Instituto Superior de Engenharia do Porto

Aplicação de Internet of Things em casas inteligentes - Serviço Aplicacional

Alan Tomás Lima

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 João Paulo Jorge Pereira

Vogais:

Doutor Paulo Manuel Baltarejo de Sousa

Doutor Nuno Alexandre Magalhães Pereira

Porto, Outubro 2014

Aos meus pais, irmã e namorada

Resumo

A *Internet of Things* tem vindo a ganhar um maior destaque no mundo atual devido ao aumento de dispositivos inteligentes. A visão desta passa pela interconexão de dispositivos, estando estes a recolher e processar informação de uma forma automática. Dispositivos pessoais, tais como televisões e telemóveis, têm vindo a ganhar uma tendência em permitir uma interação com os mais diversos dispositivos. Aplicações de controlo de casas ou de controlo de outros ambientes a partir destes dispositivos são cada vez mais procuradas. Este tipo de aplicações não são muito inovadoras, pois já existem aplicações desse género no mercado mas, até ao momento, nenhuma dessas implementações foram padronizadas. O objetivo é permitir o acesso a qualquer tipo de dispositivo de uma forma simples, tal como abrir uma página web.

Neste projeto são estudadas as tecnologias mais promissoras a tornarem-se padronizadas, possibilitando servir como meio de comunicação entre estes dispositivos[1]. Assim, recorrendo a essas tecnologias, o projeto tem como objetivo a realização de uma aplicação que permita a descoberta e gestão de dispositivos numa casa inteligente a partir de uma interface web intuitiva. As tecnologias maioritariamente abordadas são o protocolo de troca de mensagens CoAP juntamente com a troca de mensagens entre uma rede IPv4 e uma rede IPv6.

Palavras-chave: Internet of Things, CoAP, REST, Wireless Sensor Network

[1] Parte desta dissertação foi realizada em parceria com Vasco Pereira, encarregue dos serviços de rede (Aplicação de Internet of Things em casa inteligentes – Serviços de Rede)

Abstract

The Internet of Things (IoT) has gained greater prominence in today's world due to the increase of smart devices. The sight of this, involves the interconnection of devices collecting and processing information in an automated manner. Personal devices, such as televisions and mobile phones, have been gaining a tendency to allow interaction with various devices. Applications, such as the control of a home or other environments from these devices, tend to be increasingly in demand. Such applications aren't that innovative since there are solutions to this but, so far, none of these implementations were standardized. The goal is to allow access to any device, also including devices with limited capabilities in a simple way, such as opening a web page.

In this project are studied the most promising technologies to became standardized, allowing to act as a mean of communication between these devices. Therefore, using these technologies, the project aims at the creation of an application that enables the discovery and management of devices in a smart home from an intuitive interface. The technologies that are mainly addressed are the CoAP protocol for exchanging messages together with an IPv6 and IPv4 network.

Keywords: Internet of Things, CoAP, REST, Wireless Sensor Network

Agradecimentos

Gostaria de agradecer ao meu orientador do projeto, Dr. Nuno Alexandre Magalhães Pereira e todos os colaboradores do CISTER pelo excelente ambiente proporcionado e pela disponibilidade em ajudar nos mais diversos problemas e dificuldades que foram surgindo ao longo do tempo.

Um especial agradecimento aos meus pais e irmã pela minha educação, fazendo de mim a pessoa que sou hoje e por todo o incentivo e apoio dado nos bons e maus momentos da minha vida, assim como o carinho dado, que foi fundamental para alcançar os meus objetivos.

Ao apoio imprescindível dado pela minha namorada ao longo do meu percurso académico que nos momentos mais difíceis sempre me incentivou e acreditou em mim, dando-me coragem, assim como o afeto demonstrado em todos os momentos.

Um grande obrigado ao meu colega de projeto e amigo Vasco Rafael Figueiredo Pereira pelo apoio e incentivo nestes meses de projeto.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

6LoWPAN *IPv6 over Low power Wireless Personal Area Network*

ACK *Acknowledgement*

API *Application Programming Interface*

ARPANet *Advanced Research Projects Agency Network*

CON *Confirmable*

CPU *Central Processing Unit*

CoAP *Constrained Application Protocol*

CoRE - *Constrained Resource Environments*

DARPA *Defense Advanced Research Projects Agency*

DTLS *Datagram Transport Layer Security*

HTML *HyperText Markup Language*

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*

LLN *Lossy Networks*

MAC *Media Access Control*

MCU *Micro Controller Unit*

MIME *Multi-Purpose Internet Mail Extension*

MIT *Massachusetts Institute of Technology*

MTU *Maximum Transmission Unit*

MVC *Model–View–Controller*

NON *Non Confirmable*

NPM *Node Package Manager*

OSI *Open Systems Interconnection*

RAM *Random Access Memory*

REST *Representational State Transfer*

RFID *Radio-Frequency IDentification*

ROM *Read Only Memory*

RPL *Routing Protocol for Low power and Lossy Networks*

RREQ *Broadcast Routing Request Packets*

RST *Reset*

SSL *Secure Sockets Layer*

TCP/IP *Transmission Control Protocol/Internet Protocol*

TLS *Transport Layer Security*

UDP *User Datagram Protocol*

URI *Uniform Resource Identifiers*

USB *Universal Serial Bus*

WLAN *Wireless Local Area Network*

WPAN *Wireless Personal Area Network*

WSN *Wireless Sensor Network*

XML *eXtensible Markup Language*

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 as maiorias 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 permitam a criação de aplicações para a gestão de um ambiente, mais precisamente de uma casa. Espera-se, assim, que a solução obtida tire partido dos mais recentes protocolos de aplicação e que consiga ser aplicada em qualquer situação real de forma a melhor otimizar a solução. As principais contribuições deste trabalho são:

Estudo e desenho de uma aplicação que tire partido das tecnologias inerentes à *Internet of Things*.

Estudo das várias tecnologias pertencentes ao tema *Internet of Things* em contexto real, para a implementação de uma aplicação que permite efetuar uma gestão de uma casa inteligente.

Identificação dos diversos componentes tecnológicos associados ao tema.

Identificação e descrição detalhada dos componentes necessários para a criação de uma aplicação que tire partido de uma rede de sensores que disponibiliza serviços e interfaces baseados em padrões desenvolvidos para a IoT.

Estudo e implementação de uma aplicação que comunica com uma rede de sensores limitada e permita a sua gestão.

Implementação de todos os componentes necessários para uma aplicação interagir com uma rede de sensores com capacidades limitadas tendo sempre em vista a utilização de protocolos *standard* ou das mais recentes propostas de *standard* das tecnologias da IoT.

1.3 Métodos e tecnologias utilizadas

Parte deste projeto foi realizado em conjunto com Vasco Pereira, encarregue de criar uma rede de dispositivos com capacidades limitadas. 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.

Numa fase inicial do projeto foi feito um estudo das tecnologias inerentes o tema *Internet of Things*, identificando os componentes tecnológicos que permitissem uma melhor decisão na escolha das tecnologias para o desenvolvimento do projeto. Foram elaboradas várias experiencias que permitiram identificar os componentes da aplicação a implementar.

Foi implementada uma API REST que permitisse a troca de informação com os dispositivos que era um protótipo para a implementação da aplicação final servindo para testar as tecnologias escolhidas.

Para o projeto final foi implementada uma aplicação que permite efetuar uma gestão de uma casa inteligente a partir de um *browser* ou um dispositivo móvel com acesso à internet. A aplicação foi criada em Node.js e comunica com uma rede de sensores de capacidades limitadas através do protocolo CoAP (*Constrained Application Protocol*).

Para tal, foi preparada uma rede de sensores limitados que permitissem a comunicação via CoAP e foram estudadas e implementadas as características necessárias para que a aplicação pudesse tirar partido desse protocolo. Na Figura 1Figura 1está representada a arquitetura simplificada do sistema implementado.

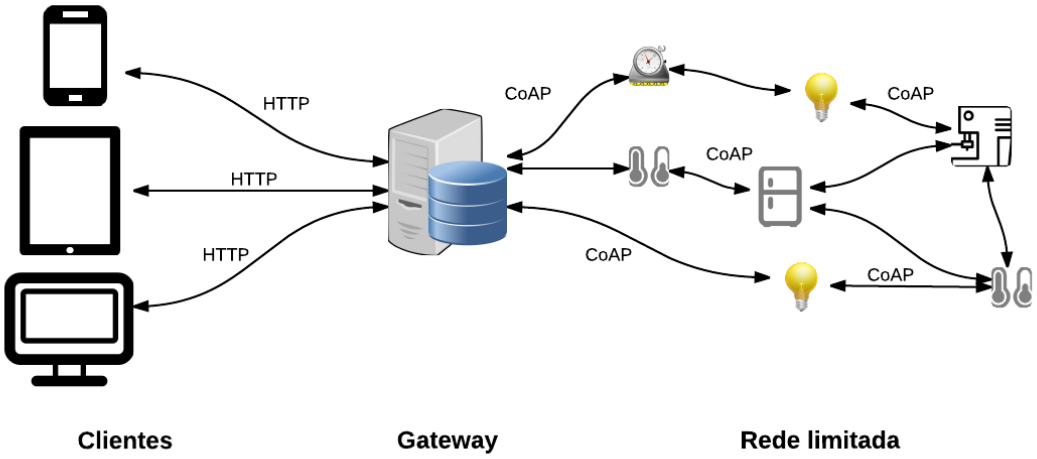

Figura 1 - Arquitetura simplificada da aplicação

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;
- No segundo capítulo, Estado da Arte, pretende-se mostrar os conceitos tecnológicos presentes atualmente no tema, possibilitando a comparação entre os mesmos;
- No terceiro capítulo, Tecnologias Utilizadas, é feita uma descrição mais detalhada das tecnologias utilizadas na implementação;

- No quarto capítulo, Desenvolvimento, é demonstrada a arquitetura e os detalhes da solução implementada;

- No quinto capítulo, Demonstração, onde é apresentada a aplicação final e demonstrada as interações possíveis;

- As conclusões deste trabalho são mostradas no quinto e último capítulo, Conclusões. Neste capítulo são descritas as principais limitações deste trabalho e apresentado trabalho futuro a realizar.

