Instituto Superior de Engenharia do Porto

Aplicação de Internet of Things em casas inteligentes

Serviços de Rede

Vasco Rafael Figueiredo Pereira

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Arquiteturas, Sistemas e Redes

Orientador: Nuno Alexandre Magalhães Pereira

Júri:

Presidente:

Doutor Luís Miguel Moreira Lino Ferreira

Vogais:

Doutor Paulo Manuel Baltarejo de Sousa

Doutor Nuno Alexandre Magalhães Pereira

Porto, Outubro 2014

Aos meus pais, irmãs, família e amigos

Resumo

O foco principal no estudo da *Internet of Things* tem sido a integração de dispositivos digitais com o mundo físico e vice-versa. Os dispositivos inteligentes têm vindo a ganhar uma forte presença na nossa vida diária e cada vez mais, tendem a integrar o sistema de uma casa, automatizando processos comuns como o controlo de temperatura ambiente ou mesmo a percentagem de luminosidade de uma divisão. A visão da IoT contempla um mundo interconectado, recolhendo informações de forma automática e possibilitando a comunicação entre dispositivos. Contudo, as tecnologias existentes para a criação de redes que albergam estes novos dispositivos carecem de padrões bem definidos, dificultando a interoperabilidade entre as diversas soluções existentes.

Neste projeto são estudadas e aplicadas as tecnologias mais promissoras aplicáveis ao paradigma *Internet of Things*, com o objetivo de encontrar um conjunto de protocolos padrão para a implementação de sistemas de automação em casas inteligentes.[1]

Como objetivo final deste projeto, pretende-se criar uma rede de dispositivos com capacidades sensoriais que tenham a capacidade de comunicar com o mundo externo, permitindo o acesso à rede por qualquer tipo de utilizador. Com isso, espera-se caminhar para mais perto da padronização dos protocolos inerentes à IoT e habilitar interoperabilidade entre as mais diversas soluções.

São apresentados e utilizados os protocolos que mais se adaptam ao tema escolhido, tentando simplificar a rede para que esta possa ser incluída em qualquer ambiente doméstico, recorrendo a *hardware* de custo reduzido. Os protocolos apresentados são o 6LoWPAN, utilizando o protocolo IEEE 802.15.4 como interface de rede juntamente com endereçamento IPv6. É também utilizado o protocolo CoAP na troca de mensagens entre os dispositivos.

Palavras-chave: Internet of Things, Smart-Houses, Wireless Sensor Network, 6LoWPAN, IEEE 802.15.4, IPv6, CoAP, Contiki, Border Router

[1] Parte desta dissertação foi realizada em parceria com Alan Lima, encarregue dos serviços aplicacionais (Aplicação de Internet of Things em casa inteligentes – Serviço Aplicacional)

Abstract

The main focus on the study of Internet of Things (IoT) has been the integration of digital devices with the physical world and vice-versa. Smart devices have been gaining a strong presence in our daily life, and increasingly tend to integrate the system of a house, automating common processes such as the control of ambient temperature or the percentage of luminosity of a room. The IoT vision comprises an interconnected world, gathering information automatically and enabling communication between devices. However, the inherent technologies for creating networks that host these new devices lack of well-defined standards, hindering their integration into existing solutions.

In this project are studied and used technologies that stand out in the study of Internet of Things, aiming to find a standard in the use of network protocols in home automation. [2]

The final goal of this project is the creation of a network of devices with sensing capabilities that have the ability to communicate with the outside world, allowing network access by any user. With this, it is expected to move closer to the standardization of protocols related to the IoT and enable interoperability between various solutions.

In this work are presented and used protocols that suit the target application (home aumation in smart houses), trying to simplify the network so that it can be included in any home environment, using low-cost hardware. The protocols presented are the 6LoWPAN, using the IEEE 802.15.4 as a network interface with IPv6 addressing capabilities. It is also used the CoAP protocol to exchange messages between the devices at the application level.

Keywords: Internet of Things, Smart Houses, Wireless Sensor Network, 6LoWPAN, IEEE 802.15.4, IPv6, CoAP, Contiki, Border Router

[2] Part of this dissertation was carried out in partnership with Alan Lima, responsible for application services (Aplicação de Internet of Things em casa inteligentes – Serviço Aplicacional)

Agradecimentos

Em primeiro lugar agradeço à minha família pelo esforço que fizeram para que eu concretizasse esta etapa, pelo apoio e incentivo recebido durante o meu percurso académico.

Agradeço também ao meu orientador no decorrer do projeto, o Dr. Nuno Alexandre Magalhães Pereira, e à equipa do CISTER por todo o apoio concedido durante a realização deste projeto.

Ao meu colega de projeto, Alan Tomás Lima, por me ter sempre incentivado durante o percurso. A todos os meus amigos que apareceram nos momentos mais difíceis.

A todas essas pessoas, o meu profundo e sincero,

Obrigado.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

6LoWPAN *IPv6 Over Low power Wireless Personal Area Networks*

ACK *Acknowledgement*

CON *Confirmable*

CSMA *Carrier Sense Multiple Access*

CoAP *Constrained Application Protocol*

DARPA *Defense Advanced Research Projects Agency*

DTLS *Datagram Transport Layer Security*

DYMO *Dynamic MANET On-demand*

FCS *Full Function Device*

FFD *Vector Space Model*

HTTP *Hypertext Transfer Protocol*

IEEE *Institute of Electrical and Electronics Engineers*

IETF *Internet Engineering Task Force*

IP *Internet Protocol*

IPv4 *Internet Protocol version 4*

IPv6 *Internet Protocol version 6*

IoT *Internet of Things*

JSON *JavaScript Object Notation*

LAN *Local Area Network*

LED *Light-Emitting Diode*

M2M *Machine to Machine*

MAC *Media Access Control*

MCU *Micro Controller Unit*

MIME *Multi-Purpose Internet Mail Extension*

MIT *Massachusetts Institute of Technology*

MPDU *MAC Protocol Data Unit*

MSDU *MAC Service Data Unit*

MTU *Maximum Transmission Unit*

NON *Non Confirmable*

OS *Operative System*

OSI *Open Systems Interconnection*

PAN *Personal Area Network*

PDU *Protocol Data Unit*

PSDU *PHY Service Data Unit*

RAM *Random Access Memory*

RDC *Radio Duty Cycling*

REST *Representational State Transfer*

RFD *Reduced Function Device*

RFID *Radio-Frequency IDentification*

ROM *Read Only Memory*

RPL *IPv6 Routing Protocol for Low power and Lossy Networks*

RREQ	*Route Request*
RST	*Reset*
SICS	*Swedish Institute of Computer Science*
SLIP	*Serial Line Internet Protocol*
SSL/TLS	*Secure Sockets Layer/ Transport Layer Security*
TCP	*Transmission Control Protocol*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
UDP	*User Datagram Protocol*
URI	*Uniform Resource Identifier*
USB	*Universal Serial Bus*
WLAN	*Wireless Local Area Network*
WPAN	*Wireless Personal Area Network*
WSN	*Wireless Sensor Network*
XML	*Extensible Markup Language*
rt	*Resource Type*
uIP	*micro IP*

1 Introdução

1.1 Motivação

A *Internet of Things* (IoT) consiste na visão de um mundo onde biliões de objetos conseguem comunicar, monitorizar e partilhar informação entre si, ligados através do Protocolo de Internet (IP). Estes objetos recolhem regularmente informações que são analisadas e utilizadas para iniciar ações de forma a criar um sistema mais inteligente e autónomo, capaz de tomar decisões sem a necessidade de interação de um utilizador.

O termo *Internet of Things* foi pela primeira vez utilizado por um cofundador do laboratório *Auto-ID* do *Massachusetts Institute of Technology* (MIT), Kevin Ashton, em 1999, numa apresentação que visava a criação de um sistema global de registo de bens recorrendo ao uso de *Radio-Frequency IDentification* (RFID) e de *Wireless Sensor Networks* (WSN's). A partir daí o termo foi cada vez mais utilizado e, em 2009, Kevin Ashton cria uma definição na qual diz que a maioria dos computadores estão dependentes da interação humana para obter informações e que perto de *50 petabytes* presentes na web foram introduzidos manualmente [Kevin Ashton, 2009]. Isto constitui um problema, pois as pessoas têm um tempo limitado para inserir estes dados e nesse processo podem existir erros de introdução, o que leva a erros nas análises e tratamento desses dados. Se os computadores conseguissem obter informação sem a necessidade de interação humana, seria possível obter dados mais detalhados e fiáveis em tempo real, reduzindo os erros e desperdício de tempo. Assim, é também possível tomar decisões em tempo real visto não existirem tempos de espera para obter as informações, podendo desta forma corrigir erros e substituir anomalias no momento em que estas são detetadas.

A IoT está a revolucionar a forma como pensamos a internet. Em 2003, o número de dispositivos conectados à internet rondava os 500 milhões para 6.3 biliões de pessoas. Em 2008, esse número aumentou de tal forma que ultrapassou a população humana. E, em 2012, ultrapassava os 8.7 biliões para 7 biliões de pessoas. Segundo um estudo da CISCO [Dave Evans, 2011], prevê-se que em 2020 estejam conectados 50 biliões de dispositivos à internet. Este aumento deve-se ao facto de ter sido ultrapassada a barreira da utilização da internet em computadores convencionais e telemóveis, passando a ser aplicada nos mais variados objetos. Tudo está a ficar ligado à internet: carros, máquinas de cafés, animais, alarmes, entre muitos outros. Ao utilizar a IoT, é atribuído um identificador único a todos os objetos que consigam suportar uma ligação à internet. Torna-se assim possível a sua descoberta a partir de qualquer lado e esses objetos ganham a capacidade de comunicar entre si, transmitindo informações sobre os seus estados sem necessidade de interação de um utilizador.

Resumidamente, a IoT é um conjunto de "coisas" que estão ligadas à internet de forma a comunicarem umas com as outras, trocando informação sobre algo como, por exemplo, um sensor de temperatura que comunica com um termóstato que, por sua vez, ajusta o ar condicionado consoante a temperatura ambiente; um sensor de pressão de ar que deteta quando o pneu necessita de revisão, emitindo um aviso; ou um sensor que monitoriza o batimento cardíaco de um paciente, entre muitas outras utilizações.

Neste documento são focadas, principalmente, as "coisas" presentes numa típica casa, de forma a criar um sistema independente de interação humana, capaz de tomar as suas próprias decisões. Aplicando os conceitos gerais presentes na IoT, pretende-se atingir um ambiente numa casa de forma a torná-la inteligente, podendo fornecer várias informações sobre o seu estado (temperatura, humidade, luzes, portas, entre outras).

1.2 Objetivos

Esta dissertação tem como objetivo principal o estudo das tecnologias envolvidas no conceito *Intenet of Things* que permitem a comunicação entre dispositivos de reduzida capacidade e um utilizador final. Espera-se, assim, que a solução obtida tire partido dos mais recentes protocolos de rede e que consiga ser aplicada em qualquer situação real de forma a melhor otimizar a solução. As principais contribuições deste trabalho são:

Estudo, avaliação e demonstração de tecnologias inerentes à *Internet of Things*.

Estudo das várias tecnologias pertencentes ao tema *Internet of Things* em contexto real, avaliando o seu desempenho face às tecnologias de rede existentes. O principal objetivo é a utilização dos protocolos que mais se destacam no presente de forma a estudar a sua interoperabilidade, tentando definir um possível *standard* de camadas protocolares.

Identificação dos diversos componentes tecnológicos associados ao tema.

Descrição detalhada dos vários componentes pertencentes ao projeto de forma a estruturar uma rede doméstica de sensores, tirando partido de *hardware* de capacidades limitadas tanto a nível energético como de processamento.

Estudo e implementação de uma rede 6LoWPAN com recursos limitados.

Identificação das tecnologias mais adequadas, a ser utilizadas numa rede 6LowPAN (*IPv6 over Low power Wireless Personal Area Network*) para obter o melhor rendimento de componentes com recursos limitados e sua implementação. Estudo dos componentes de uma rede 6LoWPAN, incluindo sensores, um *router* que permita a gestão da rede onde os sensores estão conectados e uma *gateway* que permita efetuar pedidos externos à rede.

1.3 Métodos e tecnologias utilizadas

Parte deste projeto foi realizado em conjunto com Alan Lima, encarregue de criar uma aplicação capaz de comunicar com os dispositivos a partir de qualquer dispositivo (Ex.: PC, telemóvel, *tablet*). O objetivo final desta parceria tem como base a criação de um sistema que contém uma rede de dispositivos de baixa capacidade de processamento e energia, e uma aplicação que consiga gerir os nós da rede, desenhando-os logicamente num mapa, possibilitando uma melhor interação.

A realização deste projeto começou pelo estudo das tecnologias inerentes ao tema *Internet of Things*, identificando os seus componentes tecnológicos de forma a criar uma melhor decisão na escolha das tecnologias a utilizar. Foram realizadas diversas experiências, as quais levaram à implementação de um demonstrador dos diversos componentes identificados.

Foi implementada uma rede 6LowPAN, recorrendo a várias tecnologias inerentes ao tema tais como o protocolo de comunicação CoAP (*Constrained Application Protocol*), o protocolo de rede IEEE 802.15.4 e a plataforma de desenvolvimento Contiki. De forma a obter uma *Wireless Sensor Network,* foram utilizados as plataformas de baixos recursos computacionais, com sensores e capacidades de comunicação sem fios (como o Advanticsys XM1000, Crossbow TelosB) juntamente com a BeagleBone Black como *Gateway*. Pode ser observado na Figura 1 um desenho simplista da rede e da aplicação das tecnologias.

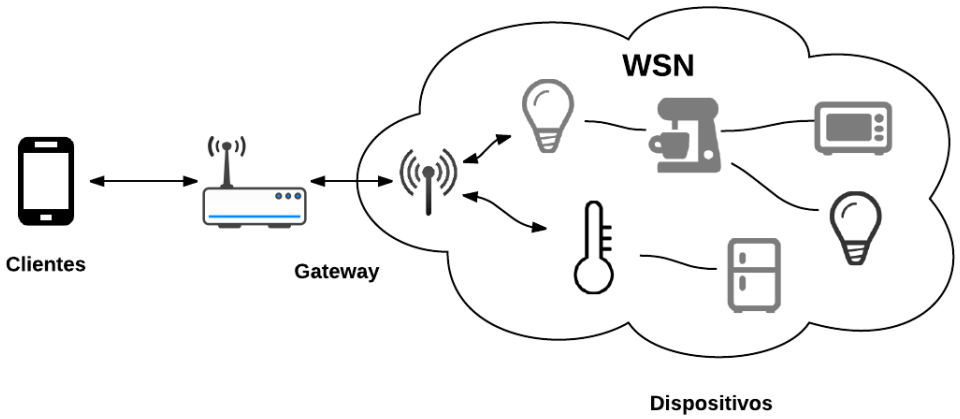


Figura 1 – Esquema da rede

1.4 Estrutura do documento

A dissertação está dividida em cinco capítulos, organizados da seguinte forma:

- No primeiro capítulo é feita uma breve descrição da solução a implementar e tecnologias associadas assim como a motivação que levou à escolha do tema e objetivos a cumprir;
- No segundo capítulo, Levantamento do Estado da Arte, pretende-se mostrar os conceitos tecnológicos presentes atualmente no tema, possibilitando a comparação entre os mesmos;
- No terceiro capítulo, Tecnologias Utilizadas, são descritas com mais pormenor as tecnologias inerentes ao projeto;

- No quarto capítulo, Demonstração, é mostrada a solução implementada com uma descrição da arquitetura e tecnologias utilizadas nas plataformas de forma a atingir o resultado final;
- No quinto capítulo, Testes, estão introduzidos os testes realizados ao sistema implementado final mostrando as suas vantagens e desvantagens;
- As conclusões deste trabalho são mostradas no sexto e último capítulo, Conclusões. Neste capítulo são mostradas as conclusões retiradas na realização do projeto, as suas principais limitações e o trabalho futuro a realizar.

2 Levantamento do Estado de Arte

O objetivo deste capítulo é fazer o levantamento de tecnologias relevantes que levaram à escolha do sistema final. Este levantamento não é exaustivo, tendo sido rapidamente identificadas as tecnologias que mais se apropriam ao tema proposto, sendo explicadas de forma mais profunda no capítulo seguinte. O estudo das tecnologias deste capítulo baseou-se maioritariamente no preenchimento das camadas de rede, possibilitando a sua definição.

2.1 Modelo TCP/IP

O modelo TCP/IP foi originalmente criado pela DARPA (*Defense Advanced Research Projects Agency*), uma agência do Departamento de Defesa dos Estados Unidos da América. Este modelo começou a ser desenvolvido na década de 60 de forma a interligar vários computadores numa só rede, surgindo o projeto ARPANet (*Advanced Research Projects Agency Network*). Este modelo foi adotado pela internet e neste momento é o modelo mais utilizado na criação de redes sendo mantido atualizado pela IETF (*Internet Engineering Task Force*).

Este modelo fornece especificações sobre como os dados devem ser colocados em pacotes, endereçados, transmitidos, roteados e recebidos no destino. É composto por quatro camadas que definem a Interface de Rede que contém os protocolos de comunicação, a camada de Endereçamento que é responsável pelas ligações de nós em redes independentes, a camada

de Transporte que trata as comunicações nó para nó e por fim, a camada de Aplicação que fornece os serviços para comunicação de dados entre processos.

O modelo TCP/IP é facilmente comparado com o modelo OSI (*Open Systems Interconnection model*). Como pode ser visto na Tabela 1, a camada mais baixa do modelo TCP/IP corresponde às duas camadas mais baixas do modelo OSI, camada Física e Enlace de Dados. A segunda camada, camada de Endereçamento, corresponde à terceira camada do modelo OSI, camada de Rede. A camada de transporte corresponde à quarta camada, contendo os mesmos conceitos tecnológicos. A camada de aplicação é associada às três camadas superiores do modelo OSI: Sessão, Apresentação e Aplicação [Microsoft, 2005].

Tabela 1 – Comparação entre o modelo TCP/IP e o modelo OSI

Modelo TCP/IP	**Modelo OSI**
Aplicação	Aplicação
	Apresentação
	Sessão
Transporte	Transporte
Endereçamento	Rede
Interfaces de Rede	Enlace de Dados
	Física

2.2 Interfaces de Rede

A interface de rede é utilizada para a comunicação entre os dispositivos. Esta encarrega-se das ligações entre dispositivos e da troca de mensagens entre estes. De seguida, são apresentados três padrões distintos para comunicações em redes sem fios.

2.2.1 IEEE 802.11

A rede sem fios IEEE 802.11, também conhecida como Wi-Fi ou *wireless*, consiste num conjunto de especificações da camada de Interface de Rede para implementação de uma *Wireless Local Area Network* (WLAN). O IEEE 802.11 consiste numa série de comunicações *half-duplex* através do ar, ou seja, existe comunicação entre dois pontos, mas quando um dos pontos está a transmitir informação o outro ponto está a receber e apenas pode transmitir

quando o outro ponto terminar a sua transmissão. O Wi-Fi é umas das tecnologias-chave para a ligação entre dispositivos e uma rede próxima, dado que existem inúmeros dispositivos a serem conectados entre si e é impossível estarem ligados fisicamente. Apesar de possuir uma boa taxa de transmissão (802.11 até 54 Mbps, 802.11b até 11Mbps, 802.11n até 300Mbps) e de possuírem um alcance entre os 100 e os 250 metros, têm um consumo de energia muito grande para os aparelhos mais simples (sensores, atuadores, entre outos), o que constitui um problema uma vez que estes dispositivos são equipados com baterias com pouca capacidade [IEEE Communication Magazine, 1997].

Podemos observar na Figura 2 o tamanho de um pacote desta rede, contendo um *Header* de 30 bytes que permite identificar o tipo de versão que está a ser utilizado, o tempo esperado para a próxima transmissão e endereços identificadores do pacote. Contém um *Payload* até 2312 bytes onde são guardados os dados a transmitir e, por fim, uma *Frame Check Sequence* (FCS) com 4 bytes de forma a verificar se ocorreram erros na transmissão do pacote.

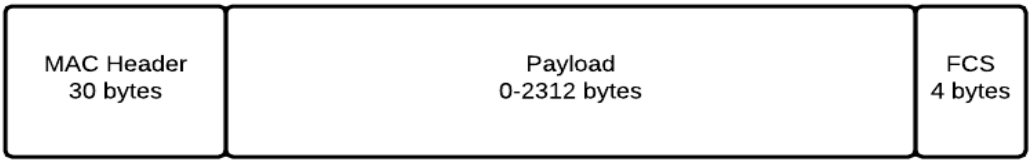


Figura 2 – Formato de um pacote da rede IEEE 802.11

2.2.2 IEEE 802.15.1

O IEEE 802.15.1, também conhecido como *Bluetooth*, é o conjunto de especificações para comunicações *wireless* de curta distância. Estas especificações foram desenvolvidas, inicialmente, para efetuar a transferência de dados entre computadores pessoais para dispositivos periféricos, tais como telemóveis. O *Bluetooth* usa a topologia *star*, a qual permite que um conjunto de nós possa comunicar com um nó central, como se pode observar na Figura 3.

