Networks (EWSN 2005)*, pp. 342–353, 2005.

[50] “MantisOS,” *[online] mantisos.org (Acedido a 29 de Dezembro de 2012)*. 2012.

[51] “ContikiOS,” *[online] www.contiki-os.org (Acedido a 29 de Dezembro de 2012)*. 2012.

[52] S. Hong and T. Kim, “SenOS: state-driven operating system architecture for dynamic sensor node reconfi gurability,” *International Conference on Ubiquitous Computing*, pp. 201–203.

[53] “Nano-RK,” *[online] www.nanork.org (Acedido a 29 de Dezembro de 2012)*. 2012.

[54] T. Sameer, N. B. Abu-Ghazaleh, and W. Heinzelman, “Infrastructure tradeoffs for sensor networks,” *WSNA ’02 Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pp. 49–58.

[55] N. Fonseca, “Implementação de Serviços de Aquisição de Dados numa Rede de Sensores sem Fios,” 2007.

[56] T. Moscibroda and R. Wattenhofer, “Coloring unstructured radio networks,” *SPAA ’05 Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pp. 39–48, 2005.

[57] I. Institute of Electrical and Electronics Engineers, “IEEE Std. 802.15.4 – 2003: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR - WPANs),” in *IEEE Press*, New York, United States, 2003.

[58] “IEEE 802.15.4,” *[online] www.ieee802.org/15/pub/TG4.html (Acedido a 14 de Janeiro de 2013)*. .

[59] “HartComm - WirelessHART - IEC,” *[online] www.hartcomm.org/hcf/news/pr2010/WirelessHART_approved_by_IEC.html (Acedido a 14 de Janeiro de 2013)*. .

[60] “Security in 802.15.4 and ZigBee networks,” *[online] sensor-networks.org/index.php?page=0903503549 (Acedido a 14 de Janeiro de 2013)*. .

[61] “IEEE 1451,” *[online] nist.gov/el/isd/ieee/ieee1451.cfm (Acedido a 13 de Janeiro de 2013)*. 2013.

[62] “IPv6 over Low power WPAN,” *[online] datatracker.ietf.org/wg/6lowpan/charter/ (Acedido a 14 de Janeiro de 2013)*. 2013.

[63] R. Silva, J. Silva, and F. Boavida, “Evaluating 6lowPAN implementations in WSNs,” presented at the Conferência sobre Redes de Computadores, 2009.

[64] “First Standard in Industrial Wireless Series,” *[online] www.isa.org/Template.cfm?Section=Press_Releases5&template=/ContentManagement/ContentDisplay.cfm&ContentID=78964 (Acedido a 14 de Janeiro de 2013)*. .

[65] “ZigBee Specifications,” *[online] www.zigbee.org/Specifications.aspx (Acedido a 14 de Janeiro de 2013)*. .

[66] “Sensor Networks, 802.15.4 vs ZigBee,” *[online] sensor-networks.org/index.php?page=0823123150 (Acedido a 14 de Janeiro de 2013)*. .

[67] “ZigBee Alliance,” *[online] www.zigbee.org (Acedido a 14 de Janeiro de 2013)*. .

[68] “Ad hoc On-Demand Distance Vector (AODV) Routing,” *[online] www.ietf.org/rfc/rfc3561.txt (Acedido a 14 de Janeiro de 2013)*. .

[69] X. Xu, X. Y. Li, X. Mao, S. Tang, and S. Wang, “A delay-efficient algorithm for data aggregation in multihop wireless sensor networks,” presented at the IEEE Trans. Parall. Distrib. Syst., 2011, vol. 22, pp. 163–175.

[70] K. W. Kai-Wei Fan, S. Liu, and P. Sinha, “Structure-free data aggregation in sensor networks,” presented at the IEEE Trans. Mob. Comput., 2007, vol. 6, pp. 929–942.

[71] P. Sausen, M. Spohny, and A. Perkusichy, “Energy Efficient Blind Flooding in Wireless Sensors Networks,” presented at the 34th Annual Conference of IEEE Industrial Electronics, 2008, pp. 1736–1741.

[72] S. Hussain and O. Islam, “An Energy Efficient Spanning Tree Based Multi-Hop Routing in Wireless Sensor Networks,” presented at the Wireless Communications and Networking Conference, 2007, pp. 4383–4388.

[73] W. R. Heinzelman, J. Kulik, and H. Balakrishnan, “Adaptive Protocols for Information Dissemination in Wireless Sensor Networks,” presented at the 5th annual ACM/IEEE international conference on Mobile computing and networking, 1999, pp. 174–185.

[74] J. Kulik, W. Heinzelman, and H. Balakrishnan, “Negotiation-based protocols for disseminating information in wireless sensor networks,” presented at the Wireless Networks, 2002, vol. 8, pp. 169–185.

[75] C. Intanagonwiwat, “Directed Diffusion: An Application-Specific and Data-Centric Communication Paradigm for Wireless Sensor Networks.” 2002.

[76] T. Semong, S. Anokye, Q. Li, and Q. Hu, “Rumor as an Energy-Balancing Multipath Routing Protocol for Wireless Sensor Networks,” presented at the International Conference on New Trends in Information and Service Science, 2009, pp. 754–759.

[77] M. Ye, C. Li, G. Chen, and J. Wu, “EECS: An Energy Efficient Clustering Scheme in Wireless Sensor Networks,” in *24th IEEE International Performance*, 2005, pp. 535–540.

[78] W. P. Chen, J. Hou, and L. Sha, “Dynamic clustering for acoustic target tracking in wireless sensor networks,” presented at the IEEE Trans. Mob. Comput., 2004, vol. 3, pp. 258–271.

[79] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “Energy-Efficient Communication Protocol for Wireless Microsensor Networks,” presented at the 33rd Annual Hawaii International Conference on System Sciences, 2000, p. 10.

[80] L. Yuan, Y. Zhu, and T. Xu, “A Multi-Layered Energy-Efficient and Delay-Reduced Chain-Based Data Gathering Protocol for Wireless Sensor Network,” presented at the IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications, 2008, pp. 13–18.

[81] J. Al-Karaki and A. Kamal, “Routing techniques in wireless sensor networks: A survey,” presented at the IEEE Wirel. Commun., 2004, vol. 11, pp. 6–28.

[82] A. Manjeshwar and D. Agrawal, “TEEN: A Routing Protocol for Enhanced Efficiency in Wireless Sensor Networks,” presented at the 15th International Parallel and Distributed Processing Symposium, 2000, pp. 2009–2015.

[83] J. Choi and C. Lee, “Energy consumption and lifetime analysis in clustered multi-hop wireless sensor networks using the probabilistic cluster-head selection method,” *EURASIP Journal on Wireless Communications and Networking*, 2011.

[84] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “LEACH: Energy-Efficient Communication Protocol for Wireless Microsensor Networks,” presented at the 33rd Hawaii International Conference on System Sciences, 2000.

[85] S. Muruganathan, D. Ma, R. Bhasin, and A. Fapojuwo, “A centralized energy-efficient routing protocol for wireless sensor networks,” presented at the IEEE Commun. Mag., 2005, vol. 43, pp. 8–13.

[86] S. Lindsey and C. Raghavendra, “PEGASIS: Power-Efficient Gathering in Sensor Information Systems,” presented at the IEEE Aerospace Conference Proceedings, 2002, vol. 3, pp. 1125–1130.

[87] N. Tabassum, Q. E. K. Mamun, and Y. Urano, “COSEN: A Chain Oriented Sensor Network for Efficient Data Collection,” presented at the Third International Conference on Information Technology : New Generations, 2006.

[88] K. Chen, J. Huang, and C. Hsiao, “CHIRON: An energy-efficient chain-based hierarchical routing protocol in wireless sensor networks,” presented at the Wireless Telecommunications Symposium, 2009, pp. 1–5.

[89] H. Li, H. Yu, and A. Liu, “A Tree Based Data Collection Scheme for Wireless Sensor Network,” presented at the International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006, p. 119.

[90] A. Woo, T. Tong, and D. Culler, “Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks,” presented at the 1st International conference on Embedded Networked Sensor Systems, 2003, pp. 14–27.

[91] Y. Park and E. S. Jung, “Plus-tree: A routing protocol for wireless wensor networks,” presented at the Adv. Hybrid Inform. Technol., 2000, vol. 4413, pp. 638–646.

[92] D. Messina, M. Ortolani, and G. Lo Re, “A Network Protocol to Enhance Robustness in Tree-Based WSNs Using Data Aggregation,” presented at the IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems, 2007, pp. 1–4.

[93] K. W. Fan, S. Liu, and P. Sinha, “Dynamic forwarding over tree-on-DAG for scalable data aggregation in sensor networks,” presented at the IEEE Trans. Mob. Comput., 2008, vol. 7, pp. 1271–1284.

[94] Z. Zhang and F. Yu, “Performance Analysis of Cluster-Based and Tree-Based Routing Protocols for Wireless Sensor Networks,” presented at the International Conference on Communications and Mobile Computing (CMC), 2010, pp. 418–422.

[95] A. Chandrakasan, R. Min, M. Bhardwaj, S. Cho, and A. Wang, “Power Aware Wireless Microsensor Systems,” presented at the 28th European Solid-State Circuits Conference, 2002, pp. 47–54.