2 Levantamento do Estado de Arte

Este capítulo teve como objetivo efetuar um levantamento das tecnologias utilizadas inerentes ao tema e permitiu a definição da arquitetura da aplicação a implementada. É feita uma descrição breve das diversas tecnologias encontradas, no capítulo seguinte é efetuada uma descrição mais pormenorizada das tecnologias utilizadas na implantação do projeto.

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 que são relevantes para a nossa aplicação.

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 chaves para a ligação entre dispositivos e uma rede próxima, uma vez que existem inúmeros dispositivos a serem conectados entre si 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 outros), 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 utilizada, 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 transferência do pacote.

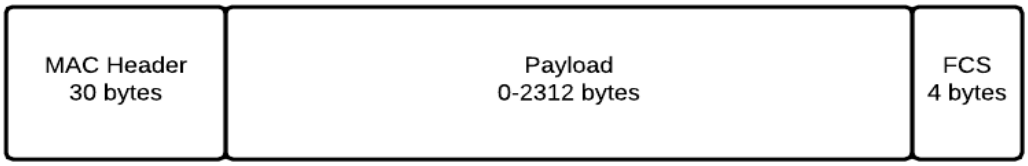

Figura 2 – Formato de um pacote da rede IEEE 802.11

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.

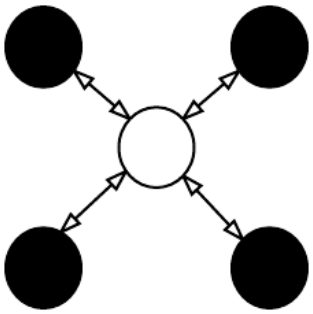

Figura 3 – Topologia Star

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].

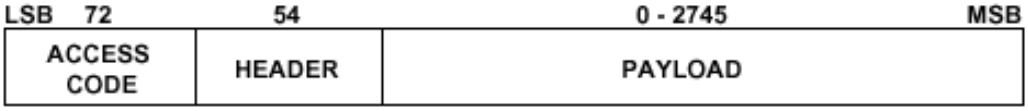

Figura 4 – Formato de um pacote da rede IEEE 802.15.1

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].

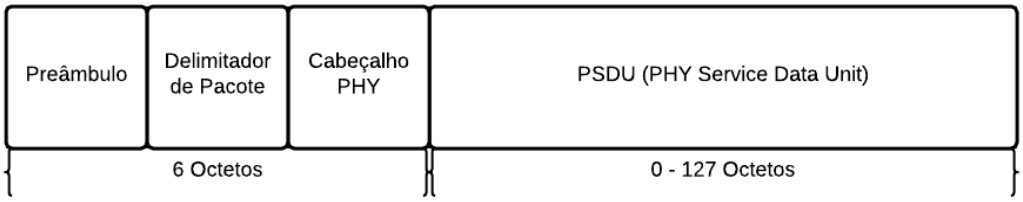

Figura 5 - Formato de um pacote da rede IEEE 802.15.4

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 esses protocolos.

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 128 *bits*, 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

Constrained Application Protocol (CoAP) é uma camada de aplicação desenvolvida pelo CoRE (Constrained Resource Environments) do Internet Engineering Task Force (IETF) que permite utilizar *Web Services* em dispositivos com menos capacidades, através da utilização do protocolo HTTP.

CoAP implementa o estilo arquitetural utilizado pelo REST, sendo mapeado transparentemente para HTTP. O seu cabeçalho é de 4 *bytes* e as mensagens têm 127 *bytes* para serem transmitidas via IEEE 802.15.4. As mensagens podem ser de dois tipos distintos: confirmable ou non confirmable (CON e NON). Ao utilizar CoAP são fornecidos quatro métodos, GET, POST, PUT e DELETE.

CoAP executa através de UDP e portanto a tecnologia SSL/TLS não está disponível nesta arquitetura mas, o DTLS (Datagram Transport Layer Security). É relevante o facto de o CoAP possuir a habilidade de observar recursos. Definindo uma *flag* no método GET do CoAP, o servidor pode continuar a responder após o documento inicial ser transmitido. Isto é vantajoso para verificar certas mudanças num recurso a ser observado.

Uma das vantagens na utilização do CoAP é a sua capacidade para descoberta de recursos. Os servidores fornecem uma listagem completa dos seus recursos e isto permite aos clientes saberem quais os recursos que podem utilizar e o seu tipo.

Na secção seguinte é feita uma descrição mais detalhada deste protocolo porque é a principal tecnologia utilizada no desenvolvimento da aplicação, pois especifica a comunicação com dispositivos de capacidade limitada.

3 Tecnologias utilizadas

A comunidade da IoT prevê um mundo onde milhões de objetos inteligentes estão ligados à *Web* permitindo a criação de inúmeras aplicações relacionadas com os mais variados ramos, tais como: saúde; monitorização de meios; cidades inteligentes; entre outros. Para isto ser possível é necessário tornar esses objetos acessíveis na *Web*. Esta tese focar-se-á no ramo das casas inteligentes mas com vista à criação de uma aplicação genérica que permita ser adaptada a outros ramos de uma forma simples.

Para que o acesso a estes objetos inteligentes seja possível através de *web*, por exemplo, a partir de um *browser*, um dos requisitos chave é a utilização do protocolo IP. Devido ao grande número de objetos inteligentes, o uso do IPv6 é fundamental e, com isto, o IETF desenvolveu o 6LowPAN. Este *standard* permite a utilização de IPv6 em redes de baixa capacidade, também conhecidas por Low-power and Lossy Networks (LLNs), que têm como base o protocolo IEEE 802.15.4.

O 6LowPAN permitiu um grande avanço na ligação entre as WSN e a internet mas apenas descreve a camada de rede, o que não é suficiente para responder aos requisitos de uma aplicação IoT.

A utilização do protocolo IP numa WSN permite a implementação de uma arquitetura baseada em *web services standard*, possibilitando assim que os dispositivos sejam acedidos através da *web*. A arquitetura usada para permitir esta ligação é o REST, a qual permite às aplicações acederem às redes através de uma interface bem definida. A utilização de *web services* nestes

tipos de rede não é simples devido às diferenças entre as aplicações comuns na internet e das aplicações de IoT.

Os *web services* de uma aplicação IoT estão em dispositivos que dependem de baterias que na maioria do tempo encontram-se suspensos para otimizar o tempo de vida da bateria e apenas acordam quando existe tráfego para ser enviado. Daí, estas aplicações necessitarem de uma comunicação assíncrona e de um serviço de *multicast* enquanto as aplicações comuns na *web* são síncronas e utilizam *unicast*. Este problema levou à necessidade do desenvolvimento de um *standard* que era baseado nos *standards* encontrados hoje na *web* mas que tenham em contas as limitações das redes com dispositivos de capacidades limitadas. Com isto o IETF tem vindo a desenvolver o protocolo CoAP que visa a responder a essas necessidades.

3.1 CoAP

O CoAP descreve um estilo de 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), o qual fornece a mesma segurança que o TLS. 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.

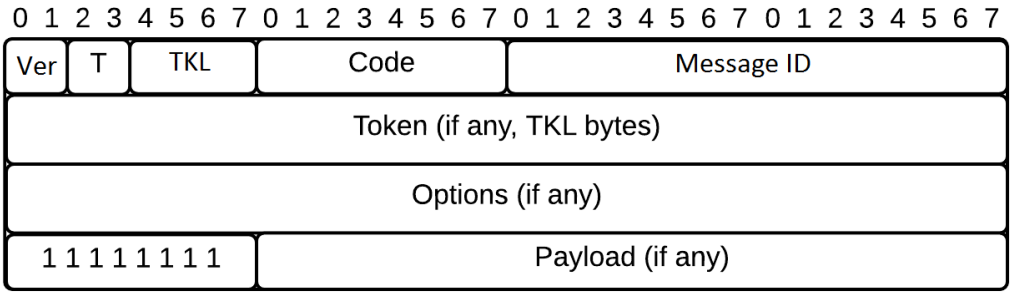

Figura 6 – Composição de um pacote CoAP

3.1.1 Formato da mensagem

Para ser possível a compreensão de como o CoAP funciona em redes de dispositivos com recursos limitados será apresentada a estrutura das mensagens trocadas:

Na Figura 6 podemos ver que os pacotes das mensagens são compostos por um *header* de 4 *bytes* seguidos por conjuntos de opções [Tomislav D., Srdan K., Nenad G., 2011]:

- 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.

3.1.2 Tipo de mensagem

Como falado anteriormente, no CoAP existem quatro tipos de mensagens que contêm um ID de mensagem para detetar duplicados e para tornar as mensagens 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 7

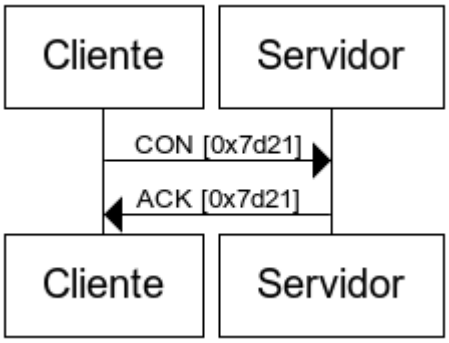

Figura 7 - Transmissões CON no CoAP

Non Confirmable (NON) – é uma mensagem que não precisa de ser confirmada pelo servidor. O cliente não tem como saber se o pedido chegou 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 levaram a este tipo de mensagem serem enviadas 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 ID 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 devido à utilização da opção *token*.