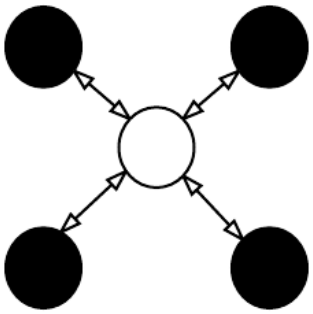


Figura 3 – Topologia Star

Esta tecnologia possui certas limitações, como um curto raio de transmissão ou uma longa demora na ligação entre nós e a rede quando estes estão suspensos, o que leva a um aumento do consumo energético do sistema e apenas permite que sete nós possam estar conectados à mesma.

Na Figura 4, é apresentado um modelo dos pacotes transmitidos numa ligação *Bluetooth* constituído por 72 *bits* para o código de acesso, utilizado para identificar a *piconet (rede wireless Bluetooth)*, o dispositivo e os procedimentos de inquisição, 54 *bits* para o cabeçalho e 2745 *bits* para os dados a transmitir.

O código de acesso permite identificar os pacotes trocados numa rede, ou seja, todos os pacotes que são enviados na mesma rede possuem o mesmo código de acesso. O cabeçalho contém informação de controlo que permite a identificação do pacote, número da sequência e controlo de erros [Jose Pique, Ignacio Almazan, Daniel Garcia, 2010].

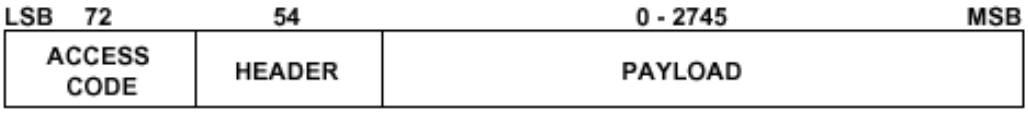


Figura 4 – Formato de um pacote da rede IEEE 802.15.1

2.2.3 IEEE 802.15.4

O padrão IEEE 802.15.4 insere-se na camada mais baixa do modelo TCP/IP. Este padrão foi desenhado para solucionar o problema de comunicação com dispositivos que possuem baixa capacidade de transmissão e de baixa capacidade energética, utilizando taxas de transmissão

baixas (250 kb/s, 40 kb/s e 20 kb/s). O padrão referido tem um alcance de 10 metros [Alan Ott, 2012].

Na camada física o formato da unidade de dados pode conter até 1016 *bits* de *payload* (corpo de dados que contém a informação a ser transmitida). O cabeçalho contém informações que permitem a sincronização do dispositivo com o pacote a receber. O pacote contém 32 *bits* de preâmbulo, o qual permite a sincronização entre dispositivo-pacote e 8 *bits* que delimitam o pacote, separando o preâmbulo da informação a transmitir. O cabeçalho físico contém o tamanho do *payload*, contendo 8 *bits,* como pode ser visto na Figura 5 [José Pereira, 2013].

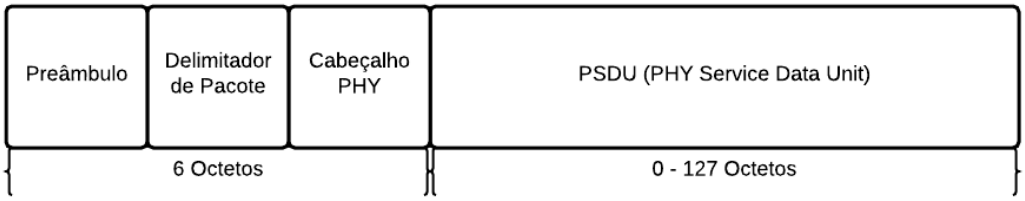


Figura 5 - Formato de um pacote da rede IEEE 802.15.4

A camada MAC suporta as topologias *Peer-to-Peer*, onde cada nó tem a capacidade de comunicar diretamente com outro nó sem precisar de enviar a informação a um sistema centralizado, e a topologia *Star* onde os nós enviam a informação para um sistema central e este a distribui de acordo. Esta camada contém um mecanismo de validação e rejeição de mensagens e garante também a entrega de pacotes. Nesta camada, as operações de baixo consumo energético são bastante facilitadas [José A. Gutierrez, 2005].

Ao utilizar este padrão, os dispositivos podem ser suspensos por longos períodos de tempo de forma a garantir maior poupança energética. Têm também uma transmissão de dados indireta permitindo verificar alternativas no encaminhamento das mensagens e permitem o controlo do "*receiver state*" pelas camadas mais altas de forma a verificar a correta entrega da mensagem

.

2.3 Rede

Na atualidade existem dois protocolos de rede de grande relevância, o IPv4 e o IPv6, que permitem identificar um dispositivo numa rede. Nesta secção apresentamos resumidamente cada um deles.

2.3.1 IPv4

O IPv4 pertence à segunda camada do modelo TCP/IP, camada de Rede, é a quarta versão do protocolo de internet (IP) e a mais utilizada no mundo. Um endereço IP consiste num identificador numérico que está associado a um dispositivo com o objetivo de o identificar e localizar. Um endereço IPv4 consiste num número de 32 bits, constituído por 4 octetos e, consequentemente, limitado a 4 294 967 296 (2^{32}) endereços. Com o grande crescimento da internet e dos dispositivos a ela ligados, rapidamente esta gama de endereços IP deixará de ser suficiente para a identificação de todos os dispositivos no mundo. Um endereço IPv4 é visto, então, da seguinte forma em formato decimal: 255.255.255.255.

2.3.2 IPv6

O IPv6 é a versão mais recente do protocolo IP que fornece uma forma de identificar dispositivos em redes. Este protocolo foi criado com vista a superar a escassez de endereços presentes no IPv4. Este protocolo usa 128bits, permitindo 2^{128} endereços, contrariamente ao IPv4, que apenas permitia 2^{32}. Com este aumento do número de endereços seria possível atribuir um endereço de IP a todos os átomos presentes no planeta e, ainda, a mais 100 planetas semelhantes. Os endereços deste protocolo são representados por oito grupos de quatro números hexadecimais separados por dois pontos. Estes endereços são normalmente escritos como oito grupos de 4 dígitos hexadecimais: 2001:0db8:85a3:0000:0000:0000:0000:7344. Se num endereço IPv6 existirem grupos de números constituídos apenas por 0000, estes podem ser omitidos: 2001:0db8:85a3::7344.

2.3.3 6LoWPAN

6LoWPAN (*IPv6 over Low power Wireless Personal Area Networks*), permite utilizar o protocolo IPv6 nas redes IEEE 802.15.4. Os tópicos mais pertinentes abordados por esta

arquitetura baseiam-se na fragmentação/desfragmentação e compressão de cabeçalhos IPv6. No IPv6, a MTU (*Maximum Transmission Unit*) é de 1280 octetos, enquanto na rede IEEE 802.15.4 o máximo permitido é de 127 octetos, daí a importância de conseguir fragmentar e comprimir os cabeçalhos IPv6 [José A. Gutierrez, 2005]. Esta arquitetura serve como camada de adaptação entre o IPv6 e o IEEE 802.15.4.

Ao utilizar 6LoWPAN, conseguimos que todos os pequenos dispositivos possam ter um endereço de IP, possibilitando que estes sejam encontrados na rede. Esta arquitetura vem facilitar o modo de comunicação com pequenos dispositivos. Estes dispositivos necessitam de ter um baixo consumo energético e, por isso, é indicado que utilizem a rede IEEE 802.15.4. Visto que cada vez existem mais dispositivos deste género, é necessário utilizar o protocolo IPv6 para lhes ser atribuído um endereço distinto. Resumidamente, o 6LoWPAN serve como um tradutor de pacotes do tipo IPv6 para que estes sejam utilizados na rede IEEE 802.15.4, comprimindo o seu tamanho sem ação do utilizador e sem perda de dados.

2.4 Aplicação

2.4.1 REST

Representational State Transfer (REST) é um estilo de arquitetura para sistemas distribuídos, apresentado por Roy Fielding, em 2000, na sua tese de doutoramento. Uma aplicação RESTful é uma aplicação que expõe o seu estado e as suas funcionalidades como forma de recurso, para que os seus clientes possam manipulá-los e que seguem um conjunto de princípios. Um exemplo muito comum deste estilo de arquitetura é o HTTP.

Um recurso constitui todos os dados e operações representadas num sistema, ou seja, desde pessoas, páginas, coleções de dados, operações (como encomendas), entre outras. Assim, este estilo de arquitetura, segue duas ideias principais: tudo é um recurso e todos os recursos possuem uma interface bem definida.

Este estilo de arquitetura rege-se pelos seguintes princípios:

- Todos os recursos são acedidos de uma forma única, ou seja, cada recurso necessita de um identificador único, que permita fazer referência a este sem qualquer tipo de erro. Na maioria das vezes são utilizados Uniform Resource Identifiers (URI's).

- Os recursos são manipulados através de uma interface uniforme, tal como definida na Tabela 2, que é constituída por um conjunto de operações conhecidas e que executam ações bem definidas, tais como: Create, Read, Update e Delete.

Tabela 2 – Operações REST

Método	**Descrição**
GET	Obtém a representação de um determinado recurso;
PUT	Cria ou atualiza um recurso fornecendo a representação do recurso;
DELETE	Apaga um determinado recurso;
POST	Envia dados para o recurso especificado para ser processada;
HEAD	Semelhante ao GET, contudo, apenas recebe os *headers* dos recursos;
OPTIONS	Retorna os métodos suportados para o recurso especificado.

A representação de um recurso representa o valor dos dados desse recurso no momento em que a requisição foi recebida. Um recurso pode possuir diversas representações e, deste modo, o cliente pode escolher, de entre as representações disponíveis, a representação que desejar. Alguns exemplos de representações são: HTML, XML, JSON, entre outros.

As comunicações efetuadas entre o cliente e o servidor devem ser *stateless*, ou seja, sem estado. Este princípio diz que, no servidor, não pode haver manutenção de informações sobre o utilizador e, para tal, sempre que é efetuado um pedido ao servidor, o cliente deve enviar toda a informação para que esse pedido possa ser compreendido. Uma das vantagens deste princípio é o aumento da escalabilidade: nos servidores que possuem um elevado número de utilizadores, guardar as informações de sessão destes iria levar a uma diminuição de desempenho do sistema. Além da escalabilidade, este princípio também diminui a dependência dos clientes em relação ao servidor, pois cada pedido não tem

dependências de informações contidas no servidor [W. Colitti, K. Steenhaut, N. De Caro, Bogdan Buta and Virgil Dobrota, 2011].

2.4.2 CoAP

O IETF tem vindo a trabalhar para a criação de padrões para a ligação de dispositivos com capacidades limitadas à Internet. O *Constrained Application Protocol* (CoAP) corresponde à camada de aplicação na criação desses standards.

O CoAP descreve um estilo arquitetura REST para redes de capacidades limitadas, tendo um mapeamento transparente para HTTP. Este implementa um conjunto de funcionalidades do HTTP completamente reestruturadas para o suporte de dispositivos com recursos limitados. CoAP funciona sobre o protocolo UDP, não sendo então disponível a tecnologia SSL/TLS mas utiliza o DTLS (*Datagram Transport Layer Security*). Os sensores CoAP são desenhados para se comportarem como servidores e clientes, de modo a possibilitarem a interação máquina-máquina. As mensagens são trocadas através de pedidos efetuados aos valores de um recurso acessíveis através de URIs [Zach Shelby, 2013].

O CoAP disponibiliza também um mecanismo de *publish/subscribe* no qual o cliente indica que pretende receber as atualizações de um recurso sempre que o seu estado for alterado. Um pedido GET com a opção observe ativa leva a que o servidor registe o cliente na lista de observadores do recurso. Cada notificação enviada para o cliente é uma resposta adicional ao pedido efetuado ao servidor.

2.5 Plataformas

2.5.1 Contiki OS

O Contiki é um sistema operativo *Open Source* para dispositivos com capacidades limitadas. Este sistema operativo é escrito na linguagem C e é baseado em eventos. Fornece multitarefas preemptivas que podem ser utilizadas ao nível individual de processos. Baseando-se em multitarefas, preempções, TCP/IP e IPv6, o Contiki distingue-se bastante perante os outros sistemas para dispositivos de baixo consumo energético, disponibilizando várias tecnologias com mais destaque na área de *Internet of Things*.

A arquitetura do Contiki é modular, permitindo adicionar e remover funcionalidades dependendo do fim pretendido. Utiliza um escalonador de eventos bastante leve, possibilitando o despacho de eventos do *kernel* para os processos. Suporta eventos síncronos e assíncronos.

É disponibilizada uma implementação do protocolo TCP/IP para microcontroladores de 8 bits (uIP – micro IP). No entanto, *multicast* não está disponível nesta plataforma, sendo os pacotes colados num *buffer* global e a pilha TCP/IP notificada desse evento, tratando-o consoante o seu propósito.

2.5.2 TinyOS

O TinyOS é um sistema operativo concebido para sistemas de baixo consumo energético sem fios. Este sistema é *Open Source* e consegue suportar programas que exijam pouca memória. A biblioteca presente no TinyOS inclui protocolos de rede, serviços distribuídos, controladores de sensores e ferramentas para aquisição de dados [TinyOS, 2013].

A arquitetura do TinyOS é monolítica, utilizando diversos componentes, agregando-os com um escalonador de forma a compor uma imagem estática para executar no dispositivo. Um dispositivo é uma entidade que deve ter três abstrações: comandos, eventos e tarefas, sendo os comandos e eventos utilizados para comunicação entre dispositivos e as tarefas para concorrência interna de serviços.

Foram disponibilizados dois protocolos de *multi-hop* no TinyOS: *dissemination* e TYMO. O protocolo dissemination entrega os dados a todos os nós na rede e fornece duas interfaces distintas: *DisseminationValue* e *DisseminationUpdate*. O *DisseminationUpdate* é utilizado por um consumidor sempre que este quer alterar um valor, enquanto o *DisseminationValue* é fornecido pelo consumidor. TYMO é a implementação do protocolo DYMO (*Dynamic MANET On-demand*), um protocolo de *routing* para redes *ad-hoc* [Edosoft, 2012].

3 Tecnologias utilizadas

Parte deste projeto foi realizado em conjunto com Alan Lima, encarregue de criar uma aplicação capaz de comunicar com os dispositivos a partir de qualquer dispositivo (Ex.: PC, telemóvel, *tablet*). O objetivo final desta parceria tem como base a criação de um sistema que contém uma rede de dispositivos de baixa capacidade de processamento e energia, e uma aplicação que consiga gerir os nós da rede, desenhando-os logicamente num mapa, possibilitando uma melhor interação.

Na escolha das tecnologias a utilizar, o fator de maior relevância mostrou-se ser o peso que as tecnologias ocupam no caminho da sua padronização e na definição de uma estrutura de rede para a *Internet of Things*. Foram também levantadas outras questões, tais como a sua utilização em dispositivos que exigem baixo consumo energético e de baixa capacidade de processamento. De uma forma resumida as principais preocupações na escolha das tecnologias foram:

- Capacidade de ROM dos nós;
- Capacidade de RAM dos nós;
- Baixo consumo energético;
- Estabilidade nas conexões;
- Disponibilidade e maturidade das implementações disponíveis atualmente.

Tendo como base o esquema apresentado na Figura 6 e os fatores de influência apresentados anteriormente, foram escolhidas as tecnologias que mais se adaptavam ao modelo. O tema

abordado por este projeto é um tema ainda pouco estudado, tendo um grande peso na escolha das tecnologias a sua padronização, abordando as que mais se aproximam deste conceito. Nas secções seguintes são apresentadas as tecnologias e protocolos a serem utilizados neste projeto de forma a completar as comunicações e interações do esquema referido na figura 6.

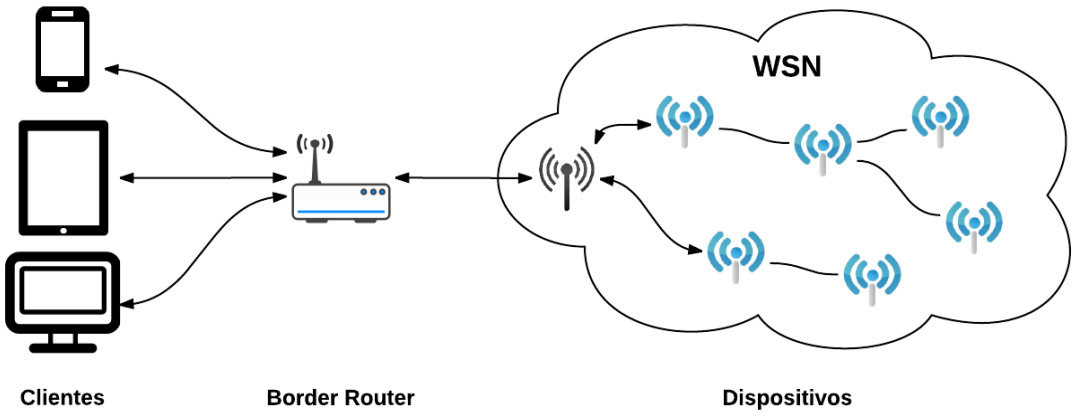


Figura 6 - Esquema de rede para atribuição de tecnologias

3.1 Interface de Rede – IEEE 802.15.4

A norma IEEE 802.15.4 foi desenvolvida pelo IEEE (*Institute of Electrical and Electronics Engineers*), uma organização sem fins lucrativos, com o objetivo de aumentar o desenvolvimento tecnológico em prol da sociedade.

3.1.1 Enquadramento

A norma IEEE 802.15.4 fornece as especificações para a camada física (PHY) e de controlo de acesso (MAC) a uma rede sem fios, tendo como base a norma 802, nas quais se inserem as normas: 802.3 (*Ethernet*), 802.11 (Redes locais sem fios WLAN) e 802.15 (Redes de área pessoal sem fios WPAN) [Luís Antunes, 2012].

A rede 802.15 foi criada tendo em vista as especificações de redes pessoais com baixo consumo energético e custo reduzido, definindo vários protocolos de rede, entre os quais:

- 802.15.1 – Baseada na tecnologia *Bluetooth* permitindo a ligação sem fios de vários dispositivos;
- 802.15.2 – Permite a coexistência das redes WPAN com redes WLAN;
- 802.15.4 – Garante um consumo energético reduzido com baixa complexidade, de alcance até 300 metros.

3.1.2 Camada física e de acesso

Esta norma especifica a camada física (PHY) e camada de acesso (MAC) para redes sem fios de baixo consumo energético. Estas camadas oferecem serviços às camadas superiores utilizadas pela rede, como pode ser visto na Figura 7.

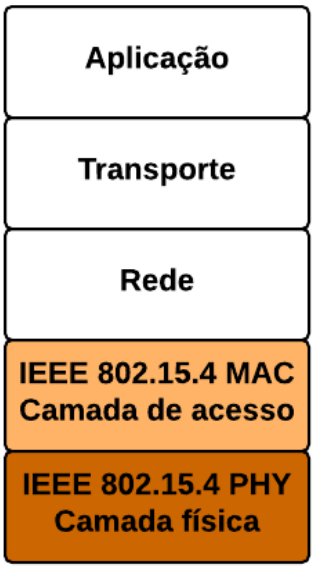


Figura 7 - Localização das camadas IEEE 802.15.4 no Modelo TCP/IP

Camada Física

A camada física é responsável pela transmissão das PDUs (*Protocol Data Units*). Esta camada suporta três frequências distintas (868 MHz, 915 MHz e 2.4GHz) em vinte e sete canais, sendo um canal dedicado para a primeira frequência, dez canais para a segunda e os restantes dezasseis para a frequência de 2.4GHz, como pode ser visto na Figura 8. Permite também a

configuração de diversos níveis de segurança, endereçamento automático de dispositivos e a possibilidade de confirmação de mensagens [José Pereira, 2013].