[96] C. Schurgers, G. Kulkarni, and M. Srivastava, “Distributed on-demand address assignment in wireless sensor networks,” presented at the IEEE Trans. Parall. Distrib. Sys., 2002, vol. 13, pp. 1056–1065.

[97] M. Rausand and A. Hoyland, *System Reliability Theory: Models, Statistical Methods, and Applications*. John Wiley & Sons: Hoboken, 2004.

[98] W. K. Lai, C. S. Shieh, and Y. T. Lee, “A Cluster-Based Routing Protocol for Wireless Sensor Networks with Adjustable Cluster Size,” presented at the Fourth International Conference on Communications and Networking, 2009, pp. 1–5.

[99] Y. Kishino, “Data gathering in high-density wireless sensor networks using hierarchical clustering,” presented at the Wireless Communication Systems. 2008. ISWCS ’08., 2008, pp. 547–551.

[100] R. Moraes, C. Ribeiro, and C. Duhamel, “Optimal solutions for fault-tolerant topology control in wireless ad hoc networks,” presented at the IEEE Trans. Wirel. Commun., 2009, vol. 8, pp. 5970–5981.

[101] J. Zheng and A. Jamalipour, *Wireless Sensor Networks: A Networking Perspective*. Wiley, 2009.

[102] O. Younis and S. Fahmy, “HEED: A Hybrid, Energy-Efficient, Distributed clustering approach for Ad Hoc sensor networks,” presented at the IEEE Transactions on Mobile Computing 3, 2004, pp. 366–379.

[103] P. Ding, J. Holliday, and A. Celik, “Distributed energy efficient hierarchical clustering for wireless sensor networks,” presented at the IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS’05), 2005, pp. 322–339.

[104] K. Wang, S. Abu Ayyash, T. D. C. Little, and P. Basu, “Attribute-based clustering for information dissemination in wireless sensor networks,” in *Sensor and Ad Hoc Communications and Networks (SECON 2005)*, 2005.

[105] B. Krishnamachari, D. Estrin, and S. Wicker, “Modeling data centric routing in wireless sensor networks,” presented at the IEEE INFOCOM, 2002.

[106] G. Gupta and M. Younis, “Load-balanced clustering in wireless sensor networks,” presented at the International Conference on Communication (ICC 2003), 2003.

[107] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, “Application specific protocol architecture for wireless microsensor networks,” presented at the IEEE Transactions on Wireless Networking (2002), 2002.

[108] G. Gupta and M. Younis, “Fault-tolerant clustering of wireless sensor networks,” presented at the IEEE Wireless Communication and Networks Conference (WCNC 2003), 2003.

[109] K. Sohrabi, “Protocols for self-organization of a wireless sensor network,” presented at the IEEE Personal Communications 7, 2000, pp. 16–27.

[110] S. Bandyopadhyay and E. Coyle, “An energy efficient hierarchical clustering algorithm for wireless sensor networks,” presented at the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003), 2003.

[111] S. Banerjee and S. Khuller, “A clustering scheme for hierarchical control in multi-hop wireless networks,” presented at the 20th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM’ 01), 2001.

[112] M. Younis and K. Akkaya, “Optimization of task allocation in a cluster–based sensor network,” presented at the 8th IEEE Symposium on Computers and Communications (ISCC’2003), 2003.

[113] F. Dai and J. Wu, “Constructing k-connected k-dominating set in wireless networks,” presented at the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS’05), 2005.

[114] F. Garcia, J. Solano, and I. Stojmenovic, “Connectivity based k-hop clustering in wireless networks,” presented at the Telecommunication Systems 22, 2003, pp. 205–220.

[115] Y. Fernandes and D. Malkhi, “K-clustering in wireless ad-hoc networks,” presented at the 2nd ACM international Workshop on Principles of Mobile Computing (POMC ’02), 2002.

[116] A. D. Amis, R. Prakash, T. H. P. Vuong, and D. T. Huynh, “Max-Min D-cluster formation in wireless ad-hoc networks,” presented at the 20th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM’2000), 2000.

[117] M. Younis, M. Youssef, and K. Arisha, “Energy-aware management in cluster-based sensor networks,” presented at the Computer Networks 43, 2003, pp. 649–668.

[118] E. Ilker Oyman and C. Ersoy, “Multiple sink network design problem in large scale wireless sensor networks,” presented at the IEEE International Conference on Communications (ICC 2004), 2004.

[119] Y. T. Hou, Y. Shi, and H. D. Sherali, “On energy provisioning and relay node placement for wireless sensor networks,” presented at the IEEE Transactions on Wireless Communications, 2005, vol. 4, pp. 2579–2590.

[120] C. R. Lin and M. Gerla, “Adaptive clustering for mobile wireless networks,” *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 7, pp. 1265–1275, 1997.

[121] C. Chiang, H. Wu, W. Liu, and M. Gerla, “Routing in clustered multihop, mobile wireless networks with fading channel,” presented at the IEEE Singapore International Conference on Networks (SICON ’ 07), 1997, pp. 197–211.

[122] M. Chatterjee, S. Das, and D. Turgut, “WCA: A weighted clustering algorithm for mobile ad hoc networks,” *Journal of Cluster Computing*, vol. 5, no. 2, pp. 193–204, 2002.

[123] A. Manjeshwar and D. P. Agrawal, “APTEEN: A Hybrid Protocol for Efficient Routing and Comprehensive Information Retrieval in Wireless Sensor Networks,” in *2nd International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile computing*, 2002, pp. 195–202.

[124] H. Chan and A. Perrig, “ACE: An Emergent Algorithm for Highly Uniform Cluster Formation,” in *1st European Workshop on Sensor Networks (EWSN)*, 2004, pp. 154–171.

[125] S. Jung, Y. Han, and T. Chung, “The Concentric Clustering Scheme for Efficient Energy Consumption in the PEGASIS,” in *9th International Conference on Advanced Communication Technology*, 2007, pp. 260–265.

[126] C. F. Li, M. Ye, G. H. Chen, and J. Wu, “An Energy-Efficient Unequal Clustering Mechanism for Wireless Sensor Networks,” in *2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems Conference (MASS)*, 2005, pp. 596–604.

[127] D. Koutsonikola, S. Das, H. Y. Charlie, and I. Stojmenovic, “Hierarchical geographic multicast routing for wireless sensor networks,” *Wireless Networks*, vol. 16, pp. 449–466, 2010.

[128] L. Buttyan and P. Schaffer, “PANEL: Position-Based Aggregator Node Election in Wireless Sensor Networks,” in *4th IEEE International Conference on Mobile Ad-hoc and Sensor Systems Conference (MASS)*, 2007, pp. 1–9.

[129] V. Loscri, G. Morabito, and S. Marano, “A Two-Level Hierarchy for Low-Energy Adaptive Clustering Hierarchy,” in *2nd IEEE Semiannual Vehicular Technology Conference*, 2005, pp. 1809–1813.

[130] H. Luo, F. Ye, J. Cheng, S. Lu, and L. Zhang, “TTDD: Two-tier data dissemination in large-scale wireless sensor networks,” *Wireless Networks*, 2005.

[131] S. Soro and W. Heinzelman, “Prolonging the Lifetime of Wireless Sensor Networks via Unequal Clustering,” in *5th IEEE International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN)*, 2005, pp. 236–243.

[132] C. Karlof and D. Wagner, “Secure routing in wireless sensor networks: Attacks and countermeasures,” *Ad-Hoc Networks*, vol. 1, pp. 293–315, 2003.

[133] A. D. Wood and J. A. Stankovic, “Denial of service in sensor networks,” *IEEE Computer*, vol. 35, no. 10, pp. 54–62, 2002.

[134] S. Zhu, S. Setia, and S. Jajodia, “LEAP: Efficient security mechanisms for large - scale distributed sensor networks,” presented at the 10th ACM conference on Computer and Communication Security, 2003, pp. 62–72.

[135] H. Chan, A. Perrig, and D. Song, “Random key predistribution schemes for sensor networks,” presented at the IEEE Symposium on Security and Privacy (S & P ’ 03), 2003, pp. 197–213.

[136] W. Du, J. Deng, Y. S. Han, P. K. Varshney, J. Katz, and A. Khalili, “A pairwise key pre-distribution scheme for wireless sensor networks,” *ACM Transactions on Information and System Security*, vol. 8, no. 2, pp. 228–258, 2005.

[137] J. Hwang and Y. Kim, “Revisiting random key predistribution schemes for wireless sensor networks,” presented at the 2nd ACM workshop on Security of ad-hoc and sensor networks, 2004, pp. 43–52.

[138] R. Barry and M. Danny, “Sinkholes: A Swiss Army Knife ISP Security Tool.” .

[139] “TinyOS - MIB510 and MIB600 Boards,” *TinyOS - SerialForwarder*, 2013. [Online]. Available: http://www.tinyos.net/dist-2.0.0/tinyos-2.x/doc/html/tutorial/programmers.html. [Accessed: 25-Sep-2013].

[140] “SerialForwarder,” *TinyOS - SerialForwarder*, 2013. [Online]. Available: http://www.tinyos.net/dist-2.0.0/tinyos-2.0.2/doc/javadoc/net/tinyos/sf/SerialForwarder.html. [Accessed: 25-Sep-2013].