3.1.3 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 - dentifica 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 valido sem ter de enviar o seu valor novamente.

Location-Path, Location-Query

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

If-Match, If-None-Match

O *IF-Match* é usado para a 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.

3.1.4 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 apresentadas algumas parâmetros dos recursos que o *Core Link Format* disponibiliza:

- **Title:** este parâmetro fornece uma descrição legível do recurso;
- **Type:** contém o tipo do recurso e apenas é permitido um tipo por recurso;
- **Resource Type (rt):** contém 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.

3.1.5 Observe

O CoAP disponibiliza um mecanismo de *publish/subscribe* no qual o cliente indica que pretende receber as atualizações do 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.

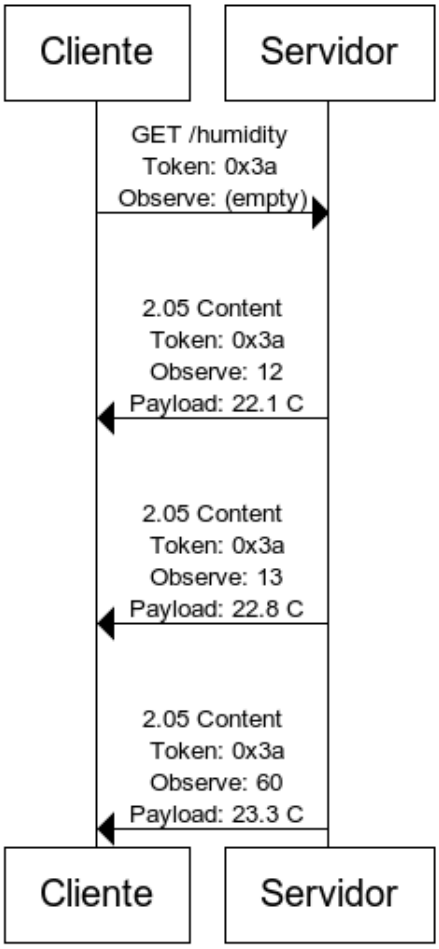

Figura 8 – Transmissão de pacotes de um *observe*

Como podemos ver na Figura 8, a correspondência entre o pedido e as notificações enviadas pelo servidor é feita recorrendo à opção *Token*. Se o pedido para observação for do tipo *confirmable*, o servidor tem que enviar respostas do tipo *confirmable* e o cliente deve enviar um ACK. Assim permite saber se o cliente ainda está interessado nas atualizações do recurso.

O valor *observe* numa resposta serve para indicar a ordem das notificações, assim, o cliente analisa esse valor para saber se este foi o valor mais recente ou se foi um valor que chegou com algum atraso.

Mapeamento HTTP-CoAP

Para ser possível a integração de dispositivos com capacidades limitadas com a *web* é crucial a capacidade destes interagirem com o HTTP. Com isto, o CoAP foi desenvolvido para que os

objetos que contêm CoAP possam comunicar para além da rede a que pertencem e serem integrados facilmente com outros protocolos que partilham a mesma estrutura do HTTP.

Devido ao facto HTTP e ao CoAP terem sido desenhados com os mesmos princípios e semânticas é possível a criação de aplicações RESTful sobre os dois protocolos, independentemente de qual dos dois está responsável por transmitir a informação. Assim, o processo de conversão das mensagens entre os dois protocolos pode ser feito de uma forma leve e simples, permitindo que a informação passe de uma rede limitada para outra rede diferente e vice-versa.

Para ser possível um mapeamento transparente e discreto entre os protocolos HTTP e CoAP é necessário um nó intermédio que perceba a sintaxe e a semântica dos dois protocolos. Este tipo de nó intermédio é um *proxy* que faz a tradução entre os protocolos e é conhecido como *cross proxy*. É composto por um cliente e um servidor de cada protocolo que, juntamente com umas funções de tradução, permitem que o fluxo de comunicação seja feito nos dois sentidos, ou seja, de HTTP para CoAP e CoAP para HTTP. Com isto, as redes limitadas (que comunicam com o protocolo CoAP) têm a capacidade de comunicar com outras redes.

3.1.6 Mapeamento de URI entre protocolos

Tanto o CoAP como o HTTP definem a estrutura dos seus URI's. Assim, um nó HTTP pode não suportar nativamente a estrutura do CoAP e passa a ser necessário a criação de um mapeamento de URI's entre estes protocolos. Deste modo, um recurso CoAP pode ser acedido a partir de um *browser* através de um URI HTTP fornecido ao *cross proxy*, mesmo sem o *browser* ter conhecimento do protocolo CoAP.

Quando um URI HTTP chega ao *proxy* os componentes da sua *path* podem ser diferentes dos componentes da *path* de um URI CoAP. Para que o mapeamento de URI's possa ser mais fácil e simples é recomendada a utilização de técnicas de mapeamento consistentes. De seguida, são mostradas algumas técnicas de mapeamento:

- **Utilização de URI's homogéneos -** um URI homogéneo entre protocolos consiste num endereço onde o mesmo recurso é identificado de forma única com os mesmos componentes tanto num protocolo como noutro. Por exemplo, para um URI Coap

coap://servidor.pt/resource/light, o seu endereço correspondente em HTTP é o seguinte: http://servidor.pt/resource/light.

- **Utilização de URI's embutidos -** um URI entre protocolos é dito embutido quando o URI do recurso coap está explicitamente no URI que chega ao *cross proxy*. Por exemplo, http://sevidor.pt/coap/servidorcoap.pt/resource/light onde o URI coap, coap://servidorcoap.pt/resource/light, está explicitamente no URI.

3.1.7 Controlo do número de acessos

No *gateway* entre a rede de internet e a rede limitada o *cross proxy* tem um papel importante de controlar o número de acessos à rede limitada. Na verdade, o número de acessos a uma rede limitada deve ser o mais pequeno possível.

Quando se ultrapassa um número pré-definido de pedidos efetuados ao *cross proxy*, este pode decidir não permitir a chegada de mais pedidos HTTP. Quando esse número é atingido, este pode rejeitá-los, usando o estado *503 Service Unavailable* que indica que o recurso pretendido está temporariamente inacessível. Quando um cliente HTTP recebe este tipo de resposta irá tentar, mais tarde, aceder a esse recurso.

Quando um *proxy* é um nó intermédio entre duas redes, é comum que este guarde as mensagens recebidas em *cache*. Assim, permite que o número de pedidos efetuado à rede limitada seja reduzido, porque se na *cache* do *proxy* ainda existir um valor atualizado do recurso, não é necessário efetuar o pedido desse recurso à rede limitada.

Outra opção, para reduzir o número de acessos à rede limitada, é estabelecer uma relação de *Observe* com os recursos mais utilizados. Ao usar o *Observe* o servidor tem conhecimento de onde um recurso está localizado e envia sempre uma representação atualizada do recurso para o *proxy* assim que o seu valor seja alterado. Deste modo, se o recurso que está na *cache* do *proxy* é uma representação atualizada, não é necessária a validação.

3.1.8 Mapeamento avançado

Para o *cross proxy* tirar o melhor partido das codificações feitas entre redes, este deve poder efetuar mais do que o simples mapeamento entre o HTTP e o CoAP. De seguida, serão apresentadas três funcionalidades de mapeamento que o *proxy* deve suportar:

Mapeamento entre IPv4 e IPv6 combinado com HTTP e CoAP respetivamente:

Como o IPv4 ainda é, nos dias de hoje, a versão do protocolo IP dominante na internet e como os dispositivos em redes limitadas utilizam o IPv6, devido ao elevado número de dispositivos previstos no futuro, o *gateway* necessita de efetuar um mapeamento entre IPv4 e IPv6 para permitir que um *host* IPv4 possa aceder aos recursos oferecidos pelos dispositivos que implementam o 6LowPAN.

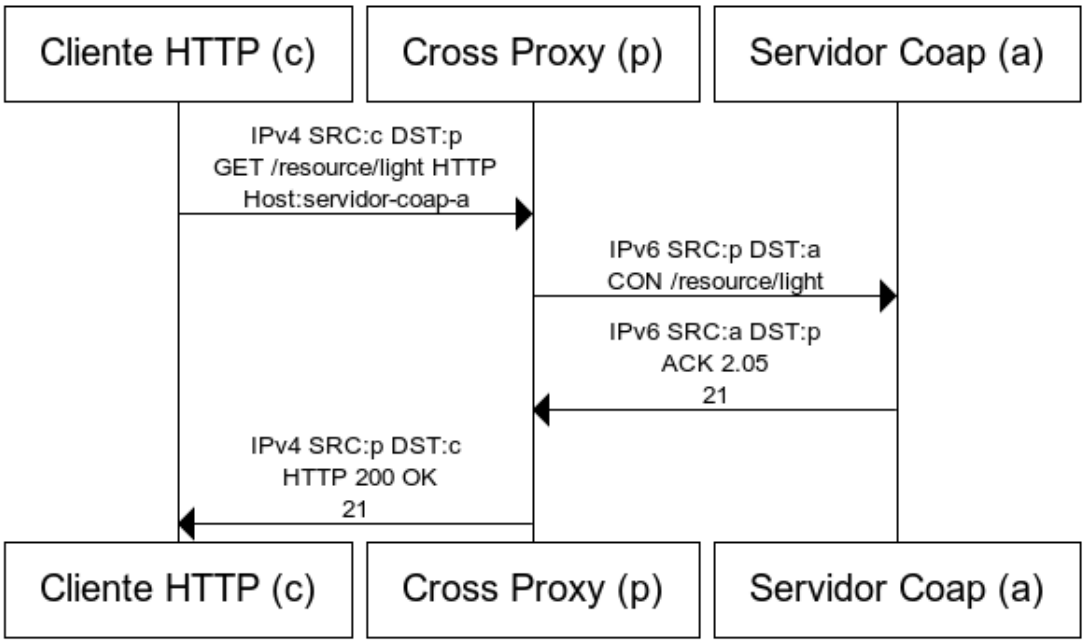

Figura 9 – Mapeamento de HTTP/IPv4 para CoAP/IPv6

A Figura 9 é uma demonstração gráfica de como o mapeamento entre HTTP/IPv4 e CoAP/IPv6 é efetuado. O cliente HTTP faz um pedido ao *cross proxy* de um recurso CoAP indicando o nome do servidor. O *cross proxy* recebe esse pedido através de IPv4 e, consoante o nome do servidor a que vai efetuar um pedido, faz o seu mapeamento buscando o seu endereço IPv6. Assim, envia um pedido CoAP ao servidor e aguarda pela resposta. Quando o *cross proxy* recebe a resposta, envia o *payload* da mensagem de volta para o cliente HTTP em IPv4.