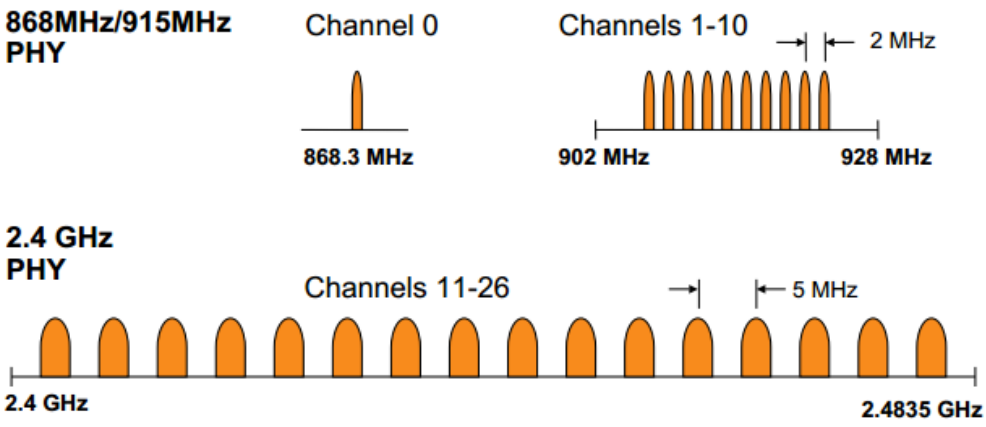


Figura 8 - Canais e Frequências do protocolo IEEE 802.15.4

A estrutura de um pacote da camada física é dividida em quatro partes distintas, como pode ser visto na Figura 9, sendo estas:

- Preâmbulo – Permite a sincronização, utilizando 32 bits;
- Delimitador de pacote – Fornece a indicação do começo dos dados usando 8 bits;
- Cabeçalho PHY – Contém a dimensão dos dados enviados, ocupa um tamanho no pacote de 8 bits;
- PSDU – Dados a transportar com o tamanho variável de 0 a 1016 bits.

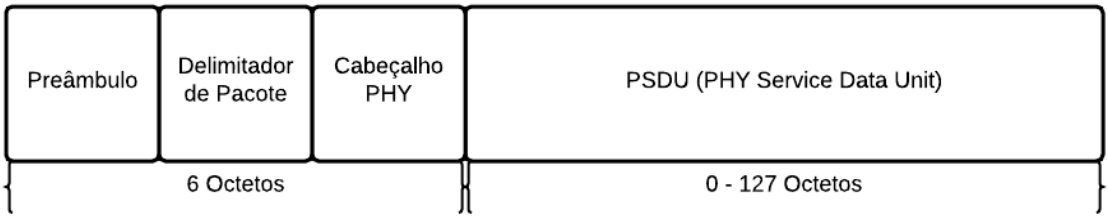


Figura 9 - Composição do pacote da camada PHY IEEE 802.15.4

Camada de Acesso

A camada de acesso (MAC) permite a transmissão dos pacotes utilizando o canal físico e é responsável por várias tarefas tais como: gerar *beacons* se o dispositivo for um coordenador,

sincronizar os *beacons*, suportar a associação e desassociação de nós em redes em áreas pessoais (PAN), entre outras [José A. Gutierrez, 2005].

Nesta norma existem três tipos de dispositivos [Ian Poole, 2014]:

- RFD (*Reduced Function Device*) – Estes dispositivos apenas podem ser utilizados na topologia Star, contendo recursos limitados. Nunca podem ser coordenadores de uma rede e são sempre terminais.
- FFD (*Full Function Device*) – Estes dispositivos podem ser utilizados em qualquer topologia, podendo comunicar com qualquer outro dispositivo. Podem tornar-se coordenadores da rede.
- PAN coordenador FFD – Este dispositivo é o mais completo de todos, contendo mais funcionalidades do que os outros dois e tendo a capacidade de coordenar uma rede.

Estes dispositivos podem ser inseridos em duas topologias de rede distintas: topologia *Star* e topologia *Peer to Peer*. Pode ainda ser feita uma mistura das duas topologias, criando assim uma topologia combinada.

- Topologia *Star* – Nesta topologia, todos os nós têm de comunicar com um nó central (PAN coordenador FFD). Um dispositivo FFD tem também a obrigação de comunicar expressamente com o nó central. Na Figura 10 pode ser visto o formato desta topologia.

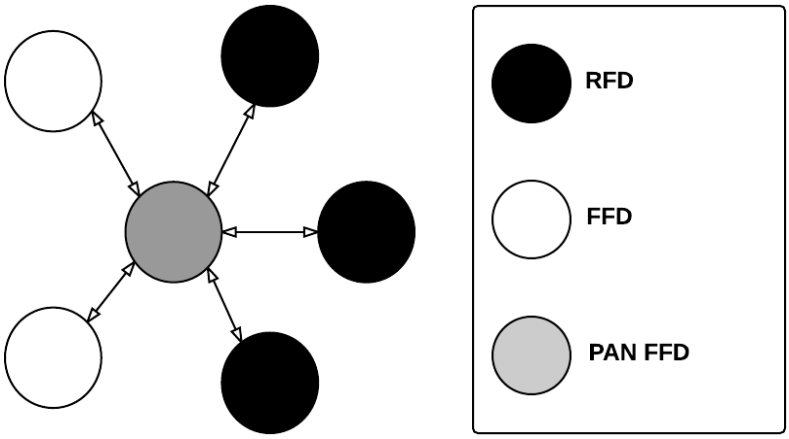


Figura 10 - Topologia Star no protocolo IEEE 802.15.4

- Topologia *Peer to Peer* – Nesta topologia, os nós do tipo FFD podem comunicar livremente entre si sem necessitarem de comunicar com um PAN coordenador FFD, sendo os FFD capazes de reencaminhar mensagens, enquanto os RFD apenas podem fazer comunicações diretas. Assim, utilizando esta topologia, consegue-se rapidamente aumentar a rede, redirecionando as comunicações pelos dispositivos FFD. Na Figura 11 pode ser visto um exemplo desta topologia.

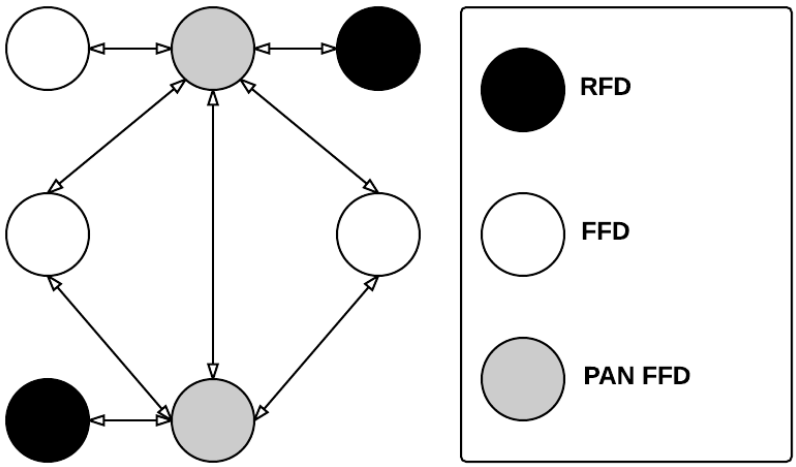


Figura 11 - Topologia Peer to Peer no protocolo IEEE 802.15.4

- Topologia Combinada – Esta topologia está reservada para redes mais complexas, combinando os tipos de topologias listadas anteriormente. Na Figura 12 pode ser visto um exemplo desta topologia.

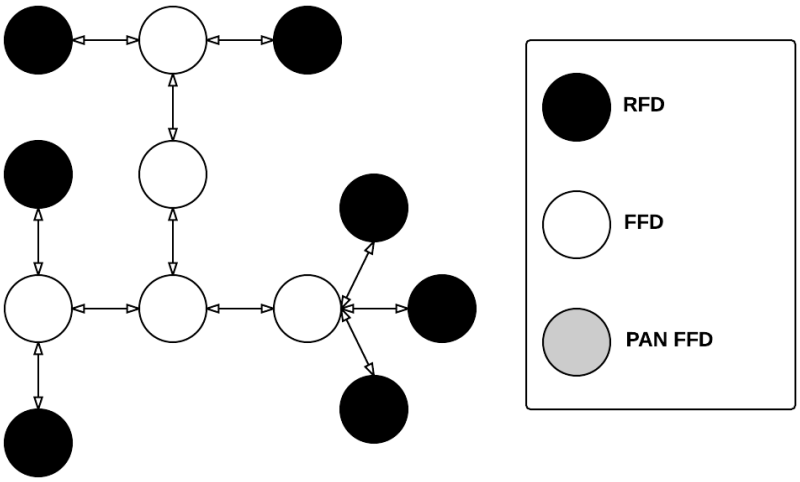


Figura 12 - Topologia Combinada no protocolo IEEE 802.15.4

Uma MAC *frame* é composta por um cabeçalho (MAC *Header*), por um *Payload* (MSDU) e por um rodapé (MAC *Footer*) [Marco Naeve, 2004]. Este é o formato geral de uma *frame* da camada de acesso, como pode ser visto na Figura 13.

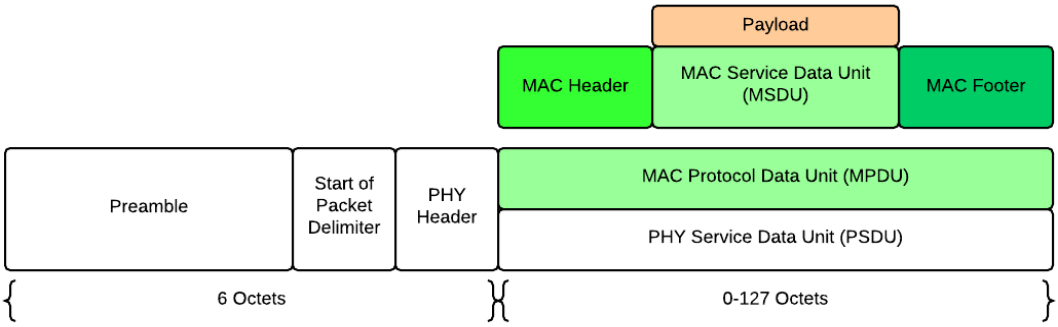


Figura 13 - Composição do pacote da camada MAC do protocolo IEEE 802.15.4

Como podemos ver na Figura 14, o MAC *Header* é composto por vários campos, focando-se principalmente no endereçamento dos pacotes e no controlo da *frame*. Podemos observar o tamanho que cada campo ocupa, podendo o MAC *Header* variar entre 3 a 23 bytes, o MSDU variável e o MAC *Footer* com 2 bytes.

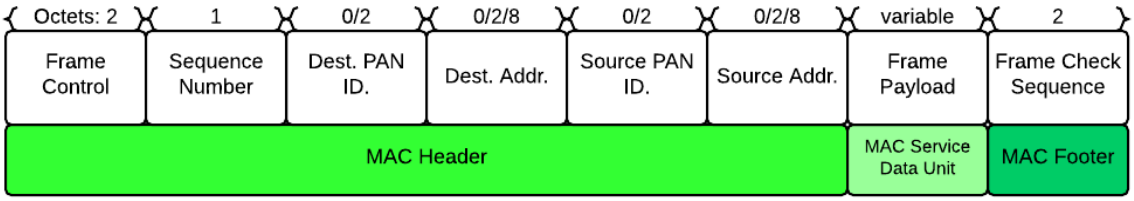


Figura 14 - Ilustração de um cabeçalho MAC do protocolo IEEE 802.15.4

3.2 Rede – IPv6

O IPv6 (*Internet Protocol version 6*) é o sucessor do protocolo de internet IPv4, desenvolvido pela IETF (*Internet Engineering Task Force*). As principais modificações do protocolo são [IETF, 1998]:

- Expansão do número total de endereços, passando o tamanho do endereço de 32 bits (IPv4) para 128 bits;
- Simplificação do formato do cabeçalho, tendo sido deixados de parte alguns dos campos presentes no IPv4 de forma a reduzir o custo de processamento e para limitar a largura de banda;
- Suporte melhorado para extensões e opções, modificando a forma como as opções no cabeçalho são codificadas, de forma a obter um melhor redireccionamento, menor limite no tamanho ocupado pelas opções e maior flexibilidade na introdução de novas opções;
- Capacidade de rotular os pacotes de forma a identificar o tráfego, tais como pedidos em tempo real;
- Capacidade de autenticação e de privacidade, possibilitando integridade de dados e confidencialidade.

Como dito anteriormente o cabeçalho IPv6 sofreu uma simplificação nos seus campos face ao seu antecessor, o IPv4 [Figura 15].

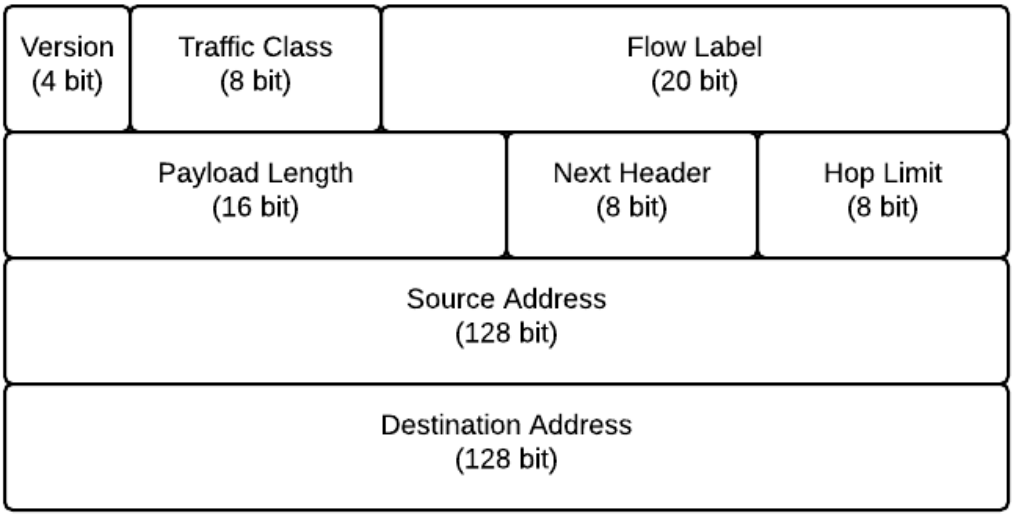


Figura 15 - Cabeçalho de um pacote IPv6

No total, os campos presentes neste cabeçalho ocupam 320 bits e são descritos da seguinte forma [Kaushi Das, 2008]:

- *Version* (4 bits) – Versão do protocolo de internet = 6;
- *Traffic Class* (8 bits) - Este campo pode assumir diferentes valores, permitindo ao *router* distinguir os pacotes com diferentes prioridades e também diferentes classes de tráfego;
- *Flow Label* (20 bits) – Pode ser utilizado pela origem do pacote para rotular um conjunto de pacotes que pertencem à mesma sequência. Uma sequência é identificada pela combinação do endereço de origem com um rótulo não igual a zero;
- *Payload Length* (16 bits) – Usado para indicar o tamanho total dos dados que sucedem o cabeçalho IPv6;
- *Next Header* (8 bits) – Identifica o tipo do cabeçalho que precede o cabeçalho IPv6, localizado no princípio do *Payload*. Geralmente utilizado para especificar a camada de transporte utilizada, como por exemplo TCP (6) ou UDP (17);
- *Hop Limit* (8 bits) – Este valor é decrementado sempre que o pacote passa por um nó ao ser reencaminhado. Se o valor chegar a zero, o pacote é descartado. O objetivo principal deste campo é descartar pacotes que fiquem presos em *loop* devido a erros de reencaminhamento;

- *Source Address* (128 bits) – Contém o endereço IPv6 em que o pacote foi originalmente criado;
- *Destination Address* (128 bits) – Contém o endereço de destino do pacote.

Um endereço IPv6 é composto por oito grupos de quartetos hexadecimais separados por dois pontos entre os mesmos. De seguida, podemos ver um exemplo de um endereço IPv6: 2001:cdba:0000:0000:0000:0000:3257:9652. Como se pode ver, este endereço contém alguns quartetos compostos apenas por zeros. Estes quartetos podem ser ocultados utilizando a notação "::", omitindo assim a sequência de zeros como visto de seguida: 2001:cdba::3257:9652.

Tal como no IPv4, no IPv6 também existem endereços reservados com notações especiais tais como [RIPE NCC, 2011]:

- ::/96 – Simboliza um endereço composto apenas por zeros, compatível com o protocolo IPv4;
- ::/128 – Um endereço IPv6 composto apenas por zeros, utilizado por um *host* a ser inicializado, previamente a conhecer o seu endereço;
- ::1/128 – Chamado de *Loopback*, é utilizado quando um *host* comunica consigo mesmo, o semelhante ao endereço 127.0.0.1 no protocolo IPv4;
- fe80::/10 – Utilizado numa rede de acesso comum, tal como uma Ethernet LAN;
- ff00::/8 – Utilizado para identificar grupos de *multicast*. Deve ser usado apenas como endereço de destino.

3.2.1 6LoWPAN

6LoWPAN (*IPv6 over Low power Wireless Personal Area Networks*), desenvolvido pelo IETF, é o conceito de que o protocolo de internet deve ser aplicado aos mais pequenos dispositivos com baixo consumo energético e reduzida capacidade de processamento. Foi desenvolvido de forma a conseguir juntar todos estes aparelhos na *Internet of Things*. O grupo que desenvolveu o 6LoWPAN definiu a compressão e fragmentação do cabeçalho IPv6 para serem enviados e recebidos utilizando o protocolo IEEE 802.15.4.

As caraterísticas principais do 6LoWPAN são:

- Pequeno tamanho dos pacotes, tendo em conta que o tamanho máximo de um pacote da camada física é de 127 bytes, restam apenas 81 octetos para os pacotes de dados;
- Suporte para endereços MAC de 16 bit ou IEEE 64 bit;
- Baixa largura de banda: 250kbps, 40kbps e 20kbps;
- Dispositivos de consumo e custo baixo;
- Maior número de dispositivos suportados devido ao uso do protocolo IPv6;
- Permite que os dispositivos fiquem inativos por certos períodos de tempo de forma a maximizar a duração da sua bateria.

Compressão do cabeçalho IPv6

O MTU (*Maximum Transmission Unit*) do protocolo IPv6 é de 1280 bytes enquanto na IEEE 802.15.4 temos um máximo de 128 bytes por *frame*, onde 25 bytes são reservados para o *overhead* das *frames*, restando apenas 102 bytes para o *payload*. Se adicionarmos os campos de segurança da camada MAC, podemos apenas ficar com 81 bytes disponíveis para o pacote IPv6, concluindo que um pacote IPv6 é demasiado grande para ser enviado a partir da rede IEEE 802.15.4. Com isto, o grupo 6LoWPAN sugeriu a adição de uma camada de adaptação entre a camada MAC e a camada de internet para conseguir comprimir e fragmentar o cabeçalho IPv6. Na Tabela 3, podemos ver o resultado da compressão do cabeçalho utilizando o codificador HC1 definido no 6LoWPAN [IETF, 2007].

Tabela 3 - Constituição do cabeçalho IPv6 comprimido pelo 6LoWPAN

Campo do cabeçalho	**Tamanho cabeçalho IPv6**	**Tamanho 6LoWPAN segundo HC1**	**Explicação**
Version	4 bits	---	Assumindo comunicação com IPv6
Traffic Class	8 bits	1 bit	0=não comprimido 1=comprimido
Flow Label	20 bits		
Payload Length	16 bits	---	Pode ser entregue pelo tamanho da MAC frame
Next Header	8 bits	2 bits	Comprimido quando os pacotes utilizam UDP, TCP ou ICMPv6

Campo do cabeçalho	Tamanho cabeçalho IPv6	Tamanho 6LoWPAN segundo HC1	Explicação
Hop Limit	8 bits	8 bits	Único campo não comprimido
Source Address	128 bits	2 bits	Se ambos os endereços são locais, os prefixos de 64 bits são comprimidos para um único bit com o valor de um. Outro bit é posto a um de forma a indicar que o identificador da interface 64 bit está omitido se o destino puder derivar do endereço do cabeçalho 802.15.4
Destination Address	128 bits	2 bits	
HC2 encoding	----	1 bit	Outro esquema de compressão que segue o cabeçalho HC1
Total	40 bytes	2 bytes	Totalmente comprimido, o cabeçalh IPv6 é reduzido a 2 bytes

Fragmentação do cabeçalho IPv6

Após a compressão do cabeçalho IPv6, verifica-se que ainda resta pouco espaço para introduzir dados, daí ter sido desenvolvido um processo de fragmentação pelo grupo 6LoWPAN. Este processo é fornecido pela nova camada de adaptação. Assim, os pacotes IPv6 que não se enquadram com o tamanho do *payload* da *frame* MAC são fragmentados em múltiplas *frames* IEEE 802.15.4 com as indicações necessárias para voltar a juntar todos os fragmentos num só pacote.

O cabeçalho fragmentado contém três campos:

- *Datagram Size* – Identifica a quantidade total de *payload* ainda não fragmentado e é incluído em todos os fragmentos para simplificar a alocação no *buffer* do recetor quando os fragmentos chegam fora de ordem;
- *Datagram Tag* – Identifica os fragmentos que pertencem a um *payload*;
- *Datagram Offset* – Identifica o *offset* do fragmento no *payload* total do fragmento e é representado em unidades de pedaços de 8 bytes.