[141] “JavaMail API,” *JavaMail API*, 2013. [Online]. Available: http://www.oracle.com/technetwork/java/javamail/index.html. [Accessed: 25-Sep-2013].

[142] “Oracle VM VirtualBox,” *Oracle VM VirtualBox*, 2013. [Online]. Available: www.virtualbox.org. [Accessed: 25-Sep-2013].

[143] “Importing a VDI in VirtualBox,” *Importing a VDI in VirtualBox*, 2013. [Online]. Available: https://blogs.oracle.com/oswald/entry/importing_a_vdi_in_virtualbox. [Accessed: 25-Sep-2013].

[144] “VirtualBox Guest Additions,” *VirtualBox Guest Additions*, 2013. [Online]. Available: http://www.virtualbox.org/manual/ch04.html. [Accessed: 25-Sep-2013].

Esta página foi intencionalmente deixada em branco

Anexos

A. Instalação e Execução do Protótipo

O anexo A descreve as tarefas necessárias para a configuração e execução do protótipo.

A.1 Máquina Virtual

Todo o *software* necessário para a execução do protótipo está presente numa máquina virtual. Durante o desenvolvimento foi o usado Oracle VM VirtualBox [142] como *software* de virtualização.

Junto com o material fornecido, encontra-se o VDI (Virtual Disk Image) com o nome `S2S_v1.0.vdi`, esta é a imagem que deve ser importada no VirtualBox. As instalações de importação podem ser seguidas na documentação fornecida pela Oracle [143].

Nota: Recomenda-se a reinstalação do pacote *"Guest Additions"* adequado à versão de VirtualBox usada para que se possa usar uma resolução gráfica mais elevada [144].

A.2 Configuração da Máquina Virtual

No que diz respeito às configurações da máquina virtual, para além da definição do sistema operativo como Linux – Debian, são essenciais o uso de uma placa de rede em modo *bridged* com a placa de rede do *host* e uma placa em modo NAT para posterior comunicação com a placa MIB600 e com a Internet, respetivamente. As restantes configurações podem ser deixadas com os valores por omissão.

Após a execução da máquina virtual, o utilizador será levado até uma *prompt* de autenticação. As contas de utilizador existentes são `root` e `s2s`, para ambos a *password* definida é `s2s`. Sempre que possível, recomenda-se o uso da conta `s2s` para a execução dos comandos apresentados.

Após a autenticação do utilizador `s2s` deverá de ser lançado o ambiente gráfico do sistema operativo, para isso bastar executar `startx` na *shell* disponível. Na pasta `/home/s2s` pode ser encontrado o código fonte para as aplicações Cluster-Member/Cluster-Head (CM/CH), Sink-Gateway (SG) e Control-Station (CS).

O primeiro passo a realizar após a entrada no ambiente gráfico será configurar a conectividade com a placa MIB600, para tal deverá ser configurado um endereço IP na mesma rede em que a MIB600 se encontra.

```
[root][~/] ifconfig eth0 10.0.0.101/24 up
[root][~/] dhclient eth1
```

Após testar a conectividade IP com a MIB600, o código das aplicações CM/CH e SG poderá ser instalado nos MicaZ.

A.3 Instalação e execução das Aplicações CH, SG e CS

Para compilação e instalação das aplicações CM/CH e SG nos MicaZ devem ser executados os seguintes comandos:

- CH

```
[s2s][~/CH] make micaz install,5 eprb,10.0.0.100
```

- SG

```
[s2s][~/SG] make micaz install,1 eprb,10.0.0.100
```

Após a execução destes comandos os MicaZ estão equipados com as aplicações TinyOS desenvolvidas. O próximo passo será a execução da aplicação CS, para tal deve ser executado o seguinte comando:

```
[s2s][~/CS] java -jar dist/CS.jar
```

Após a execução deste comando, o utilizador deverá ter acesso ao painel de autenticação da aplicação CS. As credenciais do utilizador `s2s` devem ser utilizadas. As configurações do SerialForwarder (SF) devem ser adaptadas à configuração IP da MIB600.

Se todas as configurações necessárias foram executadas corretamente, ao ligar o elemento CM/CH (MicaZ/MTS310) será visível a receção de um alerta de movimento, confirmando assim a conectividade entre Redes Sensoriais Sem Fios (RSSF) e a CS.

Nota 1: Quando a aplicação CS é executada a partir de uma consola, podem ser visualizadas

as mensagens de depuração geradas no *output* da execução. Estas mensagens podem ajudar o utilizador a diagnosticar alguma situação de erro.

Nota 2: Por omissão, a aplicação CS não contém endereços de *email* definidos para notificação de alertas. Deverão ser configurados endereços para entrega caso se pretenda usufruir da notificação de alertas à Main-Station (MS).

Esta página foi intencionalmente deixada em branco

B. Listagem do Código CM/CH e SG

O anexo B apresenta o código C/nesC para as aplicações TinyOS desenvolvidas. A documentação, no formato *javadocs*, para as aplicações CH, SG e CS pode ser encontrada junto do material fornecido.

B.1 Header File

```
/*****************************************************************************
 *      s2s.h - Sensor-2-Sensor Header File - 2013/09/30
 *
 *      Developed by Nuno Moreira     2101311@my.ipleiria.pt
 *
 *****************************************************************************
 *
 *      Copyright (C) 2013  Nuno Moreira
 *
 *      This program is free software: you can redistribute it and/or modify
 *      it under the terms of the GNU General Public License as published by
 *      the Free Software Foundation, either version 3 of the License, or
 *      (at your option) any later version.
 *
 *      This program is distributed in the hope that it will be useful,
 *      but WITHOUT ANY WARRANTY; without even the implied warranty of
 *      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *      GNU General Public License for more details.
 *
 *      You should have received a copy of the GNU General Public License
 *      along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 *****************************************************************************/

#ifndef S2S_H
#define S2S_H

enum {
        MAIN_LOOP_INTERVAL = 1000,    /* Check Time Interval - 1000 (miliseconds) */
        MAX_CM_NUMBER = 2,            /* Cluster-Member Array Size - 2 */
        MAX_AGG_NUMBER = 3,           /* Non-Critical Message Aggregation Number - 3 */

        /* Threshold for Temperature °C - ADC
        35 - 555
        36 - 567
        37 - 580
        37.5 - 586
        38 - 592
        39 - 604
```

```
       40 - 615 */
    TEMP_THRESHOLD = 586,     /* Temperature Alert Value - 37.5C */
       TEMP_ALERT_LEVEL = 0, /* Alert Level Identifier -1 */
       TEMP_ALERT_TYPE = 0,  /* Alert Type Identifier - 0 */
       /* Threshold for Accelaration variance */
    ACCEL_THRESHOLD = 10,                   /* Movement Threshold - 10 */
       ACCEL_ALERT_LEVEL = 1,               /* Alert Level Identifier - 1 */
       ACCEL_ALERT_TYPE = 1,                /* Alert Type Identifier - 1 */
       ACCEL_SAMPLES = 20,                  /* Accelerometer Samples - 20 */
    ACCEL_SAMPLE_INTERVAL = 1000,    /* Interval between Samples - 1000 (miliseconds)*/

       CS_TYPE = 0,   /* Node Type Identifier - Control-Station - 0 */
       SG_TYPE = 1,   /* Node Type Identifier - Sink-Gateway - 1 */
       CH_TYPE = 2,   /* Node Type Identifier - Cluster-Head - 2 */
       CM_TYPE = 3,   /* Node Type Identifier - Cluster-Member - 3 */

       CS_ADDR = 0,   /* Node Identifier - Control-Station - 0 */
       SG_ADDR = 1,    /* Node Identifier - Sink-Gateway - 1 */
       CH_ADDR = 5,    /* Node Identifier - Cluster-Head - 5 */
       CM_ADDR = 10,   /* Node Identifier - Cluster-Member - 10 */

       SIM_ALERT = 0,        /* Simulation Action Identifier - Alert Report - 0 */
       SIM_CLUSTER = 1,      /* Simulation Action Identifier - Cluster Management - 1 */


       AM_AGGREGATION = 11,  /* Active-Message Identifier - Aggregation Messages - 11 */
       AM_ALERT = 22,        /* Active-Message Identifier - Alert Messages - 22 */
       AM_CLUSTER = 33,      /* Active-Message Identifier - Cluster Messages - 33 */
       AM_SETTINGS = 44,     /* Active-Message Identifier - Settings Messages - 44 */
       AM_SIMULATION = 55    /* Active-Message Identifier - Simulattion Messages - 55 */
};

/*
*      Node Struct
*
*      Node Identifier, Node Type, Node Parent Identifier;
*
*/
typedef nx_struct node {
    nx_uint8_t nodeId;
       nx_uint8_t type;
       nx_uint8_t parentId;
} node_t;

/*
*      Alert Struct - Network Message (Outgoing)
*
*      Message Destination Address, Alert Creation Node Information, Alert Level, Alert Type,
Alert Value;
*
*/
```

```
typedef nx_struct alert {
        nx_uint8_t msgDst;
    node_t nodeInfo;
        nx_uint8_t level;
        nx_uint8_t type;
        nx_uint8_t value;
}alert_t;

/*
*       Aggregarion Struct - Network Message (Outgoing)
*
*       Message Destination Address, Message Number in Aggregation,
*       Cluster-Head Info, Alert Message 0, Alert Message 1, Alert Message 2;
*
*/
typedef nx_struct aggregation {
        nx_uint8_t msgDst;
        nx_int8_t alertCount;
        node_t nodeInfo;
        alert_t alert0;
        alert_t alert1;
        alert_t alert2;
}aggregation_t;

/*
*       Cluster Struct - Network Message (Outgoing)
*
*       Message Destination Address, Cluster Action Code, Cluster Cluster-Member Number,
*       Cluster Head Info, Cluster-Member 0 Information, Cluster-Member 1 Information,
*       Cluster-Member 2 Information;
*
*/
typedef nx_struct cluster {
        nx_uint8_t msgDst;
        nx_int8_t infoCode;
        nx_int8_t memberNumber;
        node_t nodeInfo;
        node_t node0;
        node_t node1;
        node_t node2;
}cluster_t;

/*
*       Settings Struct - Network Message (Incoming)
*
*       Message Destination Address, Update Type, Update Value;
*
*/
typedef nx_struct settings {
    nx_uint8_t msgDst;
        nx_uint8_t type;
```

```
        nx_uint8_t value;
} settings_t;

/*
*       Simulation Struct - Network Message  (Incoming)
*
*       Message Destination Address, Simulation Type Identifier,
*       Simulation Value Info 0, Simulation Value Info 1,
*       Simulation Value Info 2, Simulation Value Info 3;
*
*/
typedef nx_struct simulation {
    nx_uint8_t msgDst;
        nx_uint8_t type;
        nx_uint8_t value0;
        nx_uint8_t value1;
        nx_uint8_t value2;
        nx_uint8_t value3;
}simulation_t;

#endif
```