Mapeamento de pedidos HTTP *unicast* para pedidos CoAP *multicast*:

O HTTP não permite o envio de pedidos *multicast,* por isso, o *cross proxy* pode receber um pedido HTTP *unicast* e enviar mensagens *multicast* CoAP para obter valores de vários nós da rede limitada. Depois, o *cross proxy* deve agregar a resposta das múltiplas mensagens CoAP recebidas e enviar tudo numa única resposta para o cliente HTTP. A Figura 10 ilustra este funcionamento.

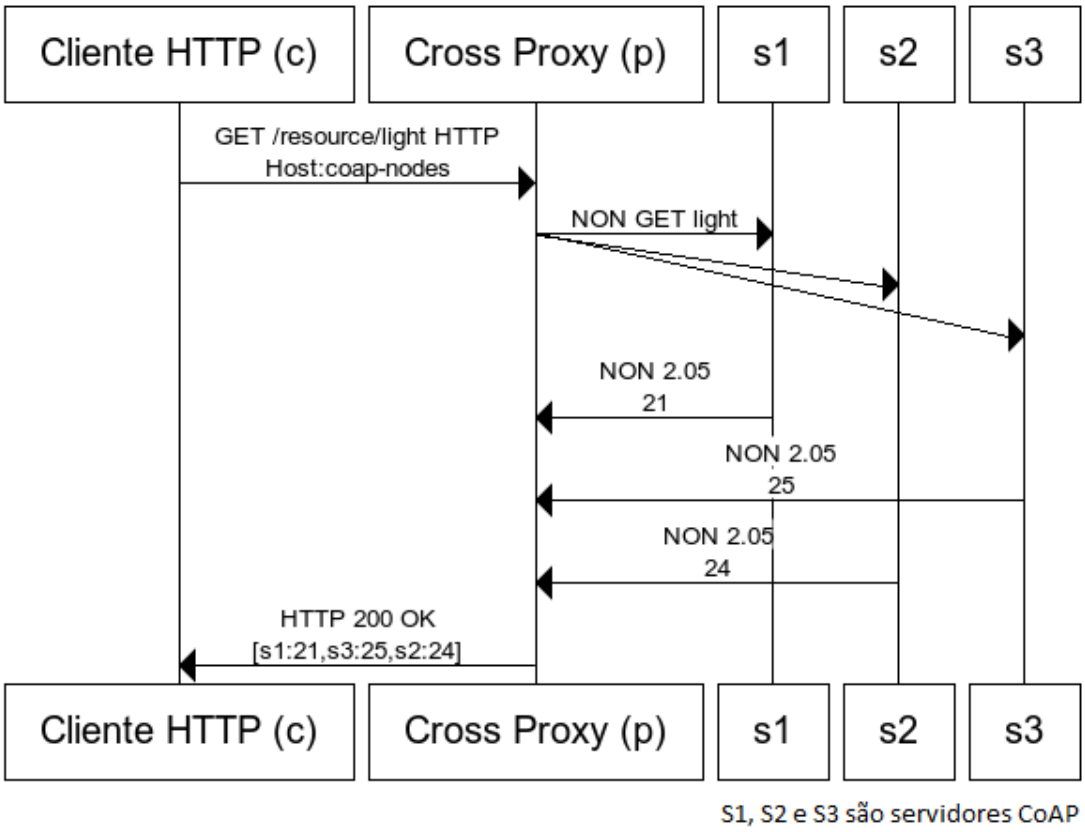

Figura 10 – Mapeamento de uma mensagem HTTP *unicast* para CoAP *multicast*

3.2 Node.js/node-coap

Segundo a definição da página do node.js [node.js,2014], este consiste numa plataforma criada sobre o interpretador *V8 JavaScript* do *Chrome* para a criação de aplicações de rede rápidas e escalonáveis de uma forma simples. O node.js utiliza um modelo de I/O direcionado a eventos não bloqueantes, tornando, assim, as aplicações mais leves e com uma grande eficiência. Este processo é ideal para aplicações em tempo real com uma troca intensiva de dados através de dispositivos distribuídos.

O interpretador *V8* JavaScript é o interpretador usado pela Google no *browser Chrome*. Este interpretador foi feito em C++ , faz a interpretação do JavaScript e está presente no Chrome, podendo ser descarregado e integrado numa aplicação, que é o caso do node.js, que utiliza este interpretador para a criação de servidores em JavaScript.

Comparando com as técnicas convencionais dos servidores *web*, que a cada ligação efetuada ao servidor é criada uma nova *thread* e, com isto, é utilizado um bloco de memória RAM para cada ligação. Eventualmente, se o número de ligações for aumentando, o servidor pode ficar sem memória suficiente para receber mais ligações. O node.js funciona com uma única *thread,* utilizando operações I/O não bloqueantes, o que permite um elevado número de ligações simultâneas. Por exemplo, assumindo que cada *thread* ocupa 2MB de memória de RAM e que o servidor tem 8GB, teoricamente o servidor apenas permite 4000 ligações simultâneas. [T. Capan, 2013] Além disso, existe um elevado custo na troca de *threads* no CPU, o que diminui o desempenho do servidor. Ao evitar este senário, o node.js suporta um milhão de ligações simultâneas.

Devido ao fato de o node.js apenas partilhar uma *thread* entre todos os clientes, isto pode revelar-se um problema para certos tipos de aplicações. Por exemplo, aplicações de computação pesada, com um uso intensivo do CPU, levam a que um pedido afete os restantes clientes, pois a *thread* estará bloqueada até a computação terminar. As operações intensivas no CPU anulam todas as vantagens que o node.js tira partido com o modelo de I/O direcionado a eventos não bloqueantes.

O node.js é indicado para situações em que é esperado um grande volume de tráfego e o nível de processamento que é necessário efetuar pelo servidor, para responder aos clientes, seja baixo. Alguns exemplos em que o node.js é indicado são:

- Criação de uma API RESTful que consiste numa aplicação que recebe parâmetros, interpreta-os e envia a resposta para o cliente. As respostas enviadas para o cliente, normalmente, são pequenas quantidades de texto e não requerem um grande processamento por parte do servidor, porque na maior parte das vezes, este apenas vai buscar informação a uma base de dados.
- O node.js é utilizado para a criação de um *proxy,* porque suporta múltiplas ligações simultâneas e, devido ao fato de ser orientada a eventos não bloqueantes, é útil para fazer *proxing* de respostas com tempo de respostas diferentes e ir buscar informação a vários pontos diferentes, como sistemas distribuídos.

O node.js possui um gerenciador de pacotes chamado Node Package Manager (NPM). Este possui um repositório *online* onde se encontram publicados projetos de código aberto para o node.js. A instalação do node.js vem com um cliente que permite instalar módulos no node.js a partir da linha de comandos com controlo de dependências e versões. Uma lista completa de módulos pode ser encontrada no site do NPM[2].

O node.js possui um módulo de CoAP chamado node-coap, desenvolvido por Matteo Collina. Este módulo trata-se de uma biblioteca para clientes e servidores CoAP, modelado a partir do módulo HTTP. Com este módulo, é possível efetuar pedidos a servidores CoAP, assim como enviar dados para esses mesmos servidores [M. Collina, 2013].

[2] https://npmjs.org/

4 Desenvolvimento

4.1 Arquitetura

No capítulo anterior foram enumeradas as tecnologias escolhidas, tendo sempre em conta a utilização de *standards* ou de propostas que, no futuro, possam tornar-se *standards*. A Figura 11 mostra o grupo de dispositivos necessários para a implementação da aplicação e o modo como o fluxo das mensagens será efetuado. De seguida, será apresentada a arquitetura dos sistemas necessários para a implementação da aplicação e a arquitetura da mesma.

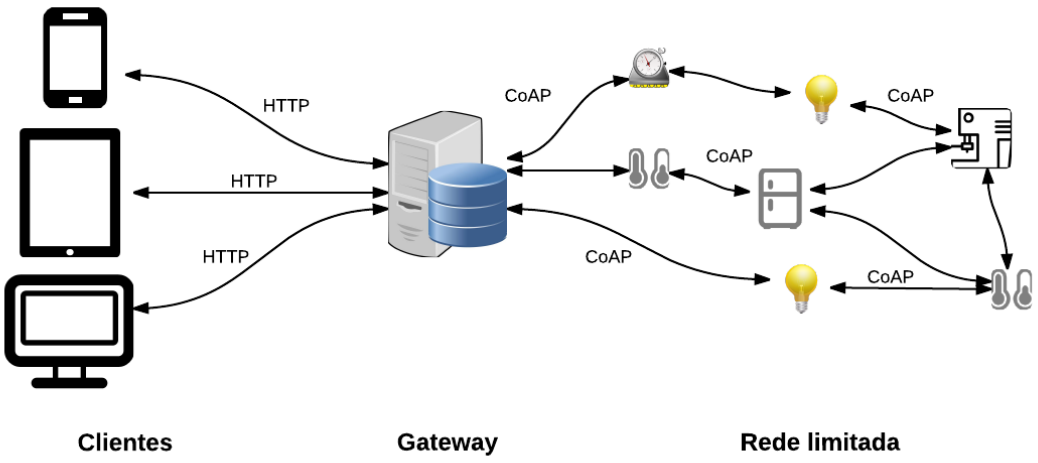

Figura 11 – Componentes presentes no sistema desenvolvido

4.1.1 Rede limitada

Para o desenvolvimento da aplicação foi necessário montar e configurar uma rede de sensores 6LowPAN que permitisse a comunicação através de CoAP, disponibilizando os seus recursos para serem utilizados na aplicação. Para tal, foram utilizados sensores XM1000 da advanticsys, iguais ao da Figura 12, que têm a capacidade de funcionar como servidores de CoAP.