Na Figura 16, podemos ver uma representação do espaço e organização que estes campos ocupam. A primeira sequência trata-se do primeiro fragmento do pacote a ser enviado, o qual não contém o campo *Datagram Offset*. A segunda sequência aplica-se a todos os fragmentos que se seguem, indicando o seu Offset. No primeiro caso, verificamos um cabeçalho fragmentado de 4 bytes face aos 5 bytes ocupados pelos restantes [Gee K., Chee K., Nor K., Borhanuddin M., 2010].

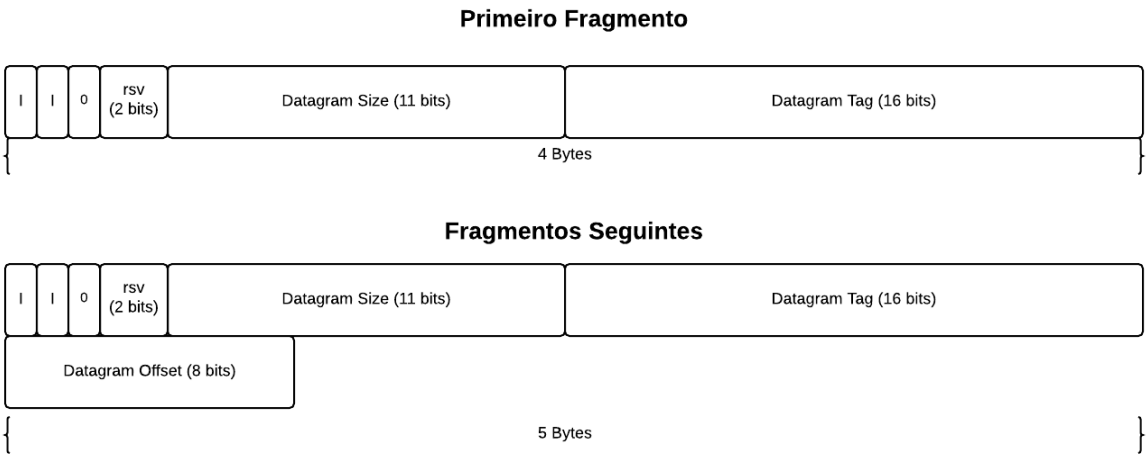


Figura 16 - Tamanho dos fragmentos de um pacote IPv6

3.3 Aplicação – CoAP

O CoAP (*Constrained Application Protocol*) é um protocolo especializado em transferências na web para ser utilizado em dispositivos com baixa capacidade energética e em redes limitadas. Desenhado para aplicações M2M (*Machine to Machine*), este protocolo fornece uma interação pedido/resposta entre *endpoints* aplicacionais.

O protocolo CoAP tem como funcionalidades principais as seguintes:

- Segue os requisitos M2M em ambientes limitados;
- Troca assíncrona de mensagens;
- Baixo *overhead* do cabeçalho;
- Suporte para URI e *Content-Type*;

- Capacidade para simples proxys e caches;
- Capacidade de construir proxys para providenciar acesso a recursos CoAP via HTTP.

3.3.1 Formato da mensagem

Os pacotes das mensagens são compostos por um *header* de 4 *bytes* seguidos por conjuntos de opções, como pode ser visto na Figura 17 [Tomislav D., Srdan K., Nenad G., 2011].

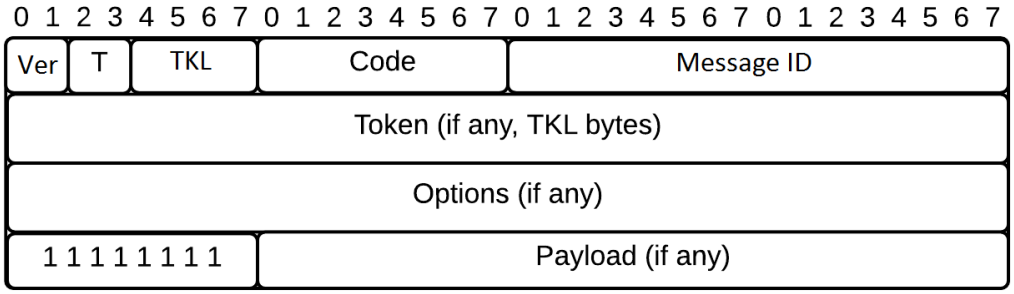


Figura 17 - Composição de um pacote CoAP

- Ver - Os primeiros dois *bits* indicam a versão do protocolo;
- T - O segundo conjunto de bits indica o tipo de mensagem: *Confirmable* (CON), *Non Confirmable* (NON), *Reset* (RST) e *Acknowledgement* (ACK).
- TKL - 4 bits utilizados para indicar o tamanho da mensagem;
- Code – 8 bits indicam os vários tipos de respostas (1-31) e de pedidos (64-191). O valor 0 é permitido para mensagens vazias;
- Message ID – os últimos 16 bits do *header* contêm o ID na mensagem, usados para identificar mensagens duplicadas e para fazer a correspondência entre o pedido e a resposta;
- O resto da mensagem contém opções e valores que dependem da mensagem enviada.

3.3.2 Tipo de mensagem

Existem quatro tipos de mensagen, as quais contêm um identificador para detetar duplicados e torna-as mais fiáveis se requisitado [Tomislav D., Srdan K., Nenad G., 2011]:

Confirmable (CON) – este tipo de mensagens oferece fiabilidade sobre o protocolo UDP. Sempre que é enviada uma mensagem deste tipo ao servidor, segue com um *timeout* associado. Quando o servidor recebe a mensagem, envia um ACK com o mesmo identificador de mensagem do pedido, confirmando a receção do mesmo. Quando o cliente não recebe um ACK passado o tempo definido no *timeout*, este retransmite a mensagem e o *timeout* enviado nesta retransmissão é duplicado para garantir que a entrega do pacote seja efetuada. Estas retransmissões são efetuadas até ser recebido um ACK com o mesmo identificador de mensagem por parte do servidor como pode ser visto na Figura 18.

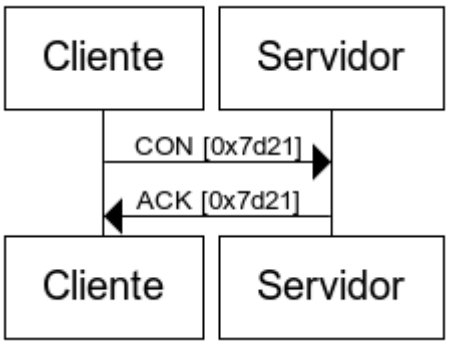


Figura 18 - Transmissões CON no CoAP

Non Confirmable (NON) – é uma mensagem que não precisa de ser confirmada pelo servidor. O cliente não tem como saber se o pedido foi entregue ao servidor. Como alternativa, o cliente pode enviar múltiplos pedidos. Este tipo de mensagens é indicado quando são feitos *polls* periódicos ao servidor.

Reset (RST) - este tipo de mensagens servem para indicar ao cliente que algo ocorreu no pedido efetuado ao servidor. O motivo que levou ao envio deste tipo de mensagem vem explicado no código da mensagem.

Acknowledge (ACK) – Tal como foi falado anteriormente, este tipo de mensagem tem de ser enviado para o cliente em resposta a uma mensagem CON. Caso o servidor necessite de

enviar alguma informação no *payload* da mensagem para o cliente, esta pode ser enviada juntamente com o ACK. Se o ACK e a informação da resposta forem enviados em mensagens separadas, o identificador destas mensagens será diferente, visto que, para cada retransmissão, é necessário um novo identificador. Quando o cliente receber a informação, tem que enviar um ACK, indicando que recebeu a resposta. Neste caso, o pedido e a resposta são identificados utilizando a opção *token*.

3.3.3 Opções

As opções do CoAP disponibilizam informação adicional para serem trocadas na mensagem. São compostas por um código numérico, formato, tamanho e, para algumas, existem valores por defeito. As opções estão divididas em dois grupos: as críticas, possuindo um código ímpar e as que não são críticas, correspondendo um código par. A diferença entre estes dois grupos distingue-se através da forma como são tratadas quando o servidor não reconhece as opções. Caso uma opção crítica não seja reconhecida pelo servidor, este envia uma mensagem RST. No caso de o servidor não reconhecer uma opção que não seja crítica, ignora essa opção. As opções possíveis são as seguintes [Mirko Rossini, 2011]:

Uri-Host, Uri-Port, Uri-Path, Uri-Query

Estas opções identificam, de forma precisa, o recurso a que se destina a mensagem. Com estas opções, o URI do recurso consegue ser construído de uma forma simples.

- URI-Host - identifica o nome do *host* onde o recurso está localizado. Pode ser um nome ou um endereço IP;
- Uri-Port - identifica a porta onde o pedido tem que ser realizado;
- Uri-Path - contém todos os componentes dentro do dispositivo que levam à localização do recurso;
- Uri-Query - permite especificar parâmetros adicionais à *query* do recurso.

Proxy-Uri, Proxy-Scheme

Permitem definir um *proxy* para encaminhar o pedido e a resposta. Assim, o pedido é efetuado a um recurso, passando pelo *proxy* especificado.

Content-Format

Indica o formato do conteúdo do *payload* da mensagem. Os valores suportados são os tipos de média da internet, também conhecidos como MIME (Multi-Purpose Internet Mail Extension).

Accept

A opção *Accept* permite ao cliente especificar qual o formato do conteúdo que este aceita receber como *payload* das suas respostas.

Max-Age

Esta opção indica quanto tempo o recurso pode permanecer na cache antes de o servidor considerar a sua atualização. Esta opção é utilizada para suportar cache dos recursos, o que leva a uma otimização da utilização dos dispositivos que contêm o recurso.

ETag

Esta opção indica a versão de uma representação de um recurso. Se o servidor suportar esta opção, deve marcar todos os valores de retorno. Quando o cliente usa esta opção o servidor deve confirmar se o valor do recurso ainda é válido sem ter de enviar o seu valor novamente.

Location-Path, Location-Query

Estas opções contêm o URI relativo e a *query string*, sendo utilizadas para indicar onde o recurso foi criado na resposta a um pedido POST.

If-Match, If-None-Match

O *IF-Match* é usado para efetuar um pedido condicional de um recurso. Pode ser usado juntamente com a opção *ETag* ou pode ser enviada simplesmente sem conteúdo. Se for enviado com a opção *ETag*, o servidor apenas responde se o *ETag* corresponder com um recurso válido. Se for usada sem conteúdo, o servidor apenas responde se o recurso existir.

O *IF-None-Match* tem um comportamento similar mas não leva qualquer tipo de valor, é utilizado para verificar se o recurso existe para prevenir que não se escreva num recurso já existente.

3.3.4 Core Link Format

A característica principal para uma comunicação máquina-máquina é a descoberta de recursos. O *Core Link Format* tem vindo a ser definido para permitir essa funcionalidade em redes REST com dispositivos de capacidades limitadas. A descoberta de recursos no *Core Link Format* é feita através da interface *well-know* (./well-know/core) de cada servidor, a qual retorna a descrição de todos os recursos disponíveis naquela máquina. Assim, todos os servidores têm um ponto de entrada comum que descreve os seus recursos. Todos os recursos são descritos através de um URI, um conjunto de atributos e, se necessário, relações com outros recursos.

De seguida, serão apresentados alguns parâmetros dos recursos que o *Core Link Format* disponibiliza:

- **Title:** este parâmetro fornece uma descrição legível do recurso;
- **Type:** contêm o tipo do recurso e apenas é permitido um tipo por recurso;
- **Resource Type (rt):** contêm a *string* usada para atribuir o tipo do recurso a uma aplicação. Enquanto o parâmetro Type descreve o tipo da mensagem de uma forma legível para humanos, o parâmetro resource type descreve o tipo da mensagem de uma forma legível por máquinas;
- **Maximum Size Estimated (sz):** se o tamanho do recurso exceder o MTU do protocolo UDP, este atributo é utilizado para indicar aproximadamente o tamanho que a resposta terá.

3.3.5 Erbium CoAP

Erbium CoAP é uma biblioteca CoAP disponibilizada no Contiki, desenvolvida por Matthias Kovatsch do ETH Zurich (Instituto Federal de Tecnologia de Zurique) em conjunto com o SICS (*Swedish Institute of Computer Science*). Esta biblioteca contém uma implementação CoAP, a

qual suporta *Blockwise Transfers* e *Observe.* Permite a descoberta de recursos, respostas separadas e bloqueamento de pedidos de clientes.

São disponibilizados três tipos distintos de recursos: *RESOURCE, PERIODIC_RESOURCE* e *EVENT_RESOURCE*. Os recursos do tipo RESOURCE são recursos normais que permitem pedidos *GET, PUT, POST e DELETE*. Os recursos do tipo *PERIODIC_RESOURCE* implementam o método Observe, permitindo que um cliente se registe no recurso de forma a receber atualizações periódicas do valor desse recurso. Os recursos *EVENT_HANDLER* permitem gerir o envio de dados caso algo pré-definido aconteça, despoletando assim um evento e enviando os dados ao cliente [Matthias K., Simon. D, Adam. D, 2011].

3.4 Plataforma – Contiki OS

O Contiki é um sistema operativo *Open Source* para sistemas de rede com baixa capacidade de memória. Criado por Adam Dunkels em 2002 e desenvolvido por uma equipa vasta de desenvolvedores pertencentes a várias entidades entre as quais se destacam a Atmel, Cisco, Enea, ETH Zurich, Redwire, RWTH Aachen University, Oxford University, SAP, Sensinode, SICS, ST MIcroelectronics, Zolertia e muitos outros.

3.4.1 Arquitetura

O Contiki é um sistema *Lightweight* baseado em eventos e contém uma arquitetura modular. O seu *Kernel* contém um escalonador de eventos que os envia para processos em execução. A execução de processos é desencadeada pelo *Kernel* para os processos ou por um sistema de *polling*, utilizado para evitar *race conditions*. O Contiki suporta eventos síncronos e assíncronos, sendo os primeiros enviados imediatamente para o processo que despoletou a sua execução. Os eventos assíncronos são tratados posteriormente pelo processo alvo. Na Figura 19, pode ser visto um esquema da arquitetura utilizada pelo Contiki [Edosoft, 2012].

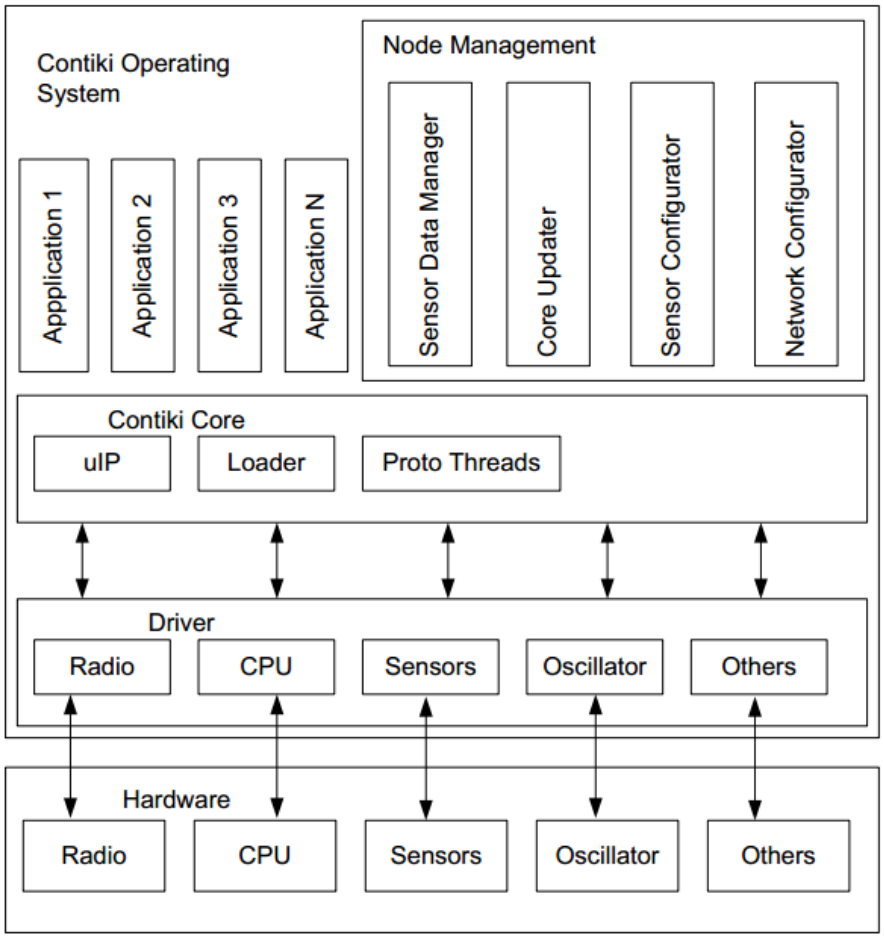


Figura 19 - Arquitetura do Contiki OS [3]

Este contém suporte para *Multi-Threading* e carregamento dinâmico de serviços individuais e sua reposição. A arquitetura deste sistema consiste num *Kernel* baseado em eventos onde os processos não executam sem a existência de um evento. Pode ser visto na Figura 20 que existe uma *stack* com uma *queue* de tarefas que são tratadas por um *Event Handler,* o qual comunica com o *kernel,* possibilitando a sua execução [Anuj Sehgal, 2011].

[3] Retirada de: ISN – Interoperable Sensor Networks – Delivarable, Contiky and TinyOS

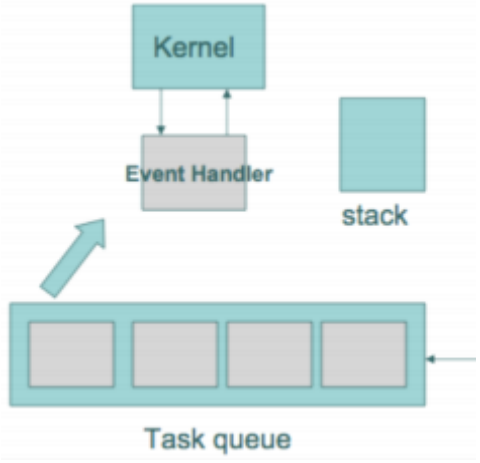


Figura 20 - Esquema de execução de eventos no Contiki

O Contiki também permite a implementação de *Multi-Threads*, que executam até existir uma declaração para o seu bloqueio. Como cada *thread* precisa da sua própria *stack*, este método é apenas aconselhado para computações que executem durante grandes períodos de tempo. Pode ser visto na Figura 21 o *Kernel* a lançar bloqueios às *Threads* enquanto estas executam.

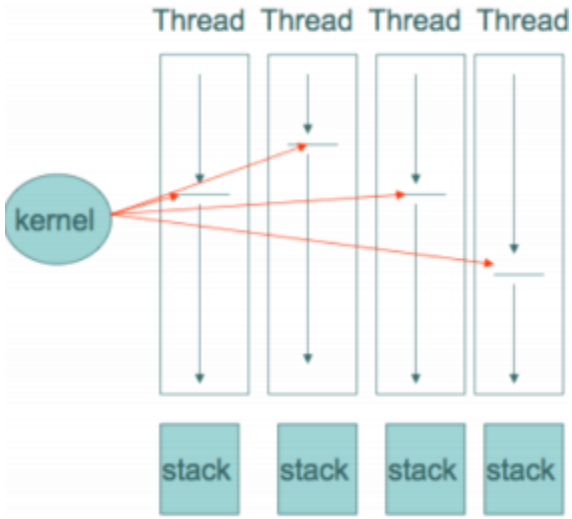


Figura 21 - Esquema de execução de threads no Contiki

3.4.2 Recursos do Sistema Operativo

O Contiki OS fornece vários recursos para serem utilizados em redes com dispositivos de capacidade limitada. Nos recursos fornecidos, encontramos implementações dos protocolos IEEE 802.15.4, IPv6, IPv4, CoAP, 6LoWPAN, entre muitos outros. Além das camadas

protocolares oferecidas pelo Contiki, este vem já com diversas aplicações para criar uma rede estável e precisa. Um dos recursos oferecidos pelo Contiki é o RPL-Border-Router, o qual fornece a capacidade de interligar vários dispositivos ao mundo exterior, fazendo o *routing* dos pacotes da rede.

O Contiki oferece bibliotecas capazes de realizar alocação de memória em dispositivos que apenas contêm poucos *kilobytes* de memória. Fornece uma *stack* IP completa, incluindo protocolos como UDP, TCP e HTTP.

Em adição a estes recursos, está disponível um simulador de redes intitulado Cooja. Este simulador é capaz de simular uma rede completa, usando virtualizações de plataformas reais. É útil para fazer o *debug* em redes complexas, preparando as aplicações para serem utilizadas em contextos reais.

3.4.3 Suporte de Plataformas

O Contiki foi desenhado para executar em dispositivos que são bastante limitados em termos de memória, energia, poder de processamento e largura de banda de comunicações. Sendo assim, o Contiki inclui suporte para as mais utilizadas plataformas de desenvolvimento.