B.2 Código da aplicação CM/CH

O código fonte da aplicação CM/CH é apresentado nos seguintes pontos.

B.2.1 Makefile

```
#*****************************************************************************
#       Makefile - Sensor-2-Sensor Cluster-Head Makefile - 2013/09/30
#
#       Developed by Nuno Moreira     2101311@my.ipleiria.pt
#
#*****************************************************************************
#
#       Copyright (C) 2013  Nuno Moreira
#
#       This program is free software: you can redistribute it and/or modify
#       it under the terms of the GNU General Public License as published by
#       the Free Software Foundation, either version 3 of the License, or
#       (at your option) any later version.
#
#       This program is distributed in the hope that it will be useful,
#       but WITHOUT ANY WARRANTY; without even the implied warranty of
#       MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#       GNU General Public License for more details.
#
#       You should have received a copy of the GNU General Public License
#       along with this program.  If not, see <http://www.gnu.org/licenses/>.
#
```

```
#*****************************************************************************/

SENSORBOARD = mts300
PFLAGS += -DMTS310CA
PFLAGS += -I%T/lib/net/ctp -I%T/lib/net -I%T/lib/net/4bitle -I%T/lib/net/drip

# Printf
CFLAGS += -DPRINTF_BUFFER_SIZE=250
CFLAGS += -I$(TOSDIR)/lib/printf
CFLAGS += -DNEW_PRINTF_SEMANTICS

COMPONENT=ChAppC
S2S_H=s2s.h

include $(MAKERULES)
```

B.2.2 ChAppC.nc

```
/*****************************************************************************
 *      ChAppC - Sensor-2-Sensor Cluster-Head, Configuration File - 2013/09/30
 *
 *      Developed by Nuno Moreira     2101311@my.ipleiria.pt
 *
 *****************************************************************************
 *
 *      Copyright (C) 2013  Nuno Moreira
 *
 *      This program is free software: you can redistribute it and/or modify
 *      it under the terms of the GNU General Public License as published by
 *      the Free Software Foundation, either version 3 of the License, or
 *      (at your option) any later version.
 *
 *      This program is distributed in the hope that it will be useful,
 *      but WITHOUT ANY WARRANTY; without even the implied warranty of
 *      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *      GNU General Public License for more details.
 *
 *      You should have received a copy of the GNU General Public License
 *      along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 *****************************************************************************/

#include "s2s.h"

configuration ChAppC { }

implementation {
    components
               ChC,
               MainC,
               LedsC,
```

```
        new TimerMilliC() as MainTimer,

        new TempC() as Temp,
        new AccelXStreamC() as Accel,
        SounderC as Sound,

        ActiveMessageC,
        new AMSenderC(AM_AGGREGATION) as AggregationRadioSend,
        new AMSenderC(AM_ALERT) as AlertRadioSend,
        new AMSenderC(AM_CLUSTER) as ClusterRadioSend,
        new AMSenderC(AM_SETTINGS) as SettingsRadioSend,
        new AMSenderC(AM_SIMULATION) as SimulationRadioSend,
        new AMReceiverC(AM_AGGREGATION) as AggregationRadioReceive,
        new AMReceiverC(AM_ALERT) as AlertRadioReceive,
        new AMReceiverC(AM_CLUSTER) as ClusterRadioReceive,
        new AMReceiverC(AM_SETTINGS) as SettingsRadioReceive,
        new AMReceiverC(AM_SIMULATION) as SimulationRadioReceive;


        ChC.Boot->MainC.Boot;
        ChC.Leds->LedsC;
        ChC.MainLoop->MainTimer;

        ChC.Temp->Temp;
        ChC.Accel->Accel;
        ChC.Sound->Sound;

        ChC.RadioControl->ActiveMessageC;
        ChC.AggregationRadioSend->AggregationRadioSend;
        ChC.AlertRadioSend->AlertRadioSend;
        ChC.ClusterRadioSend->ClusterRadioSend;
        ChC.SettingsRadioSend->SettingsRadioSend;
        ChC.SimulationRadioSend->SimulationRadioSend;
        ChC.AggregationRadioReceive->AggregationRadioReceive;
        ChC.AlertRadioReceive->AlertRadioReceive;
        ChC.ClusterRadioReceive->ClusterRadioReceive;
        ChC.SettingsRadioReceive->SettingsRadioReceive;
        ChC.SimulationRadioReceive->SimulationRadioReceive;
}
```