Figura 12 - Plataforma Advanticsys XM1000

O XM1000 é um sensor que tem as características necessárias para a criação de uma aplicação CoAP. Algumas dessas características são as seguintes:

- Possui um Micro Controller Unit (MCU) de baixo consumo energético, o microcontrolador MSP430F2618 que tem um CPU de 16-*bit* com um *clock* de 16MHz, 8KB de RAM e 116KB de memória;
- Um transmissor compatível com o IEEE 802.15.4, o CC2420, que opera com na frequência de 2.4GHz com uma capacidade de transmissão de 250Kbps;
- Sensores incorporados (sensor de luz, temperatura e humidade) e leds;
- Uma interface USB para uma rápida configuração.

A IoT necessita de um protocolo de *routing* adaptado às características de rede que é constituída por muitos dispositivos de energia e capacidade reduzida. Este sistema de *routing* deve também ter em conta a forma como é efetuado o fluxo das mensagens, ponto a ponto, ponto para múltiplos pontos e múltiplos pontos para um ponto.

O *Routing Protocol for Low power and Lossy Networks* (RPL), é um protocolo de *routing* desenhado para redes IoT. O RPL especifica como efetuar o controlo do fluxo do tráfego em redes limitadas.

O Contiki é um sistema operativo *open source* criado para a IoT. Este sistema operativo foi desenhado para funcionar em dispositivos de capacidades limitadas e, por isso, depois de instalado nos dispositivos apenas utiliza em média dois *kilobytes* de RAM e quarenta *kilobytes* de ROM. O Contiki vem uma *stack* IPv6 que suporta a compressão de cabeçalhos 6LowPAN e disponibiliza uma implementação de CoAP.

O Contiki foi utilizado para configurar os servidores de CoAP no XM1000 e também para a configuração de um *RPL Border Router* numa placa *BeagleBone Black*, juntamente com um sensor TelosB para permitir receber o tráfego proveniente dos servidores CoAP, este *border router* é responsável por ligar a rede limitada à rede da aplicação. A Figura 13 ilustra a rede de sensores utilizada para o desenvolvimento da aplicação.

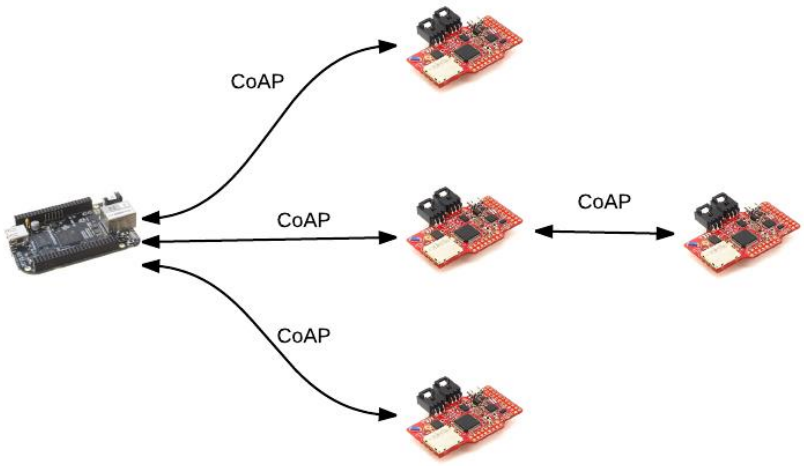

Figura 13 – Rede limitada

Recursos disponibilizados nos servidores CoAP

Os servidores de CoAP possuem recursos que descrevem as funcionalidades que este dispositivo oferece. Na criação da rede de sensores ouve um cuidado em definir recursos diferentes em cada servidor CoAP. Os recursos que um disposto possui passam pela sua informação, nome e localização, vários tipos de sensores e atuadores. Os servidores CoAP disponibilizam esses recursos de forma a ser possível interação com estes através de métodos bem definidos. Um cliente que queira efetuar operações com os recursos de um servidor CoAP, necessita de ter um conhecimento de todos os sensores que este disponibiliza.

Para tal, todos os servidores CoAP oferecem uma interface bem definida chamada *well-know* que faz a listagem e descrição dos recursos existentes.

```
</.well-known/core>;ct=40,
 </sensors/temperature>ors/temperature>;title="Temperature”; rt="Sensor",
 </info>;title="Info";rt="Info",
        </info/name>;title="Name";rt="Info",
        </info/location>;title="Location";rt="Info",
 </actuators/toggle>;title="Red LED";rt="Control"
```

Código 1 – Descrição dos recursos de um servidor CoAP

O Código 1 é um exemplo da descrição enviada por um servidor CoAP quando é feito um pedido ao interface *well-know*.

Este exemplo, o servidor possui cinco recursos. Na descrição de cada recurso é disponibilizado informações como o caminho do recurso /sensors/temperature, o tipo do recurso, rt=”Sensor” e num nome do recurso title=”Temperatura”.

4.1.2 Gateway

Na aplicação foi utilizado o padrão de arquitetura *Model–View–Controller* (MVC). Esta arquitetura divide a aplicação em componentes, o que permite facilitar o reaproveitamento de código, a manutenção e adição de recursos e manter o código organizado.

O MVC é um padrão de arquitetura de aplicações utilizado em bastantes *frameworks* que divide o código em três componentes com o objetivo de separar os dados e a lógica do negócio da interface com o utilizador. Os três componentes, como o nome indica, são os seguintes:

- Model – ou modelo, é a camada que contém a lógica e as regras do negócio, sendo responsável pelos dados da aplicação.
- View – ou visão, é responsável pela apresentação da informação do modelo. Podem ser utilizados *templates* para efetuar essa representação.
- Controller – ou controlo, é responsável por gerir todos os pedidos feitos à aplicação. A sua principal função consiste em ir buscar ou manipular os dados para satisfazer os

pedidos recebidos, ou seja, o controlador escolhe o modelo e a visão apropriada para cada pedido recebido.

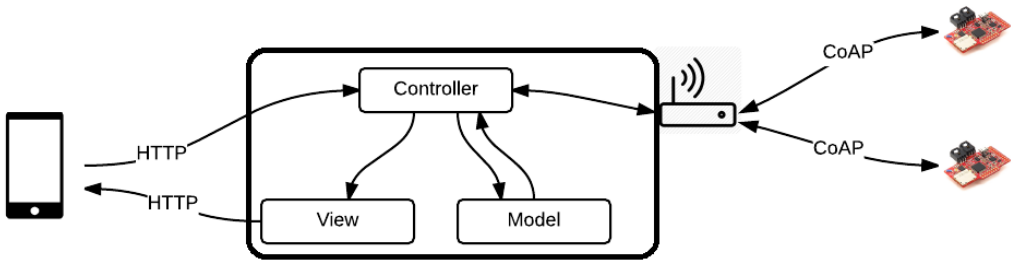

Figura 14 – Arquitetura da aplicação

A Figura 14 é a representação da arquitetura da aplicação e das interações efetuadas entre os diferentes componentes do sistema.

Controller

Os clientes enviam pedidos à aplicação através de uma API REST. É da responsabilidade do *controller* tratar esses pedidos e recolher a informação para ser apresentada como resposta ao cliente. Para tal, o *controller* tem que tomar decisões e efetuar operações que são as seguintes:

1. **Descoberta de dispositivos**

   O *controller* necessita de descobrir os dispositivos presentes na rede limitada para que possa ser possível aceder aos recursos dos mesmos. Este obtém essa informação a partir do RPL *border router.* Como dito anteriormente, *border router* é responsável pelo controlo do fluxo do tráfego da rede limitada. Devido a isto, este contém um registo de todos os nós da rede, ou seja, a lista de todos dispositivos conectados à rede. O *controller* efetua um pedido ao *border router* e recebe uma lista com os endereços dos dispositivos da rede limitada. O *controller* necessita de verificar se esses dispositivos já se encontram registados na base de dados e se se tratar de um novo dispositivo na rede este é responsável por efetuar um novo registo na base de dados para esse dispositivo. Assim, o *controller* necessita de ter conhecimento do *border router* para efetuar esse pedido. Com isto, a aplicação efetua um pedido ao *border router* e obtém os endereços dos dispositivos, criando um registo na base de dados para cada dispositivo para posteriormente ser preenchido com os recursos disponíveis nesses dispositivos.