Pode ser visto na Tabela 4, retirada da página oficial do Contiki[4], uma lista completa das plataformas suportadas pelo Contiki, listando a combinação de microcontroladores com antenas rádio e em que dispositivos estas combinações se encontram. Para cada dispositivo, é também indicado se existe suporte de executar em modo de simulação no Cooja.

Tabela 4 - Plataformas suportadas pelo Contiki

MCU/SoC	Radio	Platforms	Cooja simulation support
RL78	ADF7023	EVAL-ADF7023DB1	-
TI CC2538	Integrated	cc2538dk	-
TI MSP430x	TI CC2420	exp5438, z1	Yes
TI MSP430x	TI CC2520	wismote	Yes
Atmel AVR	Atmel RF230	avr-raven, avr-rcb, avr-	-

[4] http://www.contiki-os.org/hardware.html

MCU/SoC	Radio	Platforms	Cooja simulation support
		zigbit, iris	
Atmel AVR	TI CC2420	micaz	Yes
Freescale MC1322x	Integrated	redbee-dev, redbee-econotag	-
ST STM32w	Integrated	mb851, mbxxx	-
TI MSP430	TI CC2420	sky, jcreate, sentilla-usb	Yes
TI MSP430	TI CC1020	msb430	-
TI MSP430	RFM TR1001	esb	Yes
Atmel Atmega128 RFA1	Integrated	avr-atmega128rfa	-
Microchip pic32mx795f512l	Microchip mrf24j40	seed-eye	-
TI CC2530	Integrated	cc2530dk	-
RC2300/RC2301	Integrated	sensinode	-
6502	-	apple2enh, atari, c128, c64	-
Native	-	native, minimal-net, cooja	Yes

4 Implementação

De momento, não existe qualquer norma definida para a *Internet of Things* de forma a criar um sistema completo. O objetivo deste projeto passa por utilizar as tecnologias que mais próximas estão do desejado, de forma a demonstrar a sua utilização em sintonia. Dessa forma, foi construída uma rede composta por um router e vários sensores, definindo os protolocos de comunicação utilizados baseados em *Wireless Sensor Networks*.

4.1 Arquitetura

O sistema desejado é uma rede de sensores autónoma, comunicando entre si tendo como base de ligação um *router* que servirá como acesso para os pedidos efetuados aos sensores, como pode ser visto na Figura 22. Este esquema apresenta uma WSN com ligação a um Border Router a partir de um nó encarregue da comunicação com o Router. O Border Router tem como função permitir pedidos externos ao sistema, transformando os pedidos.

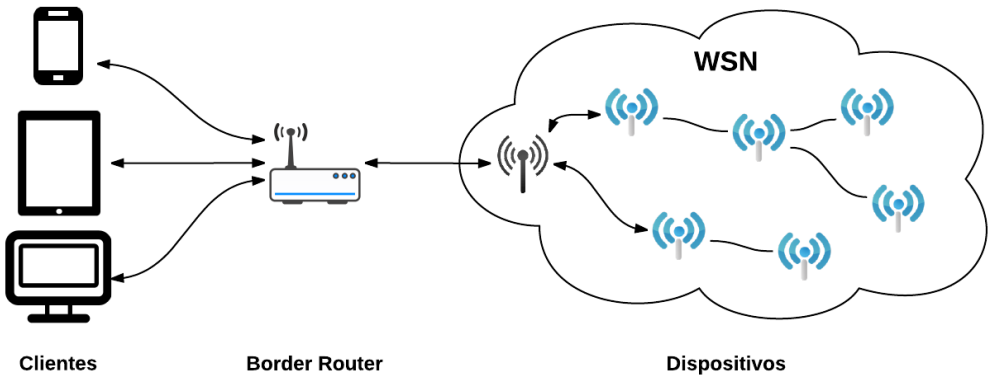
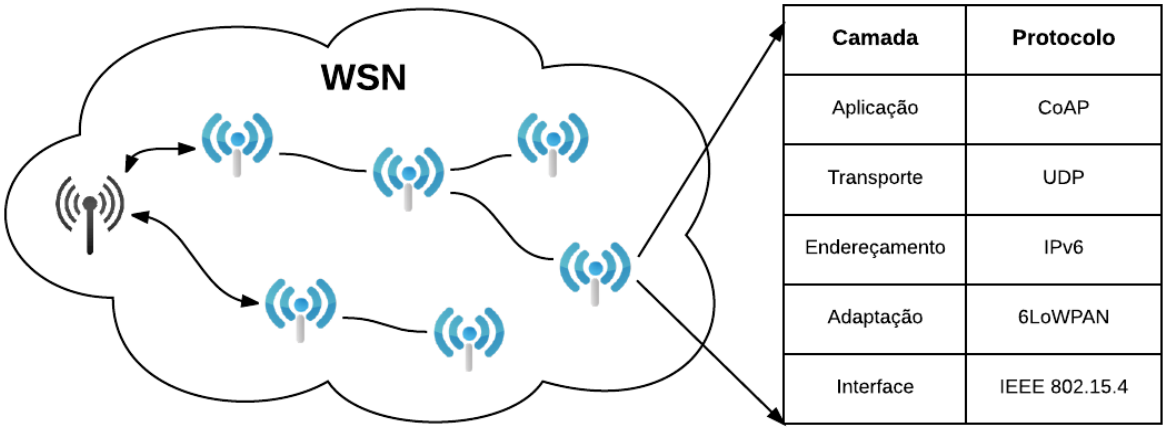


Figura 22 - Arquitetura do Sistema

Camadas Protocolares

De forma a obter o esquema presente na Figura 22, foi necessário definir a pilha protocolar a ser utilizada possibilitando uma comunicação entre os sensores sem grandes custos energéticos e de processamento. Tendo em base as tecnologias apresentadas na secção anterior, foi definida a seguinte pilha protocolar, presente na Figura 23.



Camada	Protocolo
Aplicação	CoAP
Transporte	UDP
Endereçamento	IPv6
Adaptação	6LoWPAN
Interface	IEEE 802.15.4

Figura 23 - Protocolos WSN utilizados

Border Router

Após definidas as comunicações na WSN, é necessário tratar das comunicações externas, resolvendo os pedidos à rede. Para isso foi utilizado um *Border Router* com a capacidade de implementar um sistema RPL (*IPv6 Routing Protocol for Low power and Lossy Networks*). Este

protocolo especifica o modo como o fluxo de tráfego é controlado em redes limitadas. De forma a implementar este sistema foi instalado o protocolo RPL num nó da rede, ligado ao *Border Router*. Este nó tem a função de controlar as mensagens trocadas pela rede enquanto o *Border Router* reencaminha os pedidos externos através deste nó para conseguir comunicar com a rede. Pode ser visto um esquema desta ligação na Figura 24.

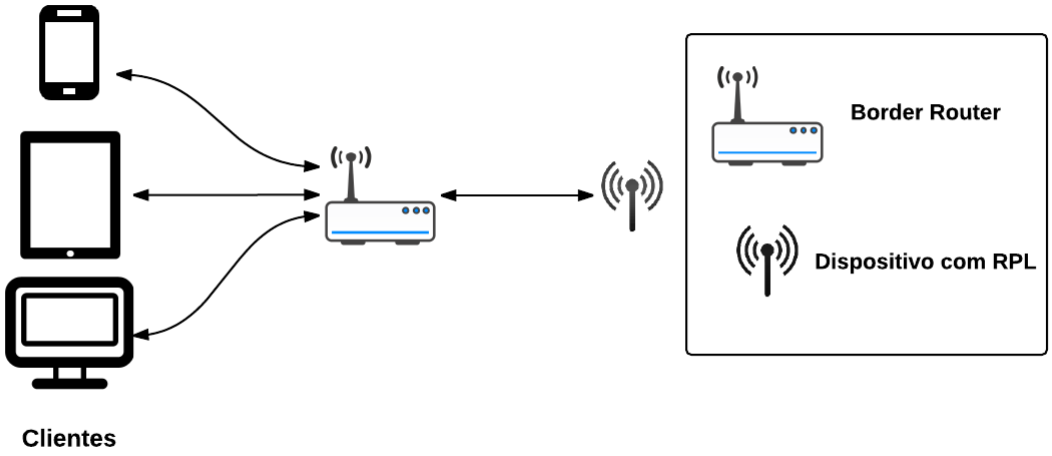


Figura 24 - Border Router

4.2 Hardware utilizado

4.2.1 Crossbow TelosB

O Crossbow TelosB (Figura 25) é uma plataforma *Open Source* que reúne todas as condições necessárias para estudos em laboratório sobre redes de sensores. É compatível com o protocolo IEEE 802.15.4, com um rácio de 250kbps, um microcontrolador TI MSP430 de 8 MHz com 10kB de RAM e uma antena integrada na placa. Permite também a sua programação via USB (*Universal Serial Bus*) e tem disponíveis sensores de temperatura, luminosidade e humidade. Além destas caraterísticas, o TelosB contém também 48kB de memória *flash* de forma a permitir que sejam inseridos pequenos programas.

Figura 25 - Plataforma Crossbow TelosB[5]

Na Tabela 5, temos a informação relativa aos sensores incorporados na placa (Luminosidade, Temperatura e Humidade).

Tabela 5 - Caraterísticas dos sensores presentes no TelosB

Sensores	**Caraterísticas**
Alcance de luz visível	320 nm -> 730 nm
Alcance de Infra Vermelhos	320 nm -> 1100 nm
Alcance do Sensor de humidade	0 -> 100% RH
Alcance do Sensor de temperatura	-40ºC -> 123.8ºC
Interface	IEEE 802.15.4

Na Figura 26, pode ser vista a distribuição dos principais componentes presentes na plataforma TelosB onde se encontram a antena integrada, o microcontrolador, os sensores, entre outros.

[5] Retirada de: http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf

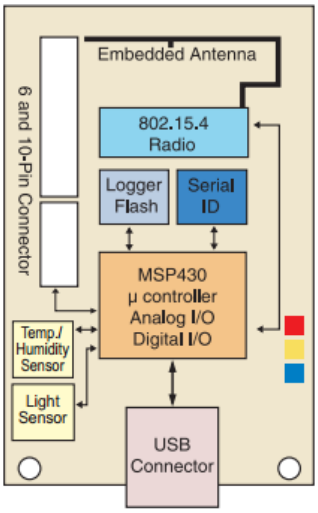


Figura 26 - Principais componentes do TelosB[6]

4.2.2 Advanticsys XM1000

O Advanticsys XM1000 (Figura 27) é uma plataforma baseada nas especificações do TelosB referido acima. Equipado com um microcontrolador TI MSP430 com 8kB de RAM, o qual recebeu uma redução face ao TelosB, e 116kB de memória *flash*, um aumento significativo face ao TelosB. Tal como o TelosB, esta plataforma contém três sensores: luminosidade, temperatura e humidade com caraterísticas idênticas.

Figura 27 - Plataforma Advanticsys XM1000[7]

[6] Retirada de: http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf
[7] Retirada de: http://www.advanticsys.com/shop/asxm1000-p-24.html

4.2.3 BeagleBoard BeagleBone Black

A *BeagleBoard BeagleBone Black* (Figura 28) é uma plataforma para desenvolvedores com um custo reduzido. Esta plataforma contém o sistema operativo Linux. Equipada com um processador Sitara de 1GHz e 512MB de RAM, esta plataforma torna-se bastante versátil para colocar aplicações mais complexas que exijam um poder de processamento maior.

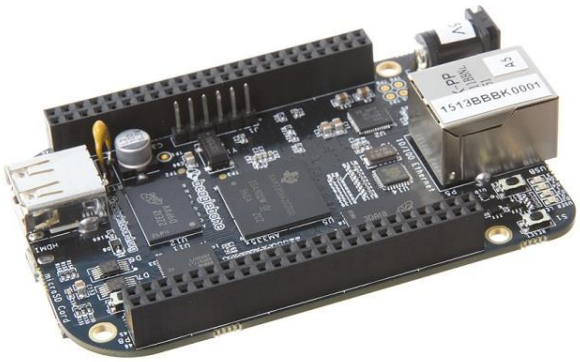


Figura 28 - Plataforma BeagleBoard BeagleBone Black[8]

Uma descrição mais detalhada sobre esta plataforma pode ser encontrada na Figura 29, retirada da página oficial da BeagleBoard, onde encontramos as descrições dos componentes.

[8] Retirada de: http://elinux.org/Beagleboard:BeagleBoneBlack

	Feature	
Processor	Sitara AM3358BZCZ100 1GHz, 2000 MIPS	
Graphics Engine	SGX530 3D, 20M Polygons/S	
SDRAM Memory	512MB DDR3L 800MHZ	
Onboard Flash	4GB, 8bit Embedded MMC	
PMIC	TPS65217C PMIC regulator and one additional LDO.	
Debug Support	Optional Onboard 20-pin CTI JTAG, Serial Header	
Power Source	miniUSB USB or DC Jack	5VDC External Via Expansion Header
PCB	3.4" x 2.1"	6 layers
Indicators	1-Power, 2-Ethernet, 4-User Controllable LEDs	
HS USB 2.0 Client Port	Access to USB0, Client mode via miniUSB	
HS USB 2.0 Host Port	Access to USB1, Type A Socket, 500mA LS/FS/HS	
Serial Port	UART0 access via 6 pin 3.3V TTL Header. Header is populated	
Ethernet	10/100, RJ45	
SD/MMC Connector	microSD , 3.3V	
User Input	Reset Button Boot Button Power Button	
Video Out	16b HDMI, 1280x1024 (MAX) 1024x768,1280x720,1440x900 ,1920x1080@24Hz w/EDID Support	
Audio	Via HDMI Interface, Stereo	
Expansion Connectors	Power 5V, 3.3V , VDD_ADC(1.8V) 3.3V I/O on all signals McASP0, SPI1, I2C, GPIO(69 max), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 4 Serial Ports, CAN0, EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)	
Weight	1.4 oz (39.68 grams)	
Power	Refer to Section 6.1.7	

Figura 29 - Caraterísticas da BeagleBone Black[9]

4.3 Configuração dos dispositvos

De forma a criar a *Wireless Sensor Network* pretendida, é necessário definir a camada protocolar a utilizar nos dispositivos e proceder à sua configuração. O cenário definido, tal como referido anteriormente, inclui o uso das mais recentes tecnologias. Para tal, é necessário programar os sensores pertencentes à rede com estas. O Contiki oferece-nos a capacidade de definir e programar os sensores como mostrado de seguida.

Para definir os sensores, utilizaram-se as plataformas Advanticsys XM1000 visto terem maior capacidade de memória *flash,* de forma a conseguir introduzir nestes a biblioteca referente ao protocolo CoAP.

[9] Retirada de: http://elinux.org/Beagleboard:BeagleBoneBlack

4.3.1 Protocolos utilizados nos dispositivos

Foi definida a pilha protocolar a ser utilizada pelos sensores, tal como listada na Tabela 6, e descrita a sua utilidade.

Tabela 6 - Protolos utilizados nos dispositivos

Camada	Protocolo
Aplicação	CoAP
Transporte	UDP
Endereçamento	IPv6
Adaptação	6LoWPAN
MAC	CSMA
Radio Duty Cycling	ContikiMAC
Física	IEEE 802.15.4

- Camada Física

  O Contiki oferece-nos as configurações necessárias para rapidamente definirmos quais os protocolos a serem utilizados nos sensores. De forma a utilizarmos o protocolo IEEE 802.15.4 na camada física, necessitamos de o definir no ficheiro de configuração do projeto [Código 1].

```
#define NETSTACK_CONF_FRAMER  framer_802154
```

Código 1 - Definição do protocolo IEEE 802.15.4

- Radio Duty Cycling

  Esta camada protocolar define a utilização da antena do recetor. Aqui é definida a utilização do ContikiMAC que permite que o dispositivo desligue totalmente o aparelho transmissor, ligando-o periodicamente. É utilizado para otimização energética no aparelho e pode ser facilmente definida [Código 2]. Também pode ser definido o rácio a utilizar no protocolo RDC [Código 3].

```
#define NETSTACK_CONF_RDC     contikimac_driver
```

Código 2 - Definição do protocolo RDC

```
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 8
```

Código 3 - Rácio do protocolo RDC

- MAC

  Aqui é definido o protocolo CSMA (*Carrier Sense Multiple Access),* que permite evitar a colisão de pacotes em redes com múltiplos acessos. Desta forma, consegue-se controlar o acesso à rede, bloqueando a transmissão enquanto outras existirem [Código 4].

```
#define NETSTACK_CONF_MAC     csma_driver
```

Código 4 - Definição do protocolo MAC

- Adaptação

  É possível indicar ao sensor para utilizar 6LoWPAN na camada de adaptação se se tratar de uma rede IPv6 [Código 5].

```
#define NETSTACK_CONF_NETWORK sicslowpan_driver
```

Código 5 - Definição do protocolo 6LoWPAN

- Endereçamento

  Na criação de um novo projeto, é possível definir rapidamente se o sensor vai utilizar a rede IPv6 no seu *Makefile*. Aqui apenas é necessário colocar a configuração [Código 6] a verdadeiro e o Contiki configura e compila automaticamente tudo o que é necessário.

```
WITH_UIP6=1
```

Código 6 - Definição do protocolo IPv6

- Aplicação

  Na secção abaixo, é mostrado como criar e configurar os recursos CoAP a utilizar no sensor. Sendo necessário compilar a REST Engine que o Contiki fornece e a versão a utilizar do protocolo CoAP [Código 7].

```
APPS += er-coap
APPS += rest-engine
```

Código 7 - Definição do protocolo CoAP

4.3.2 Definição de recursos CoAP utilizados nos dispositivos

O Contiki contém uma biblioteca CoAP, Erbium CoAP, e um motor REST, "REST Engine", bastante completos, disponibilizando vários recursos e utilizando baixa memória RAM e baixa ROM.

Ao utilizar esta biblioteca, consegue-se rapidamente configurar os recursos CoAP que são necessários. De forma a retirar o melhor partido dos sensores, foi definido que estes teriam os seguintes recursos disponíveis:

- Humidade – Tem informações sobre a humidade relativa e tem os métodos GET e Observe disponíveis;
- Temperatura – Contém a temperatura em graus Celsius e possui os métodos GET e Observe;
- Info – Composto por dois recursos (Nome e Localização) e tem disponível o método GET, devolvendo a informação contida no Nome e Localização. Este recurso contém também implementado fragmentação de pacotes pois a informação contida ultrapassa o tamanho de um pacote 6LoWPAN;
- Nome – Contém o nome identificador do dispositivo (Ex.: Candeeiro, Aquecedor), tem disponível os métodos GET e POST;
- Localização – Contém a localização do dispositivo (Ex.: Cozinha) e tem disponível os métodos GET e POST;
- Luminosidade – Devolve o nível de luminosidade de luz visível e infravermelhos, tendo disponível os métodos GET e Observe;
- Alternar LED – Liga e desliga o LED contido no sensor, tendo disponível apenas o método POST.

Declaração de recursos

Foram utilizados dois tipos de recursos no presente projeto: *RESOURCE* e *PERIODIC_RESOURCE*. O primeiro é um recurso que consegue receber pedidos normais, transmitindo apenas uma resposta; o segundo pedido trata pedidos periódicos tais como o *Observe,* registando o cliente como observador do recurso.

1. RESOURCE

   A definição de um recurso contém o nome do recurso, uma informação e as funções que permitem obter e modificar o resultado. No excerto de Código 8, pode ser vista declaração do recurso Nome, que tem como identificador o campo *res_name*, contém um título e o tipo de recurso como segundo parâmetro; os dois seguintes campos indicam as funções que tratam os pedidos GET e POST.

```
RESOURCE(res_name,
        "title=\"Name\";rt=\"Info\"",
        res_get_handler1,
        res_post_handler1,
        NULL,
        NULL);
```

Código 8 - Declaração de um recurso Erbium CoAP

   Para este projeto, foram definidos três tipos de *Resource Types* (rt): Info, Sensor e Control. Estes tratam os campos de texto informativos, os tipos de sensores (luminosidade, temperatura e humidade) e os atuadores (alternar LED).