B.2.3 ChC.nc

```
/*****************************************************************************
 *      ChC - Sensor-2-Sensor Cluster-Head, Implementation File - 2013/09/30
 *
 *      Developed by Nuno Moreira     2101311@my.ipleiria.pt
 *
 *****************************************************************************
 *
 *      Copyright (C) 2013  Nuno Moreira
 *
 *      This program is free software: you can redistribute it and/or modify
```

```
 *      it under the terms of the GNU General Public License as published by
 *      the Free Software Foundation, either version 3 of the License, or
 *      (at your option) any later version.
 *
 *      This program is distributed in the hope that it will be useful,
 *      but WITHOUT ANY WARRANTY; without even the implied warranty of
 *      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *      GNU General Public License for more details.
 *
 *      You should have received a copy of the GNU General Public License
 *      along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 ********************************************************************************/

#include "s2s.h"



module ChC {
    uses {
        /* Operating System Interfaces */
               interface Boot as Boot;
               interface Leds as Leds;
               interface Timer<TMilli> as MainLoop;

               /* Sensing Interfaces */
               interface Read<uint16_t> as Temp;
               interface ReadStream<uint16_t> as Accel;
               interface Mts300Sounder as Sound;

               /* Network Interfaces */
        interface SplitControl as RadioControl;

               interface AMSend as AlertRadioSend;
               interface AMSend as AggregationRadioSend;
               interface AMSend as SimulationRadioSend;
               interface AMSend as ClusterRadioSend;
               interface AMSend as SettingsRadioSend;

               interface Receive as AlertRadioReceive;
               interface Receive as AggregationRadioReceive;
               interface Receive as SimulationRadioReceive;
               interface Receive as ClusterRadioReceive;
               interface Receive as SettingsRadioReceive;
    }
}

implementation {
       /* Auxiliary */
       uint8_t i = 0;
       bool radioBusy;              /* Radio Access Semaphore */
```

```
	/* Aggregation Variables */
	aggregation_t aggregationInfo;	/* Aggregation Info */
	uint8_t aggCounter = 0;

	/* Alert Variables */
	alert_t alertInfo;			/* Alert Info */

	/* Cluster Variables */
	cluster_t clusterInfo;		/* Cluster Info */
	node_t headInfo;			/* Cluster-Head Info */
	node_t memberInfo;			/* Cluster-Member Info */
	node_t emptyNode;			/* Empty Node */
	uint8_t cmCounter = 0;		/* Number of Associated Members */

	/* Settings Variables */
	settings_t settingsInfo;	/* Settings Info */

	/* Simulation Variables */
	simulation_t simulationInfo;  /* Simulation Info */

	/* Sensoring - Temp */
	uint16_t tempThreshold = TEMP_THRESHOLD;


	/* Sensoring - Accel */
	uint16_t vcAccelXAxis[ACCEL_SAMPLES] = {0};			/* Array to store X
Buffer Values */
	uint16_t myAccelSampleInterval = ACCEL_SAMPLE_INTERVAL;    /* Interval Between Samples
*/
	uint16_t accelThreshold = ACCEL_THRESHOLD;
	uint8_t accelAlertLevel = ACCEL_ALERT_LEVEL;
	uint16_t accelCurrent = 0;
	uint16_t accelPrevious = 0;

	/* Message Type - Structs */
	message_t aggregationMsg;
	message_t alertMsg;
	message_t clusterMsg;
	message_t settingsMsg;
	message_t simulationMsg;

	/* Message Type - Pointers */
	aggregation_t *fwdAggregation;
	aggregation_t *newAggregation;
	alert_t *fwdAlert;
	alert_t *newAlert;
	cluster_t *fwdCluster;
	cluster_t *newCluster;
	settings_t *fwdSettings;
	settings_t *newSettings;
```

```
    simulation_t *fwdSimulation;
    simulation_t *newSimulation;


    /* Utils */
    void amberLed(bool on) {
            if(on) {
                    call Leds.led2On();
            } else {
                    call Leds.led2Off();
            }
    }

    void greenLed(bool on) {
            if(on) {
                    call Leds.led1On();
            } else {
                    call Leds.led1Off();
            }
    }

    void redLed(bool on) {
            if(on) {
                    call Leds.led0On();
            } else {
                    call Leds.led0Off();
            }
    }

    void check(error_t ok) {
     if(ok != SUCCESS) {
                    redLed(TRUE);
            }
}

    void beep(uint16_t time_ms) {
            call Sound.beep(time_ms);
    }


    /* Boot Setup */
    void initClusterHead() {      /* Init Cluster-Head */
            headInfo.nodeId = (uint8_t) TOS_NODE_ID;
            headInfo.type = (uint8_t) CH_TYPE;
            headInfo.parentId = (uint8_t) SG_ADDR;
            /* Create Empty Node for Dissociations */
            emptyNode.nodeId = 0;
            emptyNode.type = 0;
            emptyNode.parentId = 0;
    }
```

```
        void initClusterMember() {    /* Init Cluster-Member */
                memberInfo.nodeId = CM_ADDR;
                memberInfo.type = CM_TYPE;
                memberInfo.parentId = headInfo.nodeId;
        }

        void initClusterInfo() {      /* Init Cluster-Info */
                clusterInfo.nodeInfo = headInfo;
                clusterInfo.memberNumber = 0;
        }

        void initAggregationInfo() {  /* Init Aggregation-Info */
                aggregationInfo.nodeInfo = headInfo;
                aggregationInfo.alertCount = 0;
        }

        event void Boot.booted() {
         check(call RadioControl.start());
                call MainLoop.startPeriodic(MAIN_LOOP_INTERVAL);
                initClusterHead();
                initClusterMember();
                initClusterInfo();
                initAggregationInfo();
    }


        /* Handlers */
        void handleAggregation(alert_t alert) {
                /* Add new Alert to Aggregation */
                if(aggCounter == 0) { /* First Alert */
                        aggregationInfo.alert0 = alert;
                        aggCounter = 1;
                        return;
                } else if(aggCounter == 1) { /* Second Alert */
                        aggregationInfo.alert1 = alert;
                        aggCounter = 2;
                        return;
                } else if(aggCounter == 2) { /* Third and Send Alert */
                        aggregationInfo.alert2 = alert;
                        aggregationInfo.alertCount = aggCounter+1;
                        fwdAggregation = call AggregationRadioSend.getPayload(&aggregationMsg,
sizeof(aggregation_t));
                        if (fwdAggregation != NULL) {
                                *fwdAggregation = aggregationInfo;
                                radioBusy = TRUE;
                                greenLed(TRUE);
                                check(call AggregationRadioSend.send(SG_ADDR, &aggregationMsg,
sizeof *fwdAggregation));
                        }
                        aggCounter = 0;
                        return;
```

```
		}
	}

	void handleAlert(node_t cmInfo, uint8_t alertLevel, uint8_t alertType, uint8_t
alertValue) {
		/* (cmInfo, alertLevel, alertType, alertValue) */
		alertInfo.nodeInfo = cmInfo;
		alertInfo.level = alertLevel;
		alertInfo.type = alertType;
		alertInfo.value = alertValue;
		if(alertLevel == 1) { /* Critical, Send ASAP */
			fwdAlert = call AlertRadioSend.getPayload(&alertMsg, sizeof(alert_t));
			if (fwdAlert != NULL) {
				*fwdAlert = alertInfo;
				radioBusy = TRUE;
				greenLed(TRUE);
				check(call AlertRadioSend.send(SG_ADDR, &alertMsg, sizeof
*fwdAlert));
			}
		} else if(alertLevel == 0){ /* Non-Critical, Add to Aggregation */

			handleAggregation(alertInfo);
		}
	}

	void handleCluster(node_t member, uint8_t action) {
		if(action == 0) { /* Association */
			/* Add New Member if Possible */
			if(cmCounter == 0) {
				clusterInfo.node0 = member;
				cmCounter = 1;
				clusterInfo.memberNumber = cmCounter;
				clusterInfo.infoCode = 0;
			} else if(cmCounter == 1) {
				clusterInfo.node1 = member;
				cmCounter = 2;
				clusterInfo.memberNumber = cmCounter;
				clusterInfo.infoCode = 0;
			} else if(cmCounter == 2) {
				clusterInfo.node2 = member;
				cmCounter = 3;
				clusterInfo.memberNumber = cmCounter;
				clusterInfo.infoCode = 0;
			} else if(cmCounter == 3) {
				clusterInfo.infoCode = 2;     /* Notify Max Member */
			}
		} else if (action == 1) { /* Dissociation */
			/* Remove Member */
			if(clusterInfo.node0.nodeId == member.nodeId) {
				clusterInfo.node0 = clusterInfo.node1;
				clusterInfo.node1 = clusterInfo.node2;
```

```
                        clusterInfo.node2 = emptyNode;
                        clusterInfo.infoCode = 1;
                        cmCounter--;
                        clusterInfo.memberNumber = cmCounter;
                } else if(clusterInfo.node1.nodeId == member.nodeId) {
                        clusterInfo.node1 = clusterInfo.node2;
                        clusterInfo.node2 = emptyNode;
                        clusterInfo.infoCode = 1;
                        cmCounter--;
                        clusterInfo.memberNumber = cmCounter;

                } else if(clusterInfo.node2.nodeId == member.nodeId) {
                        clusterInfo.node2 = emptyNode;
                        clusterInfo.infoCode = 1;
                        cmCounter--;
                        clusterInfo.memberNumber = cmCounter;
                } else {
                        clusterInfo.infoCode = 3;     /* Cluster-Member Not Found */
                }
        }
        /* Send clusterInfo do SG */
        fwdCluster = call ClusterRadioSend.getPayload(&clusterMsg, sizeof(cluster_t));
        if (fwdCluster != NULL) {
                *fwdCluster = clusterInfo;