2. Descoberta de recursos

Depois da descoberta dos dispositivos, a base de dados contém registos dos dispositivos da rede limitada mas é necessário preencher os restantes campos desses registos que correspondem aos recursos dos dispositivos. Todos os servidores CoAP contêm um recurso chamado *well-know* que retorna uma discrição de todos os recursos disponíveis no dispositivo, como explicado anteriormente. Com esta interface comum em todos os dispositivos é possível obter os seus recursos através de um pedido GET ao recurso well-know/core. Por exemplo, para um dispositivo com o endereço 2607:f0d0:1002:51::4 é efetuado o seguinte pedido GET coap:// 2607:f0d0:1002:51::4/ well-know/core.

```
</.well-known/core>;ct=40,
 </sensors/temperature>ors/temperature>;title="Temperature”; rt="Sensor",
 </info>;title="Info";rt="Info",
        </info/name>;title="Name";rt="Info",
        </info/location>;title="Location";rt="Info",
 </actuators/toggle>;title="Red LED";rt="Control"
```

Código 2 – Retorno do GET ao well-know/core

O Código 2 é um exemplo da resposta ao pedido efetuado ao well-know. Como dito na estrutura dos dados, os recursos existentes nos dispositivos foram divididos em três grupos. A atribuição de um grupo a cada recurso é feita consoante o valor rt (*resource type*). Para o grupo informação vão os recursos em que o valor do rt é Info, para o grupo sensores os que o rt está a Sensor e para os atuadores o rt está a Control [M. Ruta, F. Scioscia, A. Pinto, E. Di Sciascio, F. Gramegna, S. Ieva, G. Loseto, 2013].

3. Cross Proxy

Uma das principais características do *controller* consiste em funcionar como o *cross proxy* da aplicação. Para controlar o número de pedidos à rede limitada, quando o *controller* recebe um pedido, consulta a base de dados e verifica se os dados que estão lá são recentes, caso estes sejam recentes envia de imediato esses valores sem efetuar qualquer pedido à rede limitada. Se os dados da base de dados não forem recentes, este necessita de os atualizar, para isso, o *controller* faz o mapeamento do URI do pedido HTTP efetuado pelo cliente para CoAP e vai aos dispositivos buscar o novo valor, atualizando a base de dados e, posteriormente, envia os dados para o cliente.

4.1.3 Cliente

Para a interface da aplicação foi utilizada o Twitter Bootstrap que consiste numa *framework* HTLM, CSS e JS para o desenvolvimento de aplicações *web* responsivas. Aplicações responsivas consistem em aplicações que se adaptam as características do *browser* em que são abertas, mais propriamente à resolução do mesmo. Um site, dependendo da resolução do *browser* em que é aberto, adapta-se alterando o seu layout sem a necessidade de criação de diversos estilos. Com isto, a aplicação pode ser aberta num dispositivo móvel com um layout próprio de um site móvel.

Esta interface é responsável pela apresentação dos dados relativos à rede de sensores e por efetuar a chamada aos métodos da API REST referida no ponto anterior para a interação com os sensores.

4.2 Implementação

4.2.1 Definição da estrutura de dados dos sensores

Para o armazenamento e transmissão dos dados referentes aos sensores foi necessário definir uma estrutura. Os sensores disponibilizam a sua informação nos seguintes formatos: XML, JSON e texto simples. Devido ao servidor da aplicação ser implementada usando node.js, ou seja, em javascript o formato utilizado foi JSON.

O JSON (JavaScript Object Notation) é um formato de troca de troca de informação e a vantagem da sua utilização juntamente com o node.js é que o JSON utilizado na parte do servidor é igual ao JSON utilizado na parte do cliente não necessitando assim de uma conversão entre tipos de dados.

A base de dados utilizada pela aplicação foi implementada em MongoDB que consiste numa base de dados NoSQL (Not Only SQL), ou seja, a estrutura dos nados não é baseada em tabelas e nas relações entre elas. O MongoDB é baseado em documentos JSON com schemas dinámicos. Para a aplicação aceder à base de dados foi utilizada a biblioteca MongoSkin que permite aceder à base de dados em Node.js.

O Código 3 é um exemplo da estrutura de dados definida para ser utilizada na aplicação para a representação dos dados provenientes dos sensores.

```
sensor = {
 _id : "5447c94f52848fa03bb1e45c",
 address : "aaaa::212:7400:13b7:284c",
 info : {
        location : "sala",
        name : "Device 1"
 },
 sensors : {
        temperature : {
               value : "27",
               time : 1414079616347
        }
 },
 actuators : {
        toggle : "on"
 }
}
```

Código 3 – Estrutura de dados de um dispositivo

Na informação presente nos sensores foram identificados três grupos de informações relevantes, que permitem dividir toda a informação neles contidos, que são os seguintes:

- Informação: este grupo é responsável por guardar todos os dados relativos à identificação e localização do recurso;
- Sensores: este grupo contem todos os sensores disponíveis no dispositivo, como por, sensores de luz e temperatura;
- Atuadores: Consiste em todos os componentes que funcionam como atuadores onde é possível efetuar operações como ligar ou desligar o seu estado.

4.2.2 Estrutura/criação do projeto

Para a criação do projeto em node.js foi utilizado o módulo express-generator que cria a estrutura base para projetos que utilizam o módulo Express. O Express é uma framework que permite facilitar e aumentar a velocidade do desenvolvimento de aplicações web, uma vez que permite a criação de aplicações com base em estruturas predefinidas. Esta framework ajuda na organização do projeto de uma aplicação web pois segue uma arquitetura Model-View-Controller (MVC).

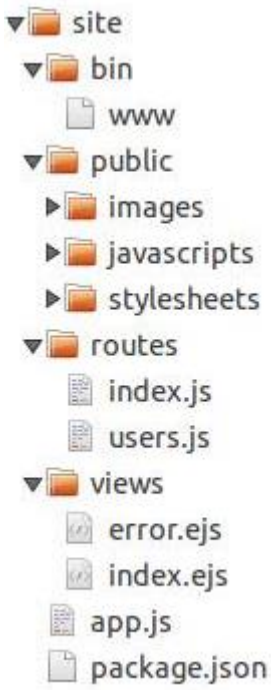

Figura 15 – Estrutura dos ficheiros da aplicação

A Figura 15 mostra a estrutura dos ficheiros da aplicação gerada executando o Código 4:

```
express --ejs site
```

Código 4 – Comando para a criação do projeto

Os pontos essenciais desta estrutura de ficheiros são:

- bin/www.js é usado para iniciar a aplicação executando o comando npm start;
- A pasta routes é onde estão todos ficheiros relativamente ao routing da aplicação;
- A pasta views contém as views da aplicação;
- O ficheiro package.json é o ficheiro que contém todas as dependecias da aplicação. Para instalar os módulos que a aplicação depende basta executar o comando `npm install` que cria uma pasta chamada node_modules e todas os módulos listados neste ficheiro serão instalados nessa pasta;
- O ficheiro app.js é o ficheiro que faz a configuração da aplicação.

4.2.3 Pedidos da aplicação aos dispositvos

A aplicação necessita de comunicar com os dispositivos da rede limitada para obter os seus valores e para alterar dados dos mesmos. Para esses pedidos foi utilizado o módulo node-coap que permite efetuar pedidos usando o protocolo CoAP.

Descoberta de recursos dos dispositivos

Quando é conhecido o endereço de um dispositivo é necessário descobrir os recursos que este possui. Em CoAP fazendo um GET à interface well-know é obtido uma descrição do dispositivo listando todos os seus recursos e características.

```
function getResources(ip,callback)
{
 var obj = {
        hostname:ip,
        port:5683,
        method: 'GET',
        confirmable: true,
        pathname:".well-known/core",
        options: {"Accept": "application/link-format"}
 var data="";
 var reqCoap = coap.request(obj);

 reqCoap.on('response',function(res){
        res.pipe(bl(function(err, data) {
               callback(data);
        });
 });
 reqCoap.end()
};
```

Código 5 – Função responsável pela descoberta dos recursos de um dispositivo

O Código 5 corresponde à função reponsável por efetuar o GET a essa interface. Esta função recebe o endereço do dispositivo e uma callback, onde a resposta desta função será tratada.

O método coap.request executa um pedido CoAP, recebe um objeto que contém todos os detalhes do pedido a exetutar e retorna uma OutgoingMessage. Uma OutgoingMessage permite esperar por uma um evento para executar uma função, neste caso o reqCoap.on espera pelo evento 'response' e executa uma função onde é feito um pipe da informação recebida e envia-a na chamada da callback.

Obter informação dos recursos dos dispositivos

A aplicação tem de aceder aos recursos dos dispositivose obter os seus valores, para tal, tem de fazer um pedido CoAP aos recursos do dispositivo. A função que é resposável por efetuar esses pedidos, Código 6, é semelhante à da função que descobre os recursos mas o pathname desta é a caminho relativo do recurso e o formato da resposta é JSON.

```
function getResourceData(ip,pathname,callback)
{
 var obj = {hostname:ip,
     port:5683,
     method: 'GET',
     confirmable: true,
     observe: false,
     pathname: pathname,
     options: {"Accept": "application/json"}
 };

 var data="";

 var reqCoap = coap.request(obj);

 reqCoap.on('response',function(res){
     if(res.code==2.05)
     {
         res.pipe(bl(function(err, data) {
             var json = JSON.parse(data);
             var err;
             callback(json);
         }));
     }else{

         callback(res.code);
     }
 });
 reqCoap.end();
};
```

Código 6 – Função que efetua GET a um recurso de um dispositivo

O pathname do recurso é passado como parâmetro. No evento ´response' é verificado se o pedido feito teve sucesso, se o código da resposta recebida é 2.05, é feito um parse da informação recebida antes de chamar a callback com a resposta.

Alterar informação dos recursos dos dispositivos

Quando a aplicação altera o valor de um recurso é enviado um POST ao recurso com um payload caso seja necessário. O *payload* é enviado na função reqCoap.end(). O Código 7 é a

função responsável por efetuar POST a um recurso de um dispositivo alterando os valores do mesmo.

```
function postResourceData(ip,pathname,payload,callback)
{
    var obj = {hostname:ip,
        port:5683,
        method: 'POST',
        confirmable: true,
        pathname:pathname,
        options: {"Accept": "application/json"}
    };

    var data="";
    var reqCoap = coap.request(obj);
    reqCoap.on('response',function(res){
        callback(res.code);
    });
    reqCoap.end(payload);
};
```

Código 7 - Função que efetua POST a um recurso de um dispositivo

4.2.4 API da aplicação

Nesta secção será apresentada a API utilizada para efetuar pedidos aos recursos dos dispositivos a partir da aplicação. Para ser possível responder aos pedidos através desta API é necessário efetuar consultas à base de dados e se esta não possuir os dados para responder a esse pedido ou se os que possuírem estiverem desatualizados a aplicação necessita de efetuar pedidos aos dispositivos. Na Tabela 3 estão enumerados os URI's que permitem interagir com os recursos dos dipositivos.