2. PERIODIC_RESOURCE

   Na definição de um recurso periódico, é necessário incluir o nome do recurso, a informação do mesmo e os métodos para tratar os eventos; os dois últimos parâmetros incluem um intervalo de tempo, de forma a permitir à REST Engine invocar periodicamente a função que é fornecida como último parâmetro, como pode ser visto no excerto de Código 9, que representa a declaração do recurso Temperatura.

```
PERIODIC_RESOURCE(res_temperature,
          "title=\"Temperature in Celsius (supports
          TEXT,XML,JSON)\";obs",
          res_get_handler,
          NULL,
          NULL,
          NULL,
          5 * CLOCK_SECOND,
          res_periodic_handler);
```

Código 9 - Declaração de um recurso periódico Erbium CoAP

Definição de recursos

Para definir um recurso, é necessário estruturá-lo num ficheiro próprio escrito na linguagem C. Este ficheiro contém informação sobre os métodos disponibilizados por aquele sensor (GET, POST, PUT, DELETE, OBSERVE). As operações disponíveis no sistema são as seguintes:

1. Método GET

   Para possibilitar a operação GET num dispositivo é necessário implementar uma função que trate o evento da forma apropriada. O método criado contém cinco parâmetros (*Request, Response, Buffer, Preferred Size e Offset*). Estes campos, por ordem de enumeração, contêm o pacote recebido com o pedido, o pacote pré construído a ser devolvido, um *buffer* para colocar a informação a ser devolvida (Ex.: temperatura ambiente), um tamanho que permite definir o tamanho máximo de dados enviados em cada transmissão e um offset para controlar o conteúdo já enviado ao cliente, no caso de fragmentação de pacotes. Pode ser visto no excerto de Código 10 o cabeçalho da função previamente descrita.

```
static void res_get_handler(void *request, void *response,
uint8_t *buffer, uint16_t preferred_size, int32_t *offset);
```

Código 10 - Método que trata o pedido GET

   Nesta função, é necessário consultar o tipo de dados pedidos pelo cliente (*PLAIN_TEXT, XML ou JSON*) usando a função disponibilizada pela REST Engine. No excerto de Código 11, observamos que o método consulta o pedido recebido de forma a obter o tipo de dados (*Accept*).

```
REST.get_header_accept(request, &accept);
```

Código 11 - Obter o tipo de dados do cabeçalho

   Após obter o tipo de dados pretendido pelo utilizador, é efetuada a construção da resposta. No excerto de Código 12, pode ser vista uma mensagem do tipo JSON a ser colocada dentro do *buffer* fornecido.

```
snprintf((char *)buffer, REST_MAX_CHUNK_SIZE,
"{\"Name\":{\"name\":\"%s\"}}", name);
```

Código 12 - Método para colocar a mensagem no campo buffer

   É agora necessário terminar o pacote de resposta a ser enviado definindo o tipo de dados a devolver e colocando o *buffer* no *payload* do mesmo, como pode ser visto no excerto de Código 13. O método que coloca o *payload* pretendido na resposta tem

como parâmetros o pacote de resposta a ser enviado, o *buffer* preenchido com os dados e o tamanho do mesmo.

```
REST.set_header_content_type(response,
REST.type.APPLICATION_JSON);

REST.set_response_payload(response, buffer, strlen((char
*)buffer));
```

Código 13 - Finalização do pacote a enviar no método GET

2. Método POST

De forma a disponibilizar a opção de um cliente efetuar um pedido POST, é necessário implementar um método que trate os pedidos em concordância. Este método é semelhante ao método implementado, contendo os mesmos campos, como pode ser visto no excerto de Código 14.

```
static void res_post_handler1(void *request, void *response,
uint8_t *buffer, uint16_t preferred_size, int32_t *offset);
```

Código 14 - Método que trata o pedido POST

Neste método, é efetuado o tratamento da informação recebida. Na implementação efetuada, temos dois tipos diferentes de pedidos POST: um que recebe a informação da localização ou do nome do dispositivo; eum outro que permite o controlo do LED embebido na plataforma. No caso do primeiro tipo de pedido, é necessário obter o *payload* transportado pela mensagem recebida, como pode ser visto no excerto de Código 15.

```
len = REST.get_request_payload(request, (const uint8_t
**)&incoming);
```

Código 15 - Obter a mensagem do payload

Este método devolve o tamanho do *payload* contido no pacote recebido. Com este valor é apenas necessário retirar o valor transportado na mensagem, percorrendo-a com o tamanho obtido. De seguida é chamada a função GET presente no recurso de forma a enviar a resposta do recurso modificado.

O segundo tipo de POST implementado apenas ativa a função, sendo pedido ao Contiki para modificar o estado do LED utilizando o método presente no excerto de Código 16.

```
leds_toggle(LEDS_RED);
```

Código 16 - Ligar/Desligar um LED

3. Método Observe

   Para implementar o método Observe deve ser criada uma função capaz de receber pedidos periódicos e tratá-los consoante os requisitos. Pode ser visto no excerto de Código 17 a assinatura do método a criar.

```
static void res_periodic_handler(void);
```

Código 17 - Método que trata os pedidos periodicos

   Este método é responsável por realizar as verificações necessárias para a devolução de novos valores. No caso do recurso de temperatura, são comparados os valores previamente conhecidos e verificado se existiu alguma alteração. Se se verificar uma alteração no valor, este método notifica os subscritores do sucedido utilizando a função presente no excerto de Código 18, a qual utiliza o método GET implementado para devolver a informação.

```
REST.notify_subscribers(&res_temperature);
```

Código 18 - Notificação de subscritores do método Observe

4. Fragmentação de pacotes

   Quando o recurso recebe um pedido de um cliente e a resposta a devolver ultrapassa o tamanho máximo que um pacote pode transportar, é necessário fragmentar o pacote. Utiliza-se esta metodologia no recurso Info de forma a enviar o nome e localização do dispositivo.

   Como qualquer pedido GET, precisamos de copiar a informação para o *buffer* fornecido no cabeçalho da função para o enviar no pacote. Não sendo possível enviar a informação toda de uma vez, são copiados apenas os dados até atingir tamanho máximo definido. Para isto, é utilizado o campo *preferred_size* de forma a copiar apenas parte da informação. Ao copiar a informação para o *buffer,* é necessário atribuir a quantidade de dados já processados para o campo offset de forma a registar quais os dados que faltam enviar.

   Desta forma, os pacotes são enviados até ser sinalizado que não restam mais informações a transmitir, colocando o valor do *offset* a -1.

Inicialização de recursos

Para a criação do programa a ser inserido no sensor, é necessário criar um ficheiro escrito na linguagem C que importe os recursos previamente definidos, utilizando o nome identificador atribuído na sua declaração, como pode ser visto no excerto de Código 19.

```
extern resource_t
res_temperature,
res_humidity,
res_info,
res_name,
res_location;
res_toggle;
res_light;
```

Código 19 - Recursos utilizados pelo dispositivo

Após a importação de todos os recursos é necessário criar o processo que irá lidar com os seus pedidos. O Contiki oferece vários comandos para a criação de processos e *threads*, sendo necessário atribuir um identificador ao processo e indicar a sua inicialização, como visto no excerto de Código 20.

```
PROCESS(tese2014, "Tese 2014");

AUTOSTART_PROCESSES(&tese2014);
```

Código 20 - Criação de um processo

Aqui, o processo é inicializado automaticamente quando o dispositivo configurado é ligado. É agora necessária a criação de uma *thread* que consiga executar o processo, utilizando o método fornecido pelo Contiki para inicialização de threads [Código 21].

```
PROCESS_THREAD(tese2014, ev, data)
```

Código 21 - Criação de uma thread

A nova *thread* fica então responsável por executar o processo, inicializando-o. Aqui, é inicializada a REST Engine que recebe os pedidos dos clientes, e efetuada a ativação dos recursos com o seu caminho relativo. Os caminhos relativos definidos permitem ao cliente aceder ao recurso configurado no dispositivo (Ex.: IPDispositvo/sensors/temperature), como pode ser visto no excerto de Código 22.

```
rest_init_engine();

rest_activate_resource(&res_temperature, "sensors/temperature");
rest_activate_resource(&res_humidity, "sensors/humidity");
rest_activate_resource(&res_info, "info");
```

```
rest_activate_resource(&res_name, "info/name");
rest_activate_resource(&res_location, "info/location");
rest_activate_resource(&res_toggle, "actuators/toggle");
rest_activate_resource(&res_light, "sensors/light");
```

Código 22 - Ativação dos recursos CoAP

Depois de efetuada esta configuração, é indicado ao processo que deve aguardar por eventos de forma a satisfazer os pedidos dos clientes. O comando utilizado para este efeito pode ser visto no excerto de Código 23.

```
while(1) {
PROCESS_WAIT_EVENT();
```

Código 23 - Colocar um processo à escuta de eventos

4.3.3 Programação dos sensores

Para a programação dos sensores, tem de se utilizar o compilador fornecido pelo Contiki para a plataforma desejada. O Contiki possui uma lista de plataformas disponíveis e as suas configurações, sendo que, para cada plataforma, existem vários ficheiros de configuração pré-definidos que são utilizados para conseguir inserir os programas desenvolvidos no dispositivo. Neste projeto, utilizou-se a plataforma XM1000, a qual não tem suporte nativo no sistema operativo Contiki. É então necessário efetuar o *download* dos controladores da plataforma do seu site oficial[10] e introduzi-los no Contiki.

Tem de ser atribuído um identificador de nó e um endereço MAC distinto a cada dispositivo, de forma a serem identificados na rede. Para atribuir, estes valores, é necessário utilizar o comando presente no excerto de Código 24 onde se seleciona o programa a inserir no dispositivo (*burn-nodeid.upload*), os identificadores escolhidos e o dispositivo no qual se pretende introduzir estes valores (*TARGET=xm1000; MOTE=1*).

```
make TARGET=xm1000 burn-nodeid.upload nodeid=158 nodemac=158 MOTE=1
```

Código 24 - Definir identificador do nó e MAC

Após a utilização desse comando, o sensor executa o programa carregado e obtém os identificadores. É agora necessário carregar o dispositivo com os recursos CoAP pretendidos e protocolos a utilizar. Para tal, devemos executar o comando presente no excerto de Código 25, que compila o programa criado com as configurações protocolares definidas no dispositivo.

[10] http://www.advanticsys.com/wiki/index.php?title=XM1000

```
make TARGET=xm1000 tese2014.upload MOTE=1
```

Código 25 - Compilação do programa final para o dispositivo

Com o programa carregado, o dispositivo está pronto a ser inserido numa rede e a receber pedidos por parte dos clientes. Caso seja pretendido observar a execução do sensor, podemos fazê-lo a partir de uma consola onde são listados os *logs* definidos, utilizando o comando presente no excerto de Código 26. Podemos observar na Figura 30 um log da inicialização de um dispositivo, que contém o endereço do mesmo e algumas das configurações fornecidas.

```
make TARGET=xm1000 login MOTE=1
```

Código 26 - Consultar os logs enviados por um dispositivo

```
root@vascopc:~/contiki-2.6/examples/Tese_Coap_Temperatura# make TARGET=xm1000 login
../../tools/sky/serialdump-linux -b115200 /dev/ttyUSB0
connecting to /dev/ttyUSB0 (115200) [OK]
0#U000+000000Rime started with address 0.18.116.0.19.183.111.244
MAC 00:12:74:00:13:b7:6f:f4 Contiki 2.6 started. Node id is set to 141.
nullmac nullrdc, channel check rate 128 Hz, radio channel 26
Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7400:13b7:6ff4
Starting 'Tese 2014'
```

Figura 30 - Log de inicialização de um dispositvo

4.4 Configuração da Gateway da rede

Na Secção 4.1, foi referido que o projeto tem uma *gateway* para gerir as comunicações dos sensores e receber os pedidos dos clientes. Esta Gateway é composta pela plataforma BeagleBone Black, a qual contém os componentes necessários para albergar um servidor web e fazer o roteamento de pedidos. Para o roteamento, é utilizada a plataforma TelosB que comunica com os dispositivos e faz a sua gestão na rede. Podemos ver na Figura 31 que estas duas plataformas estão conectadas via USB. A placa TelosB, configurada a partir do Contiki, contém o programa RPL-Border-Router, que utiliza o protocolo RPL (Routing Protocol for Low power and Lossy Networks).

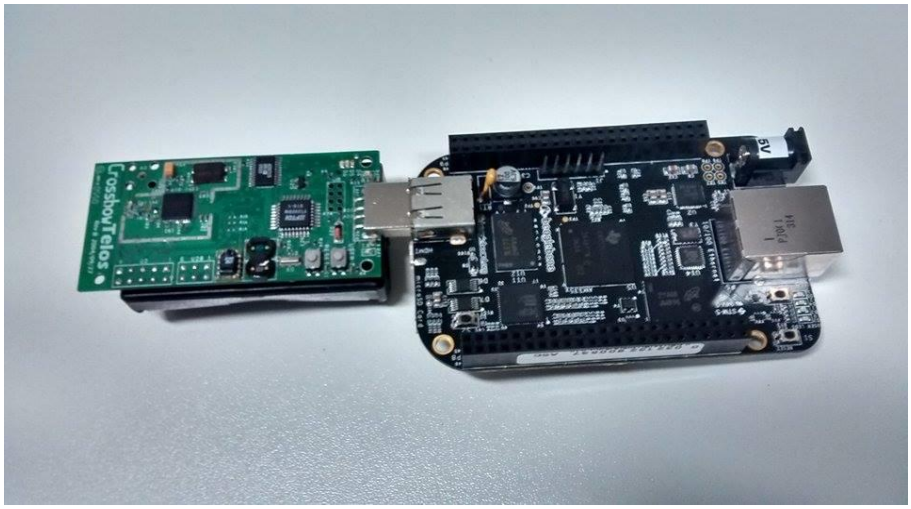


Figura 31 - Hardware utilizado pela Gateway

4.4.1 Configuração do RPL Border Router

O *RPL Border Router* é uma aplicação desenvolvida pela equipa do Contiki que implementa o conceito de RPL estudado pela IETF. O *RPL Border Router* foi utilizado na plataforma TelosB para fazer a gestão da rede dos dispositivos.

Configurações protocolares

Utilizando os ficheiros de configuração fornecidos pelo Contiki é necessário indicar que devem ser compilados todos os ficheiros necessários para o dispositivo conseguir trabalhar em redes IPv6 utilizando o comando definido no excerto de Código 27.

```
WITH_UIP6=1
```

Código 27- Definição do protocolo IPv6

Para o dispositivo conseguir comunicar corretamente com os restantes dispositivos presentes na rede, estes deves conter as mesmas camadas protocolares. São utilizados os mesmos comandos de configuração utilizados na Secção 4.4.1 para configurar o router, definindo assim a camada protocolar.

O Contiki fornece uma forte implementação do protocolo RPL, não sendo necessárias alterações adicionais às anteriormente listadas. Para o propósito deste projeto, foi estudada uma forma de conseguir que o programa possuísse um recurso CoAP encarregue de devolver a lista dos nós registados na rede. Para isso, deve-se importar a biblioteca de CoAP para o

projeto e a sua REST-Engine. Depois é criado um recurso com o método GET que utiliza a lista de nós registados e a devolve via CoAP. Esta implementação do recurso CoAP no PRL Border Router foi obsoleta do projeto devido à baixa memória ROM presente na plataforma TelosB, a qual não consegue albergar todas as configurações e bibliotecas necessárias. A plataforma XM1000 não possui memória RAM suficiente para conseguir arrancar a REST-Engine juntamente com os protocolos de roteamento.

Programação do RPL Border Router

A programação do RPL Border Router é semelhante à programação dos sensores definida na Secção 4.3.3, sendo necessária a utilização do compilador fornecido pelo Contiki para a plataforma Sky (TelosB).

Tal como na programação dos sensores, também é necessário atribuir um identificador de nó e um endereço MAC à plataforma TelosB. Para isso, deve ser utilizado o comando disponível no Contiki listado no excerto de Código 28, que permite a atribuição do mesmo. A diferença entre a programação deste identificador é o alvo definido, sendo modificado para "sky" para serem utilizados os ficheiros de configurações da plataforma TelosB.

```
make TARGET=sky burn-nodeid.upload nodeid=158 nodemac=158 MOTE=1
```

Código 28 - Definir identificador do nó e MAC

Depois de indicados os identificadores, tem de ser carregado para a plataforma o software relativo ao *RPL Border Roter*. Para isso, é executado o comando definido no excerto de Código 29, o qual efetua a compilação do projeto criado e o carrega para a plataforma com as configurações protocolares efetuadas.

```
make TARGET=sky border-router.upload MOTE=1
```

Código 29 - Compilação do RPL Border Router para o dispositivo

4.4.2 Inicialização do RPL Border Router na BeagleBone

A BeagleBone Black é utilizada para receber pedidos externos à rede de sensores e os reencaminhar para a mesma. Para isto, esta deve ser capaz de suportar uma ligação ethernet e conter as aplicações necessárias para o reencaminhamento dos pedidos. Foi utilizada uma

distribuição de Linux desenvolvida para a BeagleBone contendo as funcionalidades mínimas para a sua utilização. Esta distribuição está disponível no site oficial da Beagle Board[11].

Configurações

Assumindo que está a ser utilizada a distribuição Linux acima referida, são poucas as configurações que têm de ser efetuadas para obter o funcionamento da Gateway. De seguida, encontram-se listadas todas as aplicações e configurações necessárias para o seu funcionamento:

- Tunslip6 – Uma aplicação utilizada como uma *bridge* entre o tráfego IP de um *host* para outra rede. Cria uma interface virtual de rede no host (tun) e utiliza SLIP (*Serial Line Internet Protocol*) para encapsular e retransmitir tráfego IP entre as ligações da *Serial Line*;
- Script de inicialização – Foi criado um pequeno script de forma a inicializar os serviços necessários para o funcionamento da rede, executando a aplicação acima referida na inicialização do host.

Inicialização

Para iniciar o serviço de *router*, deve ser utilizada a aplicação *tunslip6*, uma aplicação que cria um túnel virtual na interface de rede Tun0, permitindo o encaminhamento de pedidos da rede eth0 para os sensores. Para inicializar este serviço, deve ser utilizado o comando definido no excerto de Código 30. Podemos observar na Figura 32 o resultado obtido na inicialização do RPL Border Router, o qual mostra o seu endereço de IP entre outras configurações.

```
tunslip6 aaaa::1/64
```

Código 30 - Inicializar a aplicação Tunslip6

[11] http://elinux.org/BeagleBoardUbuntu

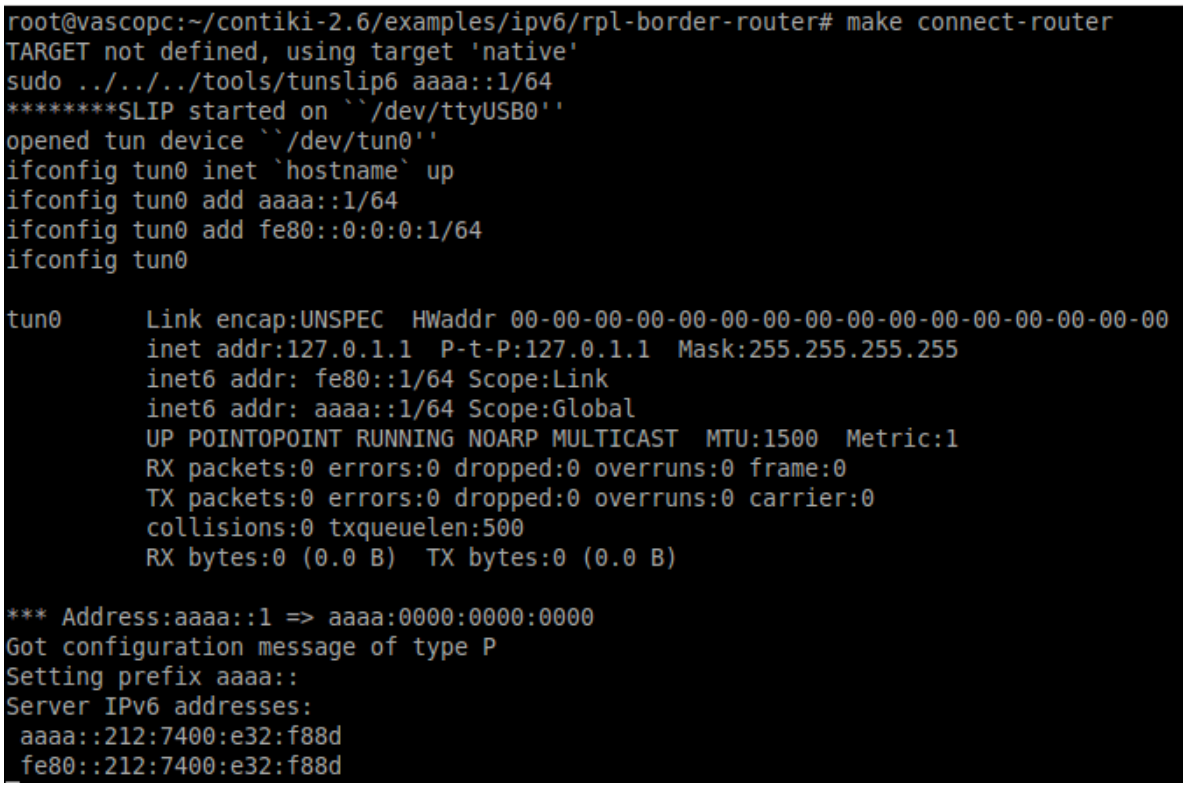


Figura 32 - Inicialização do RPL Border Router

5 Testes

Neste capítulo encontram-se os testes efetuados ao sistema final. São descritos testes funcionais e não funcionais utilizando diferente *software* na sua realização. Foram utilizadas as plataformas Crossbow TelosB como RPL-Border-Router e Advanticsys XM1000 como dispositivos pertencentes à rede com recursos CoAP. Foi utilizada a ferramentas *Copper* (cu), que é uma extensão do navegador *Mozilla Firefox*, de forma a realizar e receber pedidos CoAP nos testes referentes às secções 5.2, 5.3 e 5.4. Nas secções 5.4 e 5.6 foi criado um cliente utilizado *NodeJS* (*Node-CoAP*) para fazer os pedidos aos dispositivos.