                radioBusy = TRUE;
                greenLed(TRUE);
                check(call ClusterRadioSend.send(SG_ADDR, &clusterMsg, sizeof
*fwdCluster));
        }
    }

    void handleSettings(settings_t *settings) {
        /* Type: 0 -> Temp; 1 -> Accel */
        uint16_t newValue = settings->value;
        beep(newValue);
        switch (settings->type) {
                case 0: tempThreshold = newValue;
                case 1: accelThreshold = newValue;
                default: return;
        }
        beep(100);
    }

    void handleSimulation(simulation_t *simulation) {
        /* Alert:      msgDst, simType, 0: clusterMember, 1: alertLevel, 2: alertType,
3: alertValue */
        /* Cluster: msgDst, simType, 0: chId, 1: cmId, 2: action, 3: unused */
        if(simulation->type == 0) {   /* 0: Alert */
                        memberInfo.nodeId = simulation->value0;
                        memberInfo.type = CM_TYPE;
                        memberInfo.parentId = CH_ADDR;
```

```
                            /* (cmInfo, alertLevel, alertType, alertValue) */
                            handleAlert(memberInfo, simulation->value1, simulation->value2,
simulation->value3);
                } else if (simulation->type == 1) { /* 1: Cluster */

                            /* Create Cluster-Member Info based on Simulation Request */
                            memberInfo.nodeId = simulation->value1;
                            memberInfo.type = CM_TYPE;
                            memberInfo.parentId = simulation->value0;
                            /* (Assoc, memberInfo) */
                            handleCluster(memberInfo, simulation->value2);
                }
                initClusterMember(); /* Reset CM */
        }


        /* Networking - Init */
    event void RadioControl.startDone(error_t error) {
                check(error);
                return;
    }

    event void RadioControl.stopDone(error_t error) {
                check(error);
                return;
        }


        /* Networking - Send */
        event void AggregationRadioSend.sendDone(message_t *msg, error_t error) {
         if (msg == &aggregationMsg) {
             radioBusy = FALSE;
                        greenLed(FALSE);
                        beep(100);
         }
                return;
    }

        event void AlertRadioSend.sendDone(message_t *msg, error_t error) {
         if (msg == &alertMsg) {
             radioBusy = FALSE;
                        greenLed(FALSE);
                        beep(100);
         }
                return;
    }

        event void ClusterRadioSend.sendDone(message_t *msg, error_t error) {
         if (msg == &clusterMsg) {
             radioBusy = FALSE;
                        greenLed(FALSE);
```

```
                beep(100);
        }
            return;
    }

        event void SettingsRadioSend.sendDone(message_t *msg, error_t error) {
        if (msg == &settingsMsg) {
            radioBusy = FALSE;
                greenLed(FALSE);
                beep(100);
        }
            return;
    }

        event void SimulationRadioSend.sendDone(message_t *msg, error_t error) {
        if (msg == &simulationMsg) {
            radioBusy = FALSE;
                greenLed(FALSE);
                beep(100);
        }
            return;
    }


        /* Networking - Receive */
        event message_t *AggregationRadioReceive.receive(message_t* msg, void* payload,
uint8_t len) {
            amberLed(TRUE);
            if (len == sizeof(aggregation_t)) {
                newAggregation = payload;
                if(newAggregation->msgDst == TOS_NODE_ID) {
                    if (len == sizeof(*newAggregation) && !radioBusy) {
                        fwdAggregation = call
AggregationRadioSend.getPayload(&aggregationMsg, sizeof(aggregation_t));

                        if(fwdAggregation != NULL) {
                            *fwdAggregation = *newAggregation;
                            //handler
                        }
                    }
                }
            }
            amberLed(FALSE);
            beep(100);
            return msg;
        }

        event message_t *AlertRadioReceive.receive(message_t* msg, void* payload, uint8_t len)
{
            amberLed(TRUE);
            if (len == sizeof(alert_t)) {
```

```
                        newAlert = payload;
                        if(newAlert->msgDst == TOS_NODE_ID) {
                                if (len == sizeof(*newAlert) && !radioBusy) {
                                        fwdAlert = call AlertRadioSend.getPayload(&alertMsg,
sizeof(alert_t));
                                        if(fwdAlert != NULL) {
                                                *fwdAlert = *newAlert;
                                                //handler
                                        }
                                }
                        }
                }
                amberLed(FALSE);
                beep(100);
                return msg;
    }

        event message_t *ClusterRadioReceive.receive(message_t* msg, void* payload, uint8_t
len) {
                amberLed(TRUE);
                if (len == sizeof(cluster_t)) {
                        newCluster = payload;
                        if(newCluster->msgDst == TOS_NODE_ID) {
                                if (len == sizeof(*newCluster) && !radioBusy) {
                                        fwdCluster = call
ClusterRadioSend.getPayload(&clusterMsg, sizeof(cluster_t));
                                        if(fwdCluster != NULL) {
                                                *fwdCluster = *newCluster;
                                                //handler

                                        }
                                }
                        }
                }
                amberLed(FALSE);
                beep(100);
                return msg;
        }

        event message_t *SettingsRadioReceive.receive(message_t* msg, void* payload, uint8_t
len) {
                 amberLed(TRUE);
                 if (len == sizeof(settings_t)) {
                        newSettings = payload;
                        //if(newSettings->msgDst == CH_ADDR
                        //|| newSettings->msgDst == CM_ADDR) {
                                if (len == sizeof(*newSettings) && !radioBusy) {
                                        fwdSettings = call
SettingsRadioSend.getPayload(&settingsMsg, sizeof(settings_t));
                                        if(fwdSettings != NULL) {
                                                *fwdSettings = *newSettings;
```

```
                                        handleSettings(fwdSettings);

                                    }
                                }
                        //}
                }
                amberLed(FALSE);
                beep(100);
                return msg;
    }

        event message_t *SimulationRadioReceive.receive(message_t* msg, void* payload, uint8_t
len) {
                amberLed(TRUE);
                if (len == sizeof(simulation_t)) {
                        newSimulation = payload;
                        if(newSimulation->msgDst == TOS_NODE_ID) {
                                if (len == sizeof(*newSimulation) && !radioBusy) {
                                        fwdSimulation = call
SimulationRadioSend.getPayload(&simulationMsg, sizeof(simulation_t));

                                        if(fwdSimulation != NULL) {
                                                *fwdSimulation = *newSimulation;
                                                handleSimulation(fwdSimulation);

                                        }
                                }
                        }
                }
                amberLed(FALSE);
                beep(100);
                return msg;
        }


        /* Sensoring */
        void tempRead() {
         call Temp.read();
    }

        event void Temp.readDone(error_t error, uint16_t val) {
                /*
                 * Temperature read is always
                 * returning the value 0 (zero)
                 * For testing purpose we will
                 * define the temperature value
                 * as 25 °C
                 */
                val = 25;
                if(error == SUCCESS) {
            if (val > tempThreshold) { /* Report Alert */
```

```
                        beep(100);
                        handleAlert(memberInfo, TEMP_ALERT_LEVEL, TEMP_ALERT_TYPE, val);
                }
        }
    }

        void accelRead() {
        /* Regist our buffers */
        call Accel.postBuffer(vcAccelXAxis, ACCEL_SAMPLES);
        /* Trigger sampling at the desired interval */
        call Accel.read(myAccelSampleInterval);
        }

        event void Accel.readDone(error_t error, uint32_t usActualPeriod) {
                if(error == SUCCESS) {
                        for(accelCurrent = 0, i = 0; i < ACCEL_SAMPLES; i++) {
                                accelCurrent += vcAccelXAxis[i];
                        }
                        accelCurrent /= ACCEL_SAMPLES;
                        if (accelCurrent > accelPrevious + accelThreshold) { /* Report Alert */
                                /* Report Alert */
                                handleAlert(memberInfo, ACCEL_ALERT_LEVEL, ACCEL_ALERT_TYPE,
accelCurrent);
                        }
                        accelPrevious = accelCurrent;
        }
    }

        event void Accel.bufferDone(error_t ok, uint16_t * buf, uint16_t count) {
                //handler
        }

                /* Main Loop */
        event void MainLoop.fired() {
                tempRead();
                accelRead();
    }
}
```