Tabela 3 – Interfaces da API

Nº	URI	Discrição do recurso	Método	
			GET	POST
1	/node	Lista de todos os dispositivos	X	
2	/node/<id>	Lista de recursos de um dispositivo	X	
3	/node/<id>/info	Informação de um dispositivo	X	
4	/node/<id>/info/name	Nome de um dispositivo	X	X
5	/node/<id>/info/location	Localização de um dispositivo	X	X
6	/node/<id>/sensor	Lista de sensores de um dispositivo	X	
7	/node/<id>/sensor/light	Sensor de luz de um dispositivo	X	

Nº	URI	Discrição do recurso	Método	
			GET	POST
8	/node/<id>/sensor/humidity	Sensor de humidade de um dispositivo	X	
9	/node/<id>/sensor/temperature	Sensor de temperatura de um	X	
10	/node/<id>/actuators /	Lista de atuadores do dispositivo	X	
11	/node/<id>/actuators/toggle	Interruptor de um dispositivo		X

Lista de todos os dispositivos

Para obter a lista de dispositivos é usada a interface número 1. Esta interface é responsável pela descoberta de dispositivos, para isso, efetua um pedido ao *border router* para obter a lista de todos os dispositivos ligados.

```
router.get('/', function(req, res)
{ var db = req.db;
  var devices=db.collection('devicelist');
  request({
        uri: "http://[aaaa::212:7400:e33:539]",
        method:"GET",
        timeout: 10000,
        followRedirect: true,
        maxRedirects: 10
      }, function(error, ips, body){
        devices.find().toArray(function(err, deviceList) {
              if (err) throw err;
              for (var i = ips.length - 1; i >= 0; i--) {
                    if(ips[i].exists(deviceList){
                          getResources(ips[i], function(result){
                                //Insert result in DB
                                ...
                          );
                    });
              }
        devices.find().toArray(function(err, result){
              if (err) throw err;
              res.render('nodes',{ devices: result });
        });
      }
  );
});
```

Código 8 – Interface para listagem e descoberta de dispositivos

O Código 8 demonstra o pedido efetuado ao *border router*. Na *callback* desse pedido é verificado se algum dos endereços não está na base de dados. Se existir algum endereço que não esteja na base de dados isto significa que é um novo dispositivo na rede e, para tal, é criado um novo registo na base de dados. De seguida, é chamada a função getResources para

ser feita a descoberta dos seus recursos e a sua inserção na base de dados. Essa descoberta é feita de uma forma assíncrona, ou seja, enquanto a aplicação está a efetuar a descoberta dos recursos o utilizador já está a visualizar a lista de sensores pois é chamada a *view* nodes.

Listagem da informação dos dispositivos

As interfaces 3, 4 e 5 são responsáveis por obter a informação dos dispositivos, ou seja, buscar a localização e o nome. Estas interfaces apenas efetuam pedidos à base se dados, devido ao facto destes recursos apenas poderem ser alterados na aplicação a base de dados contém sempre os dados atualizados desses recursos.

```
router.get('/:id/info/name', function(req, res)
{
 var db = req.db;
    var ip=req.params.id;
    var devices=db.collection('devicelist');

 devices.findOne({address:ip}, function(err, result) {
        if (err) throw err;
        res.send(result.info.name);
 });
});
```

Código 9 – Exemplo da listagem da informação dos dispositivos

O Código 9 é o método que devolve o nome de um dispositivo, este faz uma pesquisa à base de dados que retornando os dados do dispositivo com o mesmo endereço recebido, retornado apenas o nome. Os métodos para a consulta da localização e da informação (nome e localização) são idênticos alterando apenas o retorno da função.

Alterar informação dos dispositivos

Os recursos informação e localização são os recursos que o utilizador pode alterar na aplicação. Para tal, é necessário enviar um POST da informação aos dispositivos para serem alterados e em caso de sucesso nesta operação é registada essa alteração na base de dados.

```
router.post('/:id/info/name', function(req, res)
{
    var db = req.db;
    var ip=req.params.id;
    var name=req.param('name');
    var devices=db.collection('devicelist');
    postResourceData(ip, "info/name",name,function(result){
        if(result!=2.05)
        res.status(400).send("bad request");
        else{
```

```
        devices.update({address:ip},{$set:{"info.name":name}},function(err)
        {
               if (err) throw err;
                      devices.findOne({address:ip}, function(err, result) {
                             if (err) throw err;
                             res.send(result.info.name);
                      });
               });
        }
    });
});
```

Código 10 - Exemplo da alteração da informação dos dispositivos

No Código 10 é efetuado a alteração do nome de um dispositivo. É efetuado um pedido POST ao dispositivo chamando o método postResourceData e o *payload* é recebido através da variável req. Na *callback* desse método é verificado se o código da mensagem recebida é 2.05, ou seja, se o POST foi efetuado com sucesso e se essa condição se verificar é alterado o estado da variável na base de dados.

Consulta de valores dos sensores

As interfaces 7, 8 e nove permitem consultar os valores recolhidos pelos sensores. Os valores recebidos pelos sensores são guardados na base de dados e passado algum tempo os valores que estão presentes na base de dados podem não ser dados verdadeiros, ou seja, quando um sensor envia os dados para a aplicação é registado esse valor na base de dados e até esse valor ser atualizado o meio ambiente pode ter sofrido alterações e os dados recolhidos nos sensores podem ser diferentes dos valores da base de dados.

```
router.get('/:id/sensors/temperature', function(req, res)
{
 var db = req.db;
 var ip=req.params.id;
 var devices=db.collection('devicelist');
 devices.findOne({address:ip}, function(err, result) {
   if(result.sensors.temperature.value==””  ||
result.sensors.temperature.time+(10*1000) <Date.now())
   {
        //call getResourceData and insert values on DB
        res.send(newReslt.sensors.temperature.value);
   }else{
        res.send(reslt.sensors.temperature.value);
   }
 });
});
```

Código 11 – Exemplo de consulta de valores de um sensor de um dispositivo

Devido à necessidade de reduzir o consumo de energia dos dispositivos é necessário reduzir o número de pedidos aos dispositivos. Para tal, foi definido um intervalo de tempo em que o valor obtido pelos sensores é válido. Nestas interfaces é verificado se a base de dados contém alguma leitura do sensor, caso não exista é efetuado um pedido ao dispositivo e posteriormente inserido o valor na base de dados com um campo adicional que corresponde ao tempo atual. Quando a base de dados possui um valor para a leitura dos sensores é verificado se essa leitura ainda é correta comparando se a hora da última consulta não ultrapassou um intervalo pré-definido. Caso os dados estejam atualizados é retornado de imediato o valor contido na base de dados e caso contrário é efetuado um pedido ao sensor e atualizado o seu valor na base de dados. O Código 11 é uma simplificação do método que devolve a informação de um sensor de temperatura.

5 Demonstração

Neste capítulo será apresentada a interface da aplicação implementada demonstrando as suas funcionalidades e o modo de interação com os dispositivos. Para a criação desta interface foi utilizado a *framework* Twitter Bootstrap que, como acima referido, permite a criação de aplicações responsivas. Devido a isto será demonstrada a interface da aplicação executada em duas plataformas distintas, num *browser* de um computador e num dispositivo móvel. O nome escolhido para a aplicação foi HomeIoT.

5.1 Interceção com a aplicação

Nesta secção será demonstrada as interações feitas entre a aplicação e o servidor demonstrando as trocas de mensagens efetuada entre ambos.

5.1.1 Efetuar operações num recurso a partir da listagem de dispositivos

O utilizador pode visualizar todos os dispositivos presentes na rede, ao selecionar um desses dispositivos são listados os recursos desse dispositivo. O utilizador da aplicação nesta fase pode efetuar operações como atualizar e alterar os valores de um recurso e o servidor retorna o resultado dessa operação. A Figura 16 corresponde à interação entre o utilizador e o servidor para esta interação.

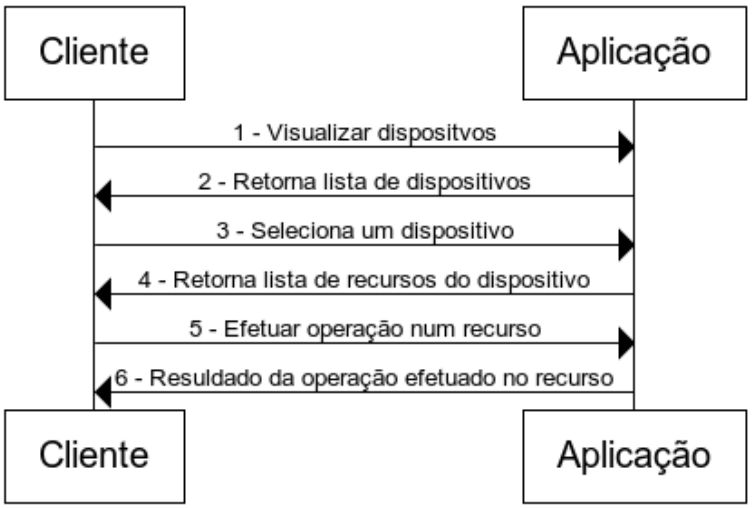

Figura 16 – Diagrama de interceção para alterar um recurso através da lista de dispositivos

5.1.2 Efetuar operações num recurso a partir de uma localização

O utilizador seleciona visualizar dispositivos por localização. Ao selecionar esta opção é listada todas as divisões da casa. O utilizador seleciona uma divisão e o servidor retorna todos os dispositivos presentes nessa divisão. O utilizador seleciona o dipositivo pretendido e efetua operações aos seus recursos.