5.1 Descoberta de dispositivos

O *RPL-Border-Router* fornece uma página web com a lista dos nós registados na sua rede e a sua rota de encaminhamento (*via*). Na Figura 33observamos que estão três nós registados na rede na divisória *Routes* e o seu caminho de roteamento. Pode ser visto que os nós com o endereço *aaaa::212:7400:13b7:7e2d e aaaa::212:7400:13b7:6ff4* comunicam diretamente com o *RPL-Border-Router*, pois estes têm o seu próprio endereço como caminho de reencaminhamento. O restante nó está também registado no *router,* mas este apenas o

consegue contactar via o endereço aaaa::212:7400:13b7:*7e2d*, tendo de usar *hops* para conseguir comunicar.

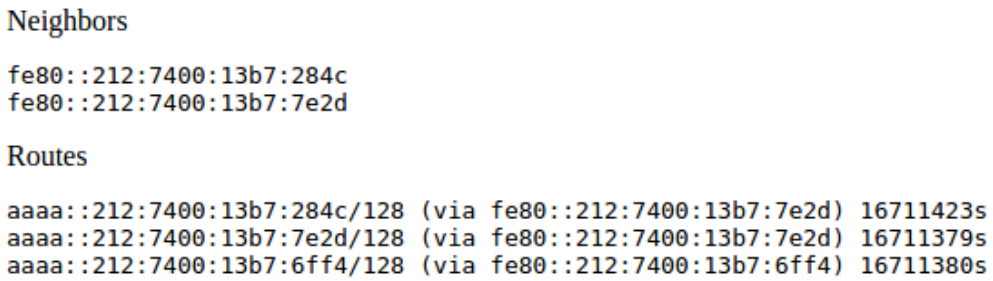


Figura 33 - Registo de nós na rede

5.2 Descoberta de Recursos

Na extensão do *Mozilla Firefox*, *Copper*, é disponibilizada a ferramenta *Discover*, a qual permite realizar um pedido ao *Well-Known Core* do dispositivo preparado para receber pedidos CoAP. Este pedido devolve uma lista completa do tipo de recursos que o dispositivo disponibiliza. Podemos ver na Figura 34 o resultado ao pedido da listagem de recursos ao nó com o endereço *aaaa::212:7400:13b7:7e2d*. Pode também ser efetuado um pedido GET ao recurso *.well-known/core* (presente em todos os dispositivos que implementam CoAP), o qual mostra o resultado apresentado na Figura 35, que contém as descrições de cada recurso.

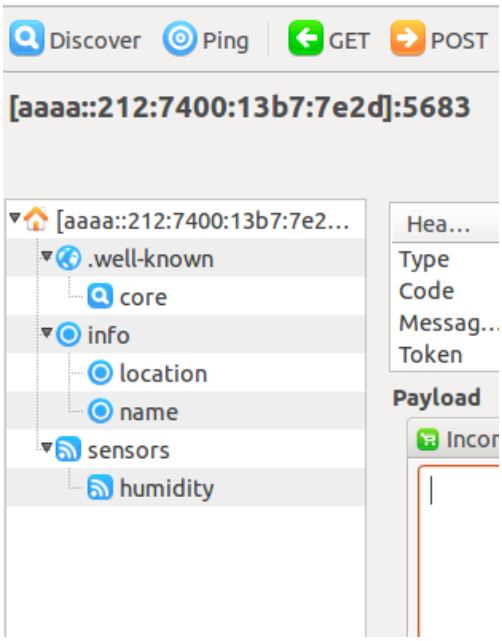


Figura 34 - Ferramenta *Discover* e Recursos listados

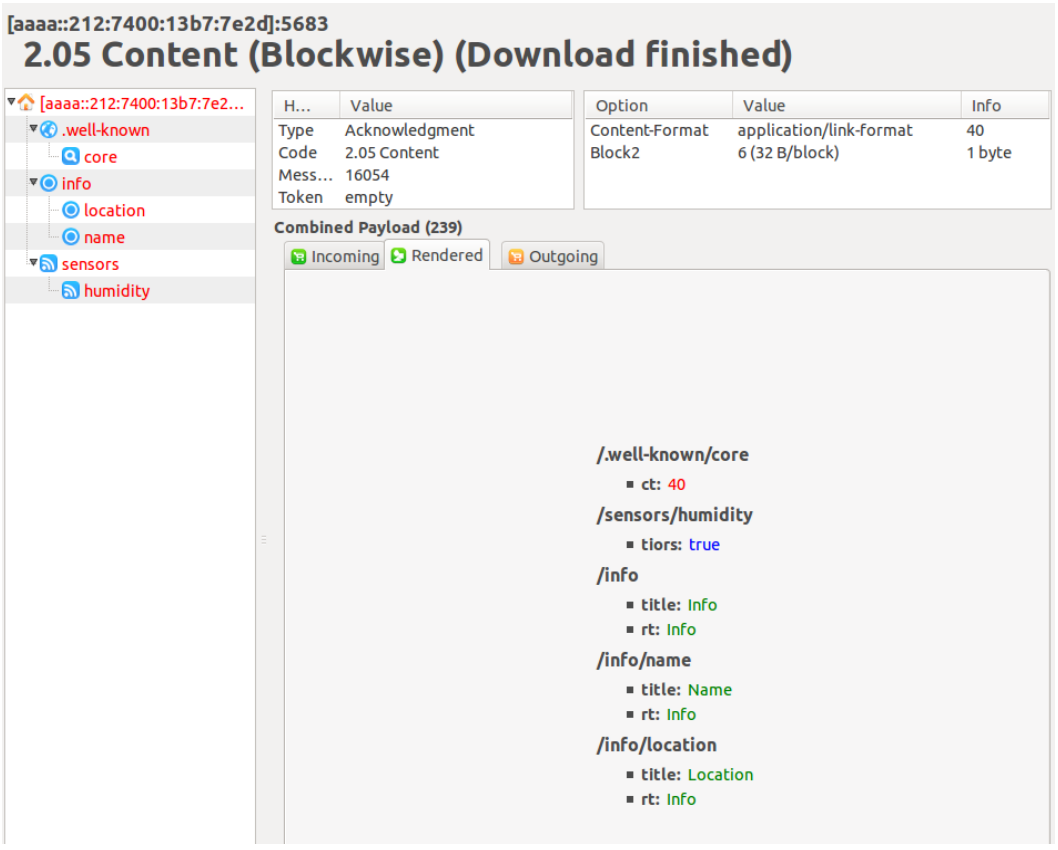


Figura 35 - Resultado do pedido ao *well-known Core*

5.3 Pedidos aos dispositivos

Os pedidos aqui mostrados consistem em pedidos realizados a partir da extensão *Copper* do navegador *Mozilla Firefox*. Foram efetuados pedidos para mostrar as opções definidas nos dispositivos a funcionar como descrito nas próximas subsecções.

5.3.1 Pedido GET

Foi definido que os dispositivos suportariam pedidos em três formatos de resposta diferentes (*PLAINTEXT, XML e JSON*). De forma a realizar estes pedidos na extensão *Copper* foi necessário definir o tipo de dados a receber. A ferramenta suporta esse tipo de configuração utilizando o seu *Debug Control*, definindo o campo *Accept* [Figura 36] com o formato de dados que se pretende receber.

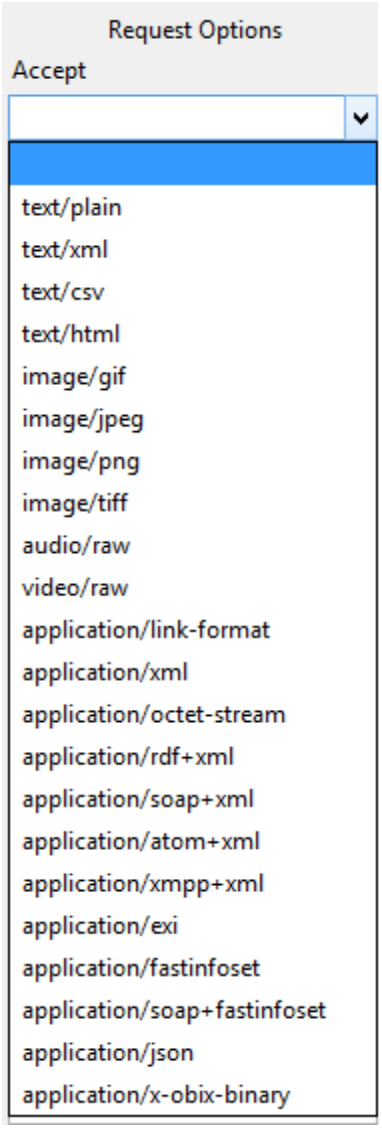


Figura 36 - Parâmetros Accept

- **Texto**

  Definindo o campo *Accept* como *text/plain*, consegue-se realizar um pedido GET ao dispositivo para obrigar (caso o dispositivo o tenha configurado) a resposta em formato de texto. Pode ser visto na Figura 37 a resposta a este pedido.

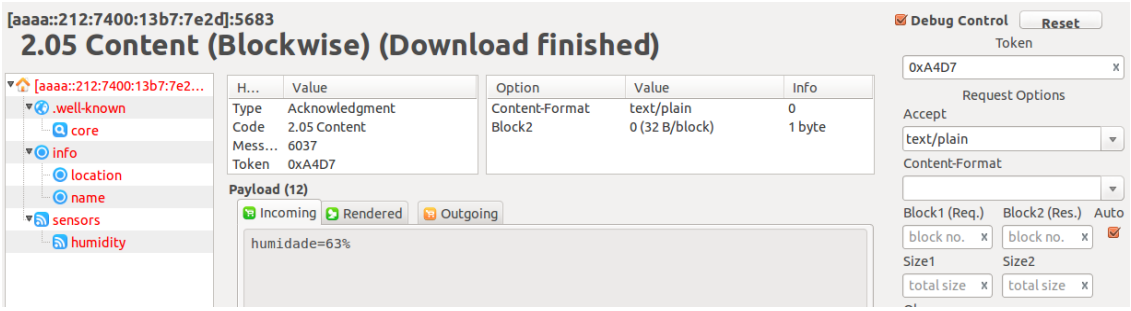


Figura 37 - Resposta a pedido com *Accept text/plain*

- **XML**

  Para receber a resposta em formato XML, o campo *Accept* deve ser definido como *application/xml*. Na Figura 38 observamos a resposta a este pedido com o formato XML.

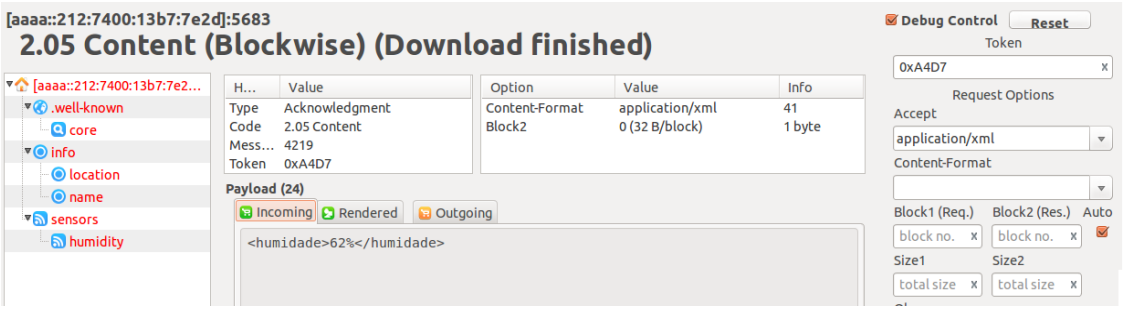


Figura 38 - Resposta a pedido com *Accept application/xml*

- **JSON**

  O formato JSON foi pré-definido em todos os dispositivos como formato de mensagem a responder. Desta forma podemos apenas efetuar um pedido GET normal a qualquer recurso e este devolve o resultado no formato JSON diretamente sem ser necessário definir o campo *Accept* do pedido. Na Figura 39 vemos o resultado de um pedido deste tipo.

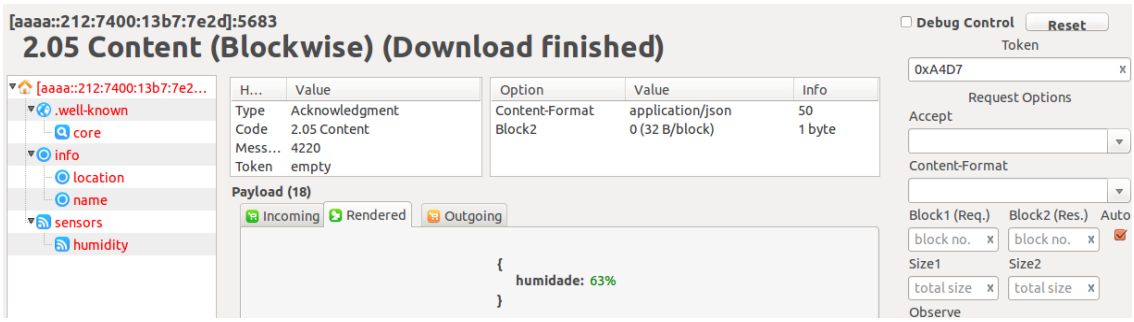


Figura 39 - Resposta a pedido com *Accept application/json*

5.3.2 Pedido POST com *Payload*

Para enviar um pedido POST com dados definidos no seu *Payload* usamos a ferramenta *Copper*, utilizando a opção *Outgoing* juntamente com a opção *Content-Format*. A opção *Content-Format* foi definida a *text/plain* e foi efetuado uma atualização ao conteúdo do recurso Location do dispositivo, escrevendo o seu conteúdo na opção *Outgoing*. Pode ser visto na Figura 40 o resultado desta operação.

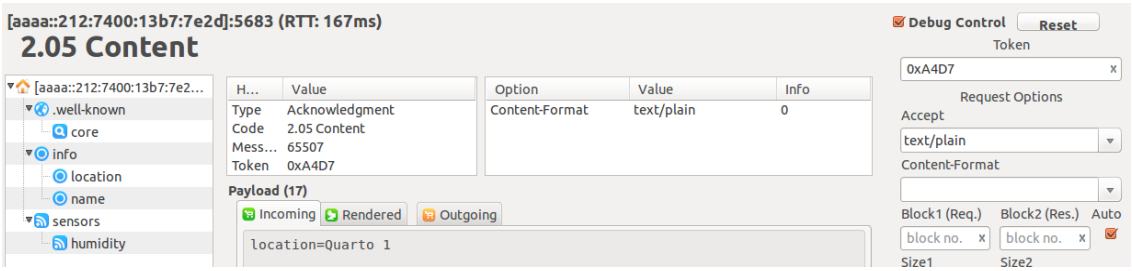


Figura 40 - Resposta a pedido POST com *Payload*

5.4 Análise de um pacote

Para a análise de um pacote transmitido foi utilizada a ferramenta *Wireshark*, um analisador de rede para *Unix* e *Windows*. Foram realizados um pedido GET e um pedido POST para mostrar o seu conteúdo no pacote.

Pacote GET

Podemos observar na Figura 41 o pacote os dois cabeçalhos do *Wireshark* referentes ao pedido GET efetuado ao recurso humidade, que contêm o tempo relativo de envio e receção

na coluna Time, o seu endereço de origem e destino, o protocolo utilizado, o tamanho da mensagem trocada e seu tipo (CON e ACK) juntamente com outras informações da mensagem.

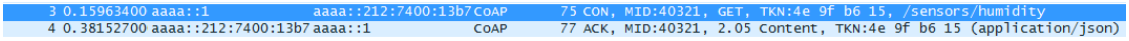


Figura 41 - Cabeçalhos *Wireshark*

Na Figura 42 temos informações mais detalhadas sobre o pedido GET enviado tais como a versão do protocolo IP utilizado (IPv6), o protocolo de transporte (UDP) e o protocolo de aplicação (CoAP). Vemos também que no protocolo CoAP vão definidas as opções *URI-Path*, a qual define o caminho do recurso que se está inquirir, e a opção *Content-Format* que define qual o tipo de formato de mensagem que foi pedido.

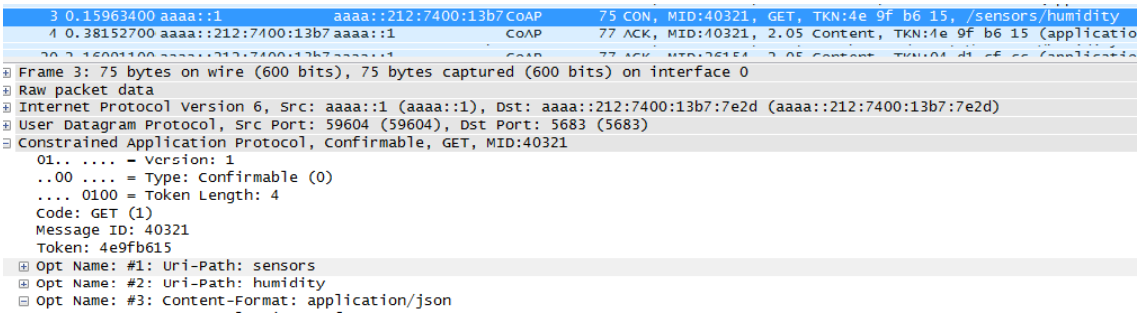


Figura 42 - Pedido CoAP *Wireshark*

A Figura 43 representa o pacote de resposta ao pedido efetuado, do tipo ACK, que contém informações sobre os protocolos utilizados. No protocolo CoAP vêm também definas as opções *Content-Format*, definindo o formato do conteúdo da mensagem, e a opção *Payload* que contém a resposta ao pedido efetuado.

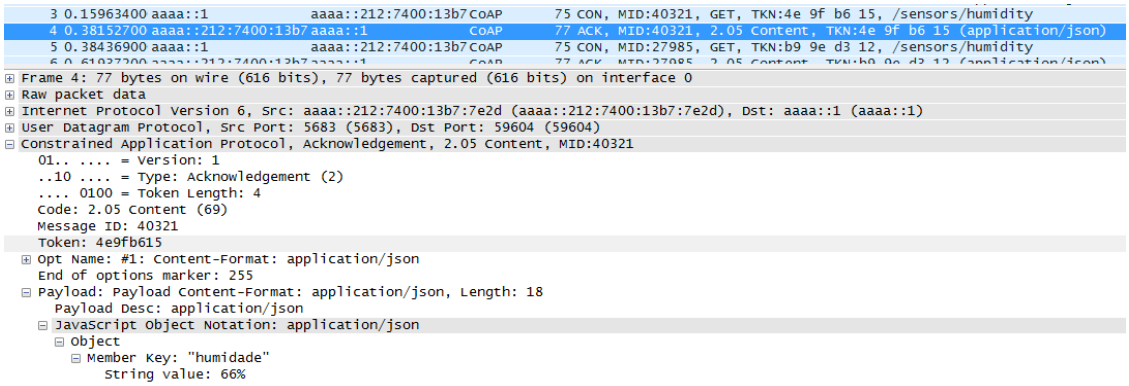


Figura 43 - Resposta CoAP *Wireshark*

5.5 Tempos de resposta

Para efetuar o registo do tempo de resposta de um dispositivo quando é inquirido por um cliente a um determinado recurso foi criada uma pequena aplicação em *NodeJS* utilizando a biblioteca *Node-CoAP*. Esta aplicação realiza mil pedidos sequenciais ao sensor, sendo estes pedidos capturados pela ferramenta *Wireshark*. Depois de terminados, os pedidos foram analisados e criado um gráfico *timestamp*/latência para observar o comportamento do dispositivo ao longo dos pedidos.

Pedidos GET ao recurso *Location*

O recurso *Location* devolve uma *String* no formato JSON contendo a informação sobre a localização do recurso. Observando a Figura 44 podemos concluir que o dispositivo responde a todos os pedidos, contendo períodos de maior exaustão, aumentando o seu tempo de resposta. Demorou-se um total de 121200ms a responder a todos os pedidos. Os picos de latência deram-se a aproximadamente a cada sequência de 43 pedidos.