B.3 Código da aplicação Sink-Gateway

O código fonte da aplicação SG é apresentado nos seguintes pontos.

B.3.1 Makefile

```
#*****************************************************************************
#       Makefile - Sensor-2-Sensor Sink-Gateway Makefile - 2013/09/30
#
#       Developed by Nuno Moreira     2101311@my.ipleiria.pt
#
```

```
#*****************************************************************************
#
#       Copyright (C) 2013  Nuno Moreira
#
#       This program is free software: you can redistribute it and/or modify
#       it under the terms of the GNU General Public License as published by
#       the Free Software Foundation, either version 3 of the License, or
#       (at your option) any later version.
#
#       This program is distributed in the hope that it will be useful,
#       but WITHOUT ANY WARRANTY; without even the implied warranty of
#       MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#       GNU General Public License for more details.
#
#       You should have received a copy of the GNU General Public License
#       along with this program.  If not, see <http://www.gnu.org/licenses/>.
#
#*****************************************************************************/

PFLAGS += -I%T/lib/net/ctp -I%T/lib/net -I%T/lib/net/4bitle -I%T/lib/net/drip

# Printf
CFLAGS += -DPRINTF_BUFFER_SIZE=250
CFLAGS += -I$(TOSDIR)/lib/printf
CFLAGS += -DNEW_PRINTF_SEMANTICS

COMPONENT=SgAppC
include $(MAKERULES)
```

B.3.2 SgAppC.nc

```
/*****************************************************************************
 *      SgAppC - Sensor-2-Sensor Sink-Gateway, Configuration File - 2013/09/30
 *
 *      Developed by Nuno Moreira     2101311@my.ipleiria.pt
 *
 *****************************************************************************
 *
 *      Copyright (C) 2013  Nuno Moreira
 *
 *      This program is free software: you can redistribute it and/or modify
 *      it under the terms of the GNU General Public License as published by
 *      the Free Software Foundation, either version 3 of the License, or
 *      (at your option) any later version.
 *
 *      This program is distributed in the hope that it will be useful,
 *      but WITHOUT ANY WARRANTY; without even the implied warranty of
 *      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *      GNU General Public License for more details.
 *
 *      You should have received a copy of the GNU General Public License
```

```
 *      along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 ********************************************************************************/

#include "../CH/s2s.h"

configuration SgAppC { }

implementation {
    components
        SgC,
               MainC,
               LedsC,
               PrintfC,
               new TimerMilliC() as MainTimer,

        SerialActiveMessageC,
               new SerialAMSenderC(AM_AGGREGATION) as AggregationSerialForward,
               new SerialAMSenderC(AM_ALERT) as AlertSerialForward,
               new SerialAMSenderC(AM_CLUSTER) as ClusterSerialForward,
               new SerialAMSenderC(AM_SETTINGS) as SettingsSerialForward,
               new SerialAMSenderC(AM_SIMULATION) as SimulationSerialForward,
               new SerialAMReceiverC(AM_AGGREGATION) as AggregationSerialReceive,
               new SerialAMReceiverC(AM_ALERT) as AlertSerialReceive,
               new SerialAMReceiverC(AM_CLUSTER) as ClusterSerialReceive,
               new SerialAMReceiverC(AM_SETTINGS) as SettingsSerialReceive,
               new SerialAMReceiverC(AM_SIMULATION) as SimulationSerialReceive,

               ActiveMessageC,
               new AMSenderC(AM_AGGREGATION) as AggregationRadioSend,
               new AMSenderC(AM_ALERT) as AlertRadioSend,
               new AMSenderC(AM_CLUSTER) as ClusterRadioSend,
               new AMSenderC(AM_SETTINGS) as SettingsRadioSend,
               new AMSenderC(AM_SIMULATION) as SimulationRadioSend,
               new AMReceiverC(AM_AGGREGATION) as AggregationRadioReceive,
               new AMReceiverC(AM_ALERT) as AlertRadioReceive,
               new AMReceiverC(AM_CLUSTER) as ClusterRadioReceive,
               new AMReceiverC(AM_SETTINGS) as SettingsRadioReceive,
               new AMReceiverC(AM_SIMULATION) as SimulationRadioReceive;


               SgC.Boot -> MainC;
               SgC.Leds -> LedsC;
               SgC.MainLoop->MainTimer;

        SgC.SerialControl -> SerialActiveMessageC;
               SgC.AggregationSerialForward -> AggregationSerialForward;
               SgC.AlertSerialForward -> AlertSerialForward;
               SgC.ClusterSerialForward -> ClusterSerialForward;
               SgC.SettingsSerialForward -> SettingsSerialForward;
               SgC.SimulationSerialForward -> SimulationSerialForward;
```

```
		SgC.AggregationSerialReceive -> AggregationSerialReceive;
		SgC.AlertSerialReceive -> AlertSerialReceive;
		SgC.ClusterSerialReceive -> ClusterSerialReceive;
		SgC.SettingsSerialReceive -> SettingsSerialReceive;
		SgC.SimulationSerialReceive -> SimulationSerialReceive;

		SgC.RadioControl -> ActiveMessageC;
		SgC.AggregationRadioSend -> AggregationRadioSend;
		SgC.AlertRadioSend -> AlertRadioSend;
		SgC.ClusterRadioSend -> ClusterRadioSend;
		SgC.SettingsRadioSend -> SettingsRadioSend;
		SgC.SimulationRadioSend -> SimulationRadioSend;
		SgC.AggregationRadioReceive -> AggregationRadioReceive;
		SgC.AlertRadioReceive -> AlertRadioReceive;
		SgC.ClusterRadioReceive -> ClusterRadioReceive;
		SgC.SettingsRadioReceive -> SettingsRadioReceive;
		SgC.SimulationRadioReceive -> SimulationRadioReceive;
}
```