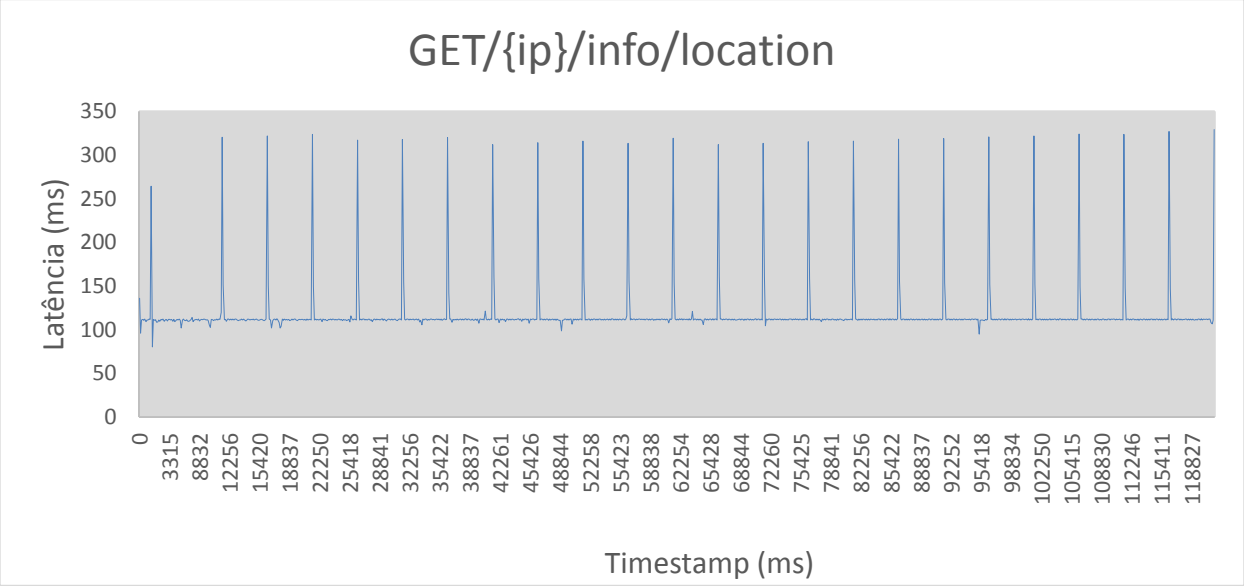


Figura 44 - Pedidos GET *Location*

Pedidos POST ao recurso *Name*

O pedido efetuado neste teste envia uma mensagem CoAP com *Payload* para atualizar o valor do recurso *Name*. Observando a Figura 45 podemos concluir que os pedidos se mantêm estáveis. Pela observação da imagem nota-se que existem alguns picos no tempo de resposta

e que, após cada pedido desses existe um tempo de resposta menor para a devolução do pedido. O dispositivo demorou 113560ms a responder aos mil pedidos efetuados, um decréscimo significativo face ao teste anterior de 121200ms.

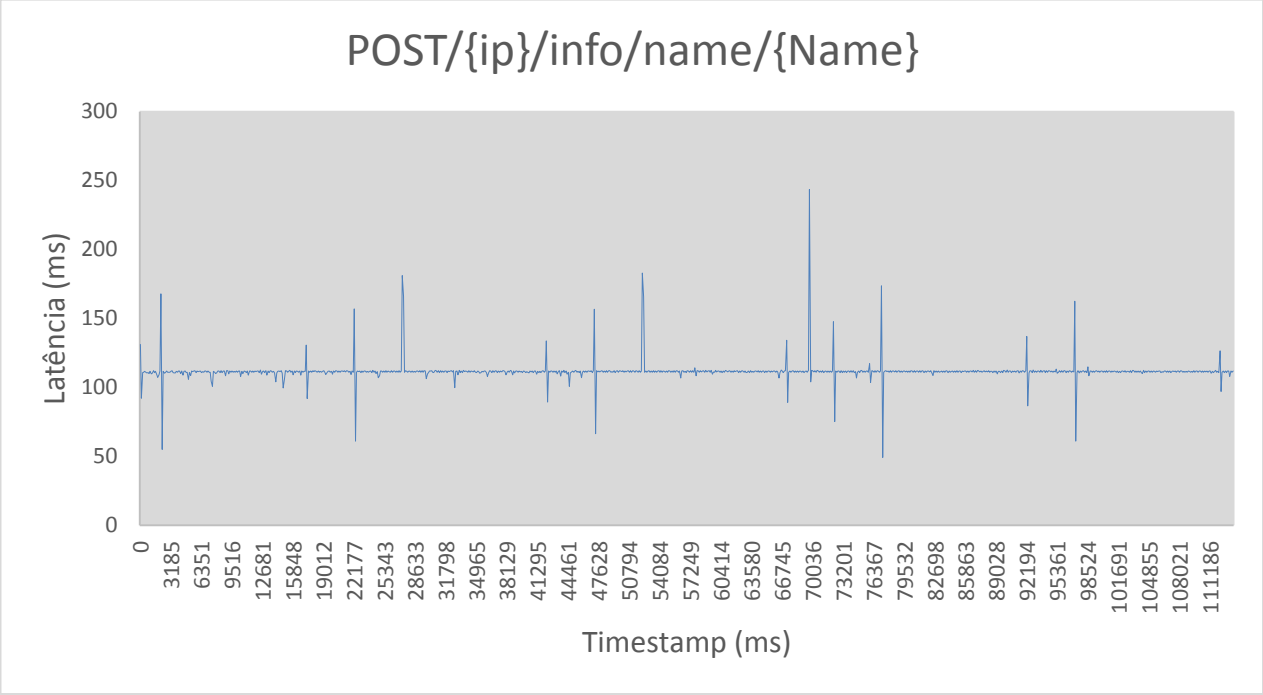


Figura 45 - Pedidos POST *Name*

Pedido GET ao recurso *Temperature*

Foram efetuados mil pedidos sequenciais ao recurso temperatura para calcular os seus tempos de resposta. Este recurso mostrou-se o mais problemático durante os testes, conseguindo apenas dar resposta a 873 pedidos. Além desta perda de pacotes, notou-se também um tempo de resposta relativamente grande quando comparado aos outros recursos. Observando a Figura 46 nota-se maior irregularidade nos tempos de respostas, tornando-se um recurso inconstante para a plataforma utilizada. O dispositivo demorou 543872ms a dar resposta aos pedidos efetuados. Devido ao enorme tempo de resposta que este recurso atinge, poder-se-ia melhorar a implementação das respostas na plataforma, por forma a melhorar este aspeto.

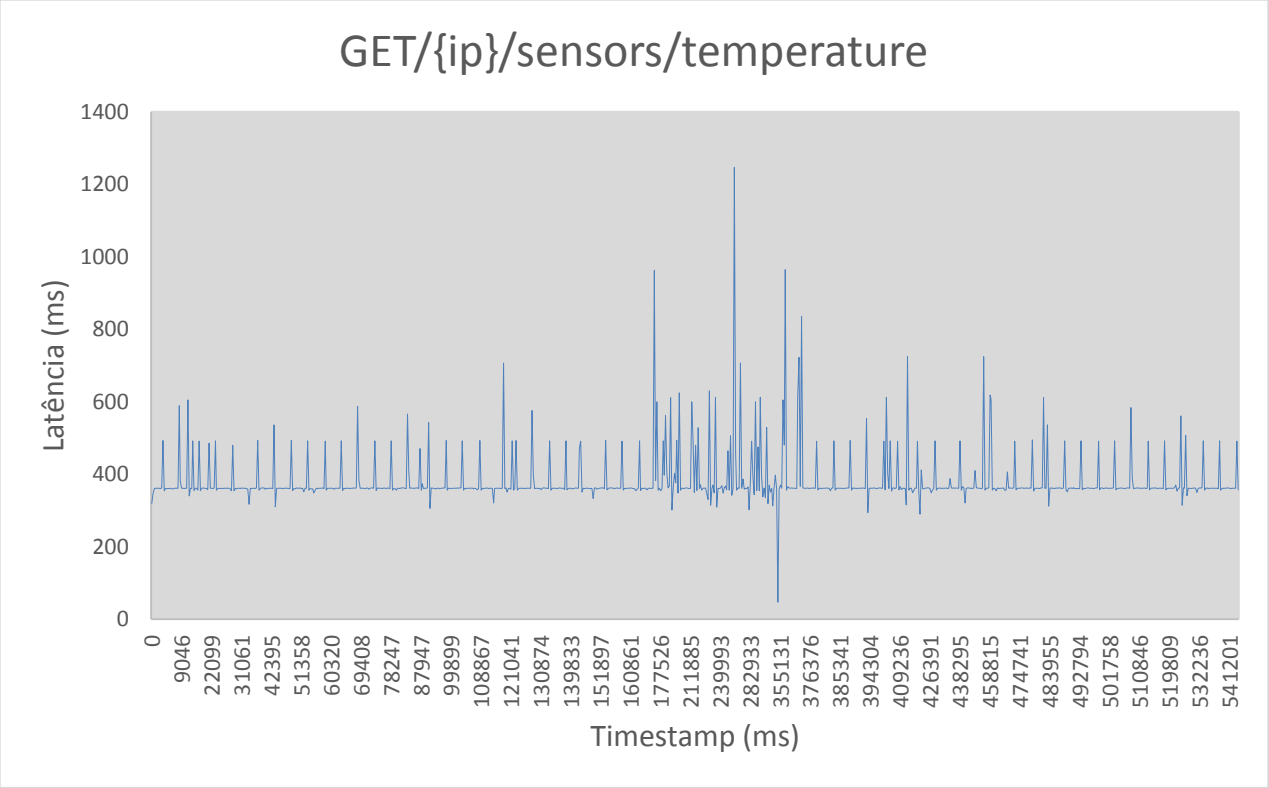


Figura 46 - Pedidos GET temperatura

Pedidos POST ao recurso *Toggle*

O recurso *toggle* oferece bons tempos de resposta na transmissão de mensagens, não necessitando de *Payload*, fazendo apenas atuar um LED. Tal como os outros testes, este demonstra certos picos de tempos de resposta. Este recurso obteve um total de 121854ms ao responder aos mil pedidos efetuados.

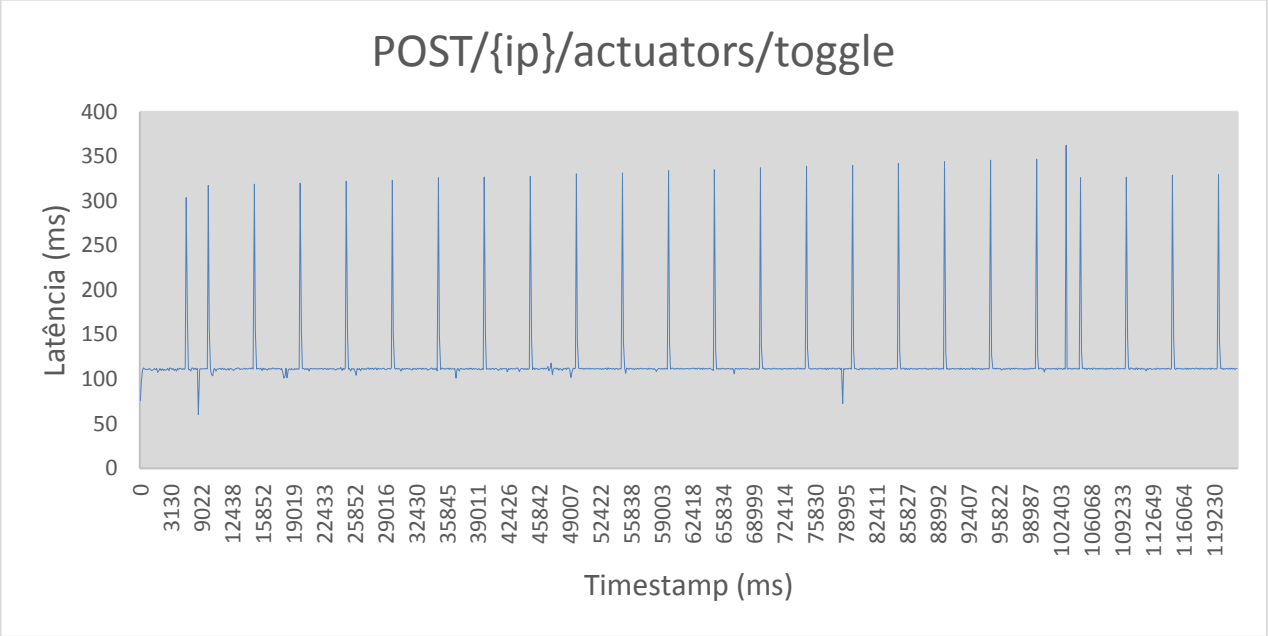


Figura 47 - Pedido POST *toggle*

5.6 Tempos médios de resposta

Consoante os testes realizados na subsecção anterior, foi realizado uma média dos tempos de resposta dos mil pedidos efetuados, transmitindo-se no gráfico presente na Figura 48. No gráfico podemos ver os seus tempos de reposta em milissegundos como eixo dos YY. No eixo dos XX, estão presentes as chamadas efetuadas ao dispositivo, indicando o recurso a ser analisado.

Os recursos com menor tempo de resposta são os que implementam o método POST (118ms, 112ms, 112ms) face aos pedidos GET efetuados (118ms, 117ms, 224ms, 381ms, 113ms).

Figura 48 - Tempos médios de resposta dos recursos

6 Conclusões e Trabalho Futuro

6.1 Contribuições

Nesta tese foi proposta uma arquitetura que permite criar rapidamente uma *Wireless Sensor Network*, conectando qualquer dispositivo à *Internet of Things*. Foi desenvolvida possibilitando a sua escalabilidade, permitindo interconectividade entre dispositivos e interações com o mundo físico com base no uso de dispositivos de baixa capacidade de processamento e reduzida capacidade energética. Foram utilizados os protocolos com mais ênfase na atualidade face aos temas IoT e WSN, tendo estes dado indícios do seu crescimento no mercado, aumentando o seu desenvolvimento. O objetivo principal no estudo destes protocolos é a criação de um padrão de rede bem definido. Pensando neste objetivo foram então abordadas as seguintes tecnologias:

- IEEE 802.15.4 – O uso deste protocolo tem vindo a crescer ao longo dos anos, alargando o seu desenvolvimento. Foi utilizado nesta tese como primeira camada do modelo TCP/IP;
- IPv6 – O IPv6 é um recurso necessário na criação de redes inerentes ao tema, dando uso à sua enorme gama de endereços e estabilidade. Foi utilizado no projeto como segunda camada do modelo TCP/IP
- 6LoWPAN – Utilizado como uma camada de adaptação, o 6LoWPAN é um recurso necessário a incluir numa rede deste tipo, possuindo fortes capacidades de

fragmentação e compressão de pacotes, permitindo o uso do protocolo IPv6 por dispositivos de capacidades reduzidas;

- CoAP – Este protocolo torna-se cada vez mais usado pela *Internet of Things* como meio de troca de mensagens, servindo a camada de aplicação do modelo TCP/IP. Com grandes capacidades de transferências em bloco e compressão de conteúdo, o CoAP é o protocolo mais promissor para utilização em dispositivos com pequena capacidade;
- Contiki – o Contiki é um sistema operativo que consegue reunir todos estes protocolos, possibilitando a rápida configuração de uma *Wireless Sensor Network*, recorrendo às suas ferramentas. Foi utilizado para definir todas as tecnologias utilizadas pelos dispositivos.

6.2 Avaliação e Trabalho Futuro

Durante a realização deste trabalho, surgiram várias situações que dificultaram a sua realização e que condicionaram a forma como este foi conduzido. Foi tentado implementar o sistema no *hardware* TI CC2538DK, que contém boas capacidades de memória e de processamento. Após várias tentativas na configuração do sistema, a abordagem foi abandonada devido à falta de suporte pelos sistemas operativos, nomeadamente o Contiki, que ainda carecem de desenvolvimento nessa parte. Adotou-se então o uso das plataformas Crossbow TelosB e Advanticsys XM1000 mas, devido à sua baixa capacidade, não permitiram o desenvolvimento de aplicações mais complexas.

No fim do projeto obteve-se uma boa solução, utilizando os dispositivos com capacidade reduzida, tentando que estes conseguissem integrar uma rede como qualquer outro. O resultado final foi satisfatório, tendo sida montada e utilizada uma rede que enfrentou os objetivos proposto no início do projeto, mesmo face à sua desvantagem tecnológica.

Os resultados obtidos nos testes efetuados, como mostrado na secção 5.5, mostram que existem alguns problemas de desempenho quando o dispositivo é sobrecarregado. Estes problemas podem ser melhorados, recorrendo à melhora da implementação de respostas a este tipo de pedidos, ou mesmo implementando uma fila de mensagens no *BorderRouter*/*Gateway* de forma a não sobrecarregar o dispositivo com os pedidos, efetuando-os quando o sensor estiver num momento de menor sobrecarga.

Como trabalho futuro propõe-se a configuração do *RPL-Border-Router* com um recurso do tipo CoAP para se conseguir obter uma comunicação inteiramente por esse protocolo, permitindo maior escalabilidade da aplicação. Propõe-se também a configuração de um *router* externo equipado com IPv6, definindo as suas rotas de encaminhamento para ser conseguida uma Gateway totalmente independente, capaz de ligar a rede criada com qualquer outra rede. Assim, definindo as tabelas de encaminhamento do novo *router*, consegue-se transmitir pedidos de qualquer nó ligado à sua rede para a rede 6LoWPAN criada.

Referências

[Alan Ott, 2012]	Alan Ott. Wireless Networking with IEEE 802.15.4 and 6LoWPAN
[Anuj Sehgal, 2011]	Anuj Sehgal. Embedded Operating Systems – Part II – The Contiki OS
[Dave Evans, 2011]	Dave Evans. The Internet of Things - How the Next Evolution of the Internet is Changing Everything
[Edosoft, 2012]	Edosoft Factory, S.L . ISN – Interoperable Sensor Networks – Delivarable, Contiky and TinyOS
[Gee K., Chee K., Nor K., Borhanuddin M., 2010]	Gee Keng EE*, Chee Kyun Ng, Nor K. Noordin e Borhanuddin Mohd. Ali. A Review of 6LoWPAN Routing Protocols
[IEEE Communication Magazine, 1997]	Brian P. Crow, Indra Widjaja, Jeong Geun Kim, Prescott T.. Sakai IEEE 802.11 Wireless Local Area Network
[IETF, 1998]	S. Deering, Cisco, R. Hinden, Nokia. Internet Procolo, Version 6 (IPv6) Specification
[IETF, 2007]	G. Montenegro, Microsoft Corporation, N. Kushalnagar, Inter Corp, J. Hui, D. Culler e Arch Rock Corp. Transmission of IPv6 Packets over IEEE 802.15.4 Networks draft-ietf-6lowpan-format-13
[José A. Gutierrez, 2005]	José A. Gutierrrez. IEEE Std. 802.15.4 – Enabling Pervasive Wireless Sensor Networks
[José Pereira, 2013]	José Pereira. A camada Física do Padrão IEEE 802.15.4
[Jose P., Ignacio A., Daniel G., 2010]	Jose Pique, Ignacio Almazan, Daniel Garcia. BLUETOOTH
[Luís Antunes, 2012]	Luís Antunes. Identificação de pessoas numa portaria virtual
[Marco Naeve, 2004]	Marco Naeve. IEEE 802.15.4 MAC Overview
[Matthias K., Simon. D, Adam. D, 2011]	Matthias Kovatsch, Simon Duquennoy e Adam Dunkels. A Low-Power CoAP for Contiki
[Mirko Rossini, 2011]	Mirko Rossini. Design e implementazione di un proxy HTTP/CoAP
[RIPE NCC, 2011]	RIPE NCC e ICANN. IPv6 Address Types
[Tomislav D., Srdan K., Nenad G., 2011]	Tomislav Dimcic, Srdan Krc, Nenad Gligoric. CoAP implementation in M2M Environmental Monotoring System
[W. Colitti, K. Steenhaut, N. De Caro, Bogdan Buta and Virgil Dobrota, 2011]	W. Colitti, K. Steenhaut, N. De Caro, Bogdan Buta and Virgil Dobrota. REST Enabled Wireless Sensor Networks for Seamless Integration with Web Applications
[Zach Shelby, 2013]	Zach Shelby. CoAP: The Internet of Things Protocol
[Ian Poole, 2014]	Ian Poole. IEEE 802.15.4 Networking Topologies, http://www.radio-electronics.com/info/wireless/ieee-802-15-4/mesh-networking-topology-topologies.php [último acesso: Out 2014]
[Kaushi Das, 2008]	Kaushi Das. IPv6 Header Deconstructed, http://ipv6.com/articles/general/IPv6-Header.htm [último acesso: Out 2014]
[Kevin Ashton, 2009]	Kevin Ashton. That 'Internet of Things' Thing, http://www.rfidjournal.com/articles/view?4986 [último acesso: Out 2014]
[Microsoft, 2005]	Microsoft. The TCP/IP model, http://technet.microsoft.com/en-us/library/cc786900(v=ws.10).aspx [último acesso: Out 2014]
[TinyOS, 2013]	TinyOS. TinyOS Documentation WIki, http://tinyos.stanford.edu/tinyos-wiki/index.php/TinyOS_Documentation_Wiki [último acesso: Out 2014]