B.3.3 SgC.nc

```
/*********************************************************************************
 *	SgAppC - Sensor-2-Sensor Sink-Gateway, Implementation File - 2013/09/30
 *
 *	Developed by Nuno Moreira	2101311@my.ipleiria.pt
 *
 *********************************************************************************
 *
 *	Copyright (C) 2013  Nuno Moreira
 *
 *	This program is free software: you can redistribute it and/or modify
 *	it under the terms of the GNU General Public License as published by
 *	the Free Software Foundation, either version 3 of the License, or
 *	(at your option) any later version.
 *
 *	This program is distributed in the hope that it will be useful,
 *	but WITHOUT ANY WARRANTY; without even the implied warranty of
 *	MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *	GNU General Public License for more details.
 *
 *	You should have received a copy of the GNU General Public License
 *	along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 *********************************************************************************/

#include "../CH/s2s.h"
#include "printf.h"

module SgC {
    uses {
		/* Operating System Interfaces */
```

```
        interface Boot as Boot;
        interface Leds as Leds;
               interface Timer<TMilli> as MainLoop;

               /* Network Interfaces (Serial/Ethernet) */
               interface SplitControl as SerialControl;
               interface AMSend as AggregationSerialForward;
               interface AMSend as AlertSerialForward;
               interface AMSend as ClusterSerialForward;
               interface AMSend as SettingsSerialForward;
               interface AMSend as SimulationSerialForward;

               interface Receive as AggregationSerialReceive;
               interface Receive as AlertSerialReceive;
               interface Receive as ClusterSerialReceive;
               interface Receive as SettingsSerialReceive;
               interface Receive as SimulationSerialReceive;

               /* Network Interfaces (Radio) */
               interface SplitControl as RadioControl;

               interface AMSend as AggregationRadioSend;
               interface AMSend as AlertRadioSend;
               interface AMSend as ClusterRadioSend;
               interface AMSend as SettingsRadioSend;
               interface AMSend as SimulationRadioSend;

               interface Receive as AggregationRadioReceive;
               interface Receive as AlertRadioReceive;
               interface Receive as ClusterRadioReceive;
               interface Receive as SettingsRadioReceive;
               interface Receive as SimulationRadioReceive;
    }
}

implementation {

        bool radioBusy;

        uint8_t srcId;
        uint8_t dstId;

        /* Message Type - Structs and Pointers */
        message_t aggregationMsg;
        message_t alertMsg;
        message_t clusterMsg;
        message_t settingsMsg;
        message_t simulationMsg;

        aggregation_t *fwdAggregation;
        aggregation_t *newAggregation;
```

```
        alert_t *fwdAlert;
        alert_t *newAlert;
        cluster_t *fwdCluster;
        cluster_t *newCluster;
        settings_t *fwdSettings;
        settings_t *newSettings;
        simulation_t *fwdSimulation;
        simulation_t *newSimulation;

        void check(error_t ok) {
         if(ok != SUCCESS) {
                        printf("E,CE,-,-,ERROR;"); printfflush();
                }
    }

        /* Boot Setup */
        event void MainLoop.fired() {
                //handler
    }

        event void Boot.booted() {
                check(call SerialControl.start());
         check(call RadioControl.start());
                call MainLoop.startPeriodic(MAIN_LOOP_INTERVAL);
    }

    event void RadioControl.startDone(error_t error) {
                check(error);
                return;
    }

        event void RadioControl.stopDone(error_t error) {
                check(error);
                return;
        }

        event void SerialControl.startDone(error_t error) {
                check(error);
                return;
        }

        event void SerialControl.stopDone(error_t error) {
                check(error);
                return;
        }

        /* Control-Station to Sink-Gateway Messages (Receive using Serial/Ethernet) Handlers
*/
        event  message_t  *AggregationSerialReceive.receive(message_t*  msg,  void*  payload,
uint8_t len) {
                if (len == sizeof(aggregation_t) && !radioBusy) {
```

```
                    printf("I,SR,%u,%u,%u;",     CS_ADDR,     SG_ADDR,     AM_AGGREGATION);
printfflush();
                    fwdAggregation = call AggregationRadioSend.getPayload(&aggregationMsg,
sizeof(aggregation_t));
                    radioBusy = TRUE;
                    check(call AggregationRadioSend.send(CH_ADDR, &aggregationMsg, sizeof
*fwdAggregation));
             } else {
                    printf("E,SR,%u,%u,%u;",     CS_ADDR,     SG_ADDR,     AM_AGGREGATION);
printfflush();
             }
             return msg;
    }

    event message_t *AlertSerialReceive.receive(message_t* msg, void* payload, uint8_t len) {

             if (len == sizeof(alert_t) && !radioBusy) {
                    printf("I,SR,%u,%u,%u;", CS_ADDR, SG_ADDR, AM_ALERT); printfflush();
                    fwdAlert = call AlertRadioSend.getPayload(&alertMsg, sizeof(alert_t));

                    radioBusy = TRUE;
                    check(call AlertRadioSend.send(CH_ADDR, &alertMsg, sizeof *fwdAlert));
             } else {
                    printf("E,SR,%u,%u,%u;", CS_ADDR, SG_ADDR, AM_ALERT); printfflush();

             }
             return msg;
    }

       event message_t *ClusterSerialReceive.receive(message_t* msg, void* payload, uint8_t
len) {
             if (len == sizeof(cluster_t) && !radioBusy) {
                    printf("I,SR,%u,%u,%u;", CS_ADDR, SG_ADDR, AM_CLUSTER); printfflush();
                    fwdCluster     =     call     ClusterRadioSend.getPayload(&clusterMsg,
sizeof(cluster_t));
                    radioBusy = TRUE;
                    check(call    ClusterRadioSend.send(CH_ADDR,    &clusterMsg,    sizeof
*fwdCluster));
             } else {
                    printf("E,SR,%u,%u,%u;", CS_ADDR, SG_ADDR, AM_CLUSTER); printfflush();

             }
             return msg;
    }

       event message_t *SettingsSerialReceive.receive(message_t* msg, void* payload, uint8_t
len) {
             if (len == sizeof(settings_t) && !radioBusy) {
                    printf("I,SR,%u,%u,%u;", CS_ADDR, SG_ADDR, AM_SETTINGS); printfflush();
                    fwdSettings    =    call    SettingsRadioSend.getPayload(&settingsMsg,
sizeof(settings_t));
```

```
                        radioBusy = TRUE;
                        check(call    SettingsRadioSend.send(CH_ADDR,    &settingsMsg,    sizeof
*fwdSettings));
                } else {
                        printf("E,SR,%u,%u,%u;", CS_ADDR, SG_ADDR, AM_SETTINGS); printfflush();

                }
                return msg;
    }

        event   message_t   *SimulationSerialReceive.receive(message_t*   msg,   void*   payload,
uint8_t len) {
                if (len == sizeof(simulation_t) && !radioBusy) {
                        printf("I,SR,%u,%u,%u;",      CS_ADDR,      SG_ADDR,      AM_SIMULATION);
printfflush();
                        newSimulation = payload;
                        fwdSimulation   =   call   SimulationRadioSend.getPayload(&simulationMsg,
sizeof(simulation_t));
                        if (fwdSimulation != NULL) {
                 *fwdSimulation = *newSimulation;
                                radioBusy = TRUE;
                                check(call   SimulationRadioSend.send(CH_ADDR,   &simulationMsg,
sizeof *fwdSimulation));
                        }
                } else {
                        printf("E,SR,%u,%u,%u;",      CS_ADDR,      SG_ADDR,      AM_SIMULATION);
printfflush();
                }
                return msg;
    }

        /*   Sink-Gateway   to   Control-Station   Messages   (Send   Post-Processing   (using
Serial/Ethernet)) Handlers */
        event void AggregationSerialForward.sendDone(message_t *msg, error_t error) {
         if (msg == &aggregationMsg) {
                        printf("I,SF,%u,%u,%u;",      SG_ADDR,      CS_ADDR,      AM_AGGREGATION);
printfflush();
         } else {
                        printf("E,SF,%u,%u,%u;",      SG_ADDR,      CS_ADDR,      AM_AGGREGATION);
printfflush();
                }
                return;
    }

        event void AlertSerialForward.sendDone(message_t *msg, error_t error) {
         if (msg == &alertMsg) {
                        printf("I,SF,%u,%u,%u;", SG_ADDR, CS_ADDR, AM_ALERT); printfflush();
         } else {
                        printf("E,SF,%u,%u,%u;", SG_ADDR, CS_ADDR, AM_ALERT); printfflush();
                }
                return;
```

```
    }

        event void ClusterSerialForward.sendDone(message_t *msg, error_t error) {
        if (msg == &clusterMsg) {
                    printf("I,SF,%u,%u,%u;", SG_ADDR, CS_ADDR, AM_CLUSTER); printfflush();
        } else {
                    printf("E,SF,%u,%u,%u;", SG_ADDR, CS_ADDR, AM_CLUSTER); printfflush();
              }
              return;
    }

        event void SettingsSerialForward.sendDone(message_t *msg, error_t error) {
                if (msg == &settingsMsg) {
                    printf("I,SF,%u,%u,%u;", SG_ADDR, CS_ADDR, AM_SETTINGS); printfflush();
        } else {
                    printf("E,SF,%u,%u,%u;", SG_ADDR, CS_ADDR, AM_SETTINGS); printfflush();
              }
              return;
    }

        event void SimulationSerialForward.sendDone(message_t *msg, error_t error) {
        if (msg == &simulationMsg) {
                    printf("I,SF,%u,%u,%u;",      SG_ADDR,      CS_ADDR,      AM_SIMULATION);
printfflush();
        } else {
                    printf("E,SF,%u,%u,%u;",      SG_ADDR,      CS_ADDR,      AM_SIMULATION);
printfflush();
              }
              return;
    }

        /* Sink-Gateway to Cluster-Head Messages (Send using Radio) Handlers */
        event void AggregationRadioSend.sendDone(message_t *msg, error_t error) {
        if (msg == &aggregationMsg) {
            radioBusy = FALSE;
                    printf("I,RS,%u,%u,%u;",      SG_ADDR,      CH_ADDR,      AM_AGGREGATION);
printfflush();
        } else {
                    printf("E,RS,%u,%u,%u;",      SG_ADDR,      CH_ADDR,      AM_AGGREGATION);
printfflush();
              }
              return;
    }

        event void AlertRadioSend.sendDone(message_t *msg, error_t error) {
        if (msg == &alertMsg) {
            radioBusy = FALSE;
                    printf("I,RS,%u,%u,%u;", SG_ADDR, CH_ADDR, AM_ALERT); printfflush();
        } else {
                    printf("E,RS,%u,%u,%u;", SG_ADDR, CH_ADDR, AM_ALERT); printfflush();
              }
```

```
            return;
    }

        event void ClusterRadioSend.sendDone(message_t *msg, error_t error) {
        if (msg == &clusterMsg) {
            radioBusy = FALSE;
                    printf("I,RS,%u,%u,%u;", SG_ADDR, CH_ADDR, AM_CLUSTER); printfflush();
        } else {
                    printf("E,RS,%u,%u,%u;", SG_ADDR, CH_ADDR, AM_CLUSTER); printfflush();
            }
            return;
    }

        event void SettingsRadioSend.sendDone(message_t *msg, error_t error) {
        if (msg == &settingsMsg) {
            radioBusy = FALSE;
                    printf("I,RS,%u,%u,%u;", SG_ADDR, CH_ADDR, AM_SETTINGS); printfflush();
        } else {
                    printf("E,RS,%u,%u,%u;", SG_ADDR, CH_ADDR, AM_SETTINGS); printfflush();
            }
            return;
    }

        event void SimulationRadioSend.sendDone(message_t *msg, error_t error) {
        if (msg == &simulationMsg) {
            radioBusy = FALSE;
                    printf("I,RS,%u,%u,%u;", SG_ADDR, dstId, AM_SIMULATION); printfflush();
        } else {
                    printf("E,RS,%u,%u,%u;", SG_ADDR, dstId, AM_SIMULATION); printfflush();
            }
            dstId=0;
            return;
    }


        /* Cluster-Head to Sink-Gateway Messages (Receive using Radio) Handlers */
        event   message_t   *AggregationRadioReceive.receive(message_t*   msg,   void*   payload,
uint8_t len) {
        if (len == sizeof(aggregation_t)) {
                    printf("I,RR,%u,%u,%u;",     CH_ADDR,     SG_ADDR,     AM_AGGREGATION);
printfflush();
                    newAggregation = payload;
                    if (len == sizeof(*newAggregation) && !radioBusy) {
                           fwdAggregation                        =                        call
AggregationRadioSend.getPayload(&aggregationMsg, sizeof(aggregation_t));

                           if(fwdAggregation != NULL) {
                                  *fwdAggregation = *newAggregation;
                                  check(call
AggregationSerialForward.send(AM_BROADCAST_ADDR,  &aggregationMsg,  sizeof  *fwdAggregation));
```

```
                        }
                    }
                } else {
                    printf("E,RR,%u,%u,%u;",     CH_ADDR,     SG_ADDR,     AM_AGGREGATION);
printfflush();
                }
                return msg;
        }

        event message_t *AlertRadioReceive.receive(message_t* msg, void* payload, uint8_t len)
{
         if (len == sizeof(alert_t)) {
                    printf("I,RR,%u,%u,%u;",  CH_ADDR,  SG_ADDR,  AM_ALERT);  printfflush();

                    newAlert = payload;
                    if (len == sizeof(*newAlert) && !radioBusy) {
                        fwdAlert     =     call     AlertRadioSend.getPayload(&alertMsg,
sizeof(alert_t));
                        if(fwdAlert != NULL) {
                            *fwdAlert = *newAlert;
                            check(call     AlertSerialForward.send(AM_BROADCAST_ADDR,
&alertMsg, sizeof *fwdAlert));
                        }
                    }
                } else {
                    printf("E,RR,%u,%u,%u;", CH_ADDR, SG_ADDR, AM_ALERT); printfflush();
                }
                return msg;
    }

        event  message_t  *ClusterRadioReceive.receive(message_t*  msg,  void*  payload,  uint8_t
len) {
                if (len == sizeof(cluster_t)) {
                    printf("I,RR,%u,%u,%u;",  CH_ADDR,  SG_ADDR,  AM_CLUSTER);  printfflush();

                    newCluster = payload;
                    if (len == sizeof(*newCluster) && !radioBusy) {
                        fwdCluster    =    call    ClusterRadioSend.getPayload(&clusterMsg,
sizeof(cluster_t));
                        if(fwdCluster != NULL) {
                            *fwdCluster = *newCluster;
                            check(call    ClusterSerialForward.send(AM_BROADCAST_ADDR,
&clusterMsg, sizeof *fwdCluster));
                        }
                    }
                } else {
                    printf("E,RR,%u,%u,%u;", CH_ADDR, SG_ADDR, AM_CLUSTER); printfflush();
                }
                return msg;
        }
```

```
        event message_t *SettingsRadioReceive.receive(message_t* msg, void* payload, uint8_t
len) {
         if (len == sizeof(settings_t)) {
                        printf("I,RR,%u,%u,%u;", CH_ADDR, SG_ADDR, AM_SETTINGS); printfflush();
                        fwdSettings     =     call     SettingsRadioSend.getPayload(&settingsMsg,
sizeof(settings_t));
                        if (fwdSettings != NULL) {
                               check(call          SettingsSerialForward.send(AM_BROADCAST_ADDR,
&settingsMsg, sizeof *fwdCluster));
                        }
                } else {
                        printf("E,RR,%u,%u,%u;", CH_ADDR, SG_ADDR, AM_SETTINGS); printfflush();
                }
                return msg;
    }

        event message_t *SimulationRadioReceive.receive(message_t* msg, void* payload, uint8_t
len) {
                if (len == sizeof(simulation_t)) {
                        printf("I,RR,%u,%u,%u;",      CH_ADDR,      SG_ADDR,      AM_SIMULATION);
printfflush();
                        fwdSimulation   =   call   SimulationRadioSend.getPayload(&simulationMsg,
sizeof(simulation_t));
                        if (fwdSimulation != NULL) {
                 check(call   SimulationSerialForward.send(AM_BROADCAST_ADDR,   &simulationMsg,
sizeof *fwdSimulation));
                        }
                } else {
                        printf("E,RR,%u,%u,%u;",      CH_ADDR,      SG_ADDR,      AM_SIMULATION);
printfflush();
                }
                return msg;
        }
}
```