Figura 17 - Diagrama de interceção para alterar um recurso através da localização do dispositivo

5.2 Página inicial

A página inicial da aplicação mostra as opções possíveis para visualizar os dispositivos. É possível escolher uma visualização por dispositivos e por localização. A Figura 18 representa a página inicial da aplicação num *browser*.

A escolha de visualização dos dispositivos é feita selecionando uma das opções presentes na barra de navegação na parte superior da página.

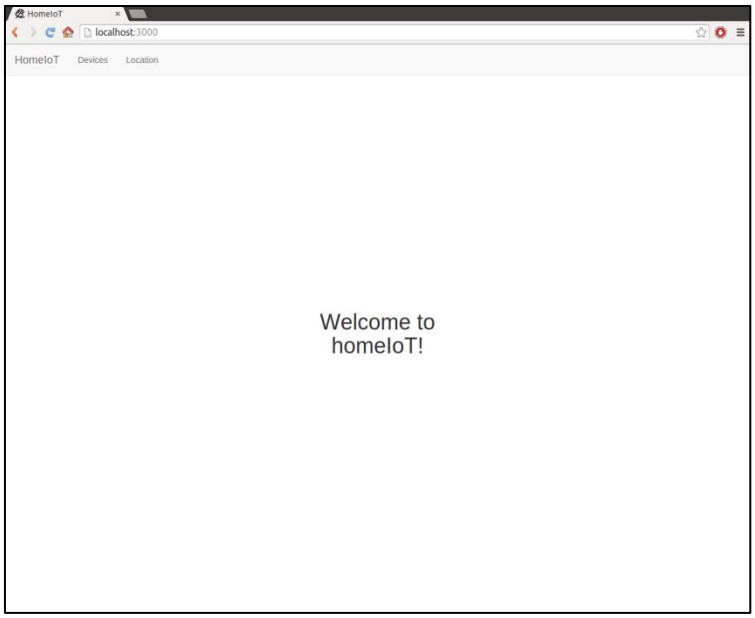

Figura 18 – Página inicial da aplicação acedida através de um *browser*

Na Figura 19 encontram se a página inicial acedida através de um dispositivo móvel. O ecrã do lado direito corresponde ao resultado de carregar no botão da aplicação que expande um menu que permite escolher a visualização pretendida dos dispositivos.

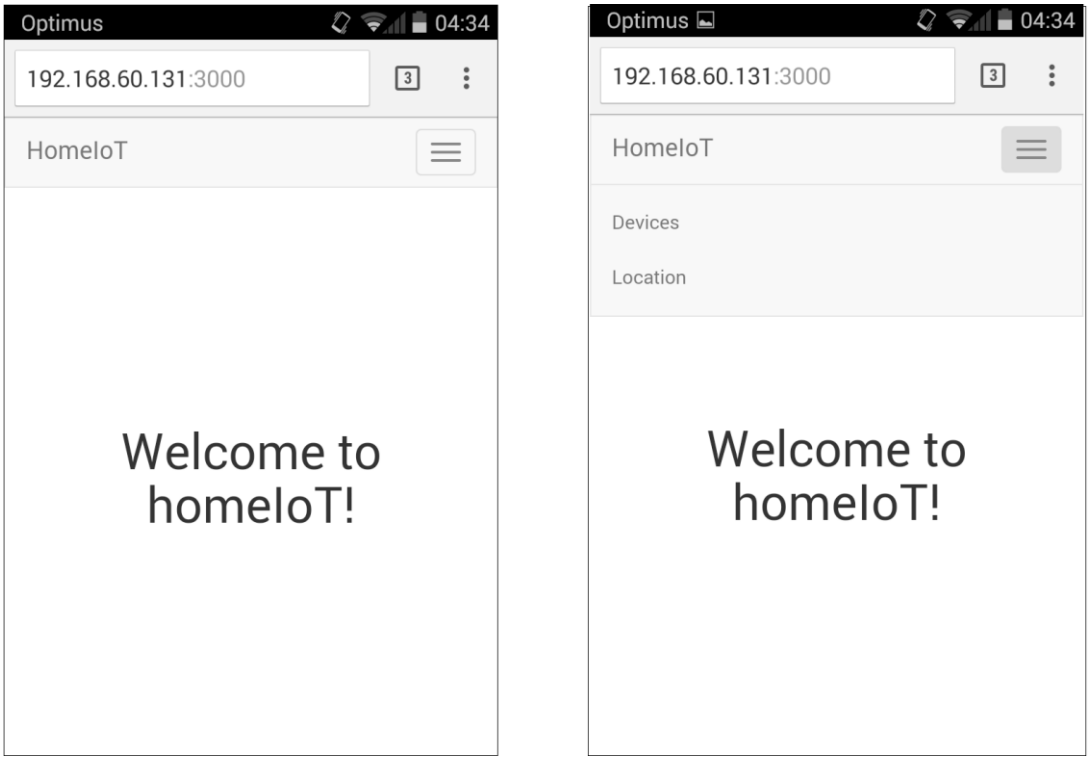

Figura 19 – Página inicial da aplicação acedida através de um dispositivo móvel

5.3 Listagem de todos os dispositivos

A escolha da opção *Devices* na página inicial permite uma listagem de todos os dispositivos ligados na rede. Essa escolha efetua um pedido à aplicação chamando a interface /nodes da API REST implementada. A Figura 20 e Figura 21 são as listagens efetuadas nas duas plataformas.

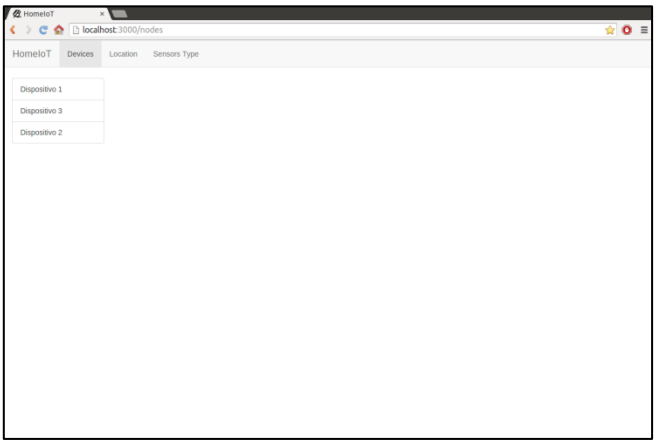

Figura 20 – Listagem de dispositivos no *browser*

Com esta listagem de dispositivos é possível selecionar um dispositivo para serem apresentados os seus recursos.

Figura 21 - Listagem de dispositivos no dispositivo móvel

5.4 Visualização e interação com os recursos dos dispositivos

A seleção de um dispositivo leva à chamada da interface /node/id/. São listados todos os recursos que esse dispositivo possui e é gerado um formulário que permite ver, atualizar e alterar os valores desses recursos. A Figura 22 e Figura 23 representam a listagem de recursos de um dispositivo.

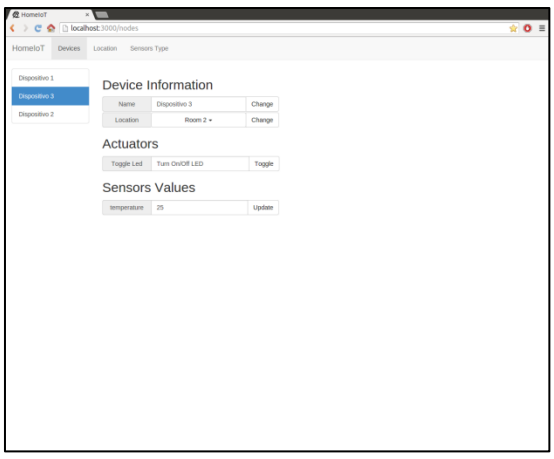

Figura 22 – Listagem dos recursos de um dispositivo no *browser*

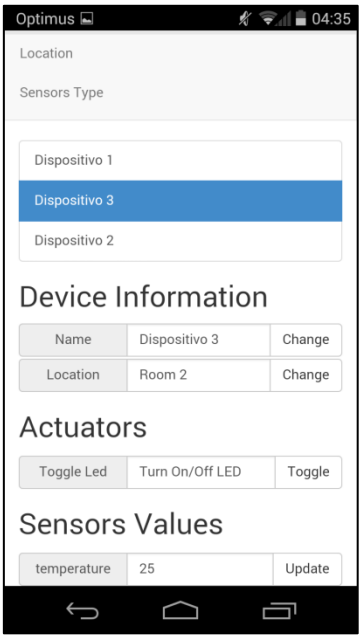

Figura 23- Listagem de recursos no dispositivo móvel

O recurso *Name* contém o nome do dispositivo e é possível alterá-lo inserindo o novo nome no campo de texto e carregando *Change*. O recurso *Location* também pode ser alterado mas devido a esse recurso corresponder a divisões de uma casa, as opções de alteração desse campo estão restringidas aos valores contidos na Drop-down list.

Tal como, a alteração dos valores do nome e da localização dos dispositivos também é possível alterar os estados dos atuadores. Essa alteração é efetuada através de um POST para cada um dos recursos.

Ao carregar *update* num sensor é efetuado uma nova leitura da informação.

5.5 Visualização dos dispositivos por localização

Ao selecionar a visualização dos dispositivos por localização é listada todas as divisões da casa. É possível selecionar todos os dispositivos assim como apenas os dispositivos de uma determinada divisão. Ao selecionar uma dessas opções é apresentada uma planta da casa onde os dispositivos são representados dentro da divisão em que se encontram.

A Figura 24 demonstra a listagem dos dispositivos por localização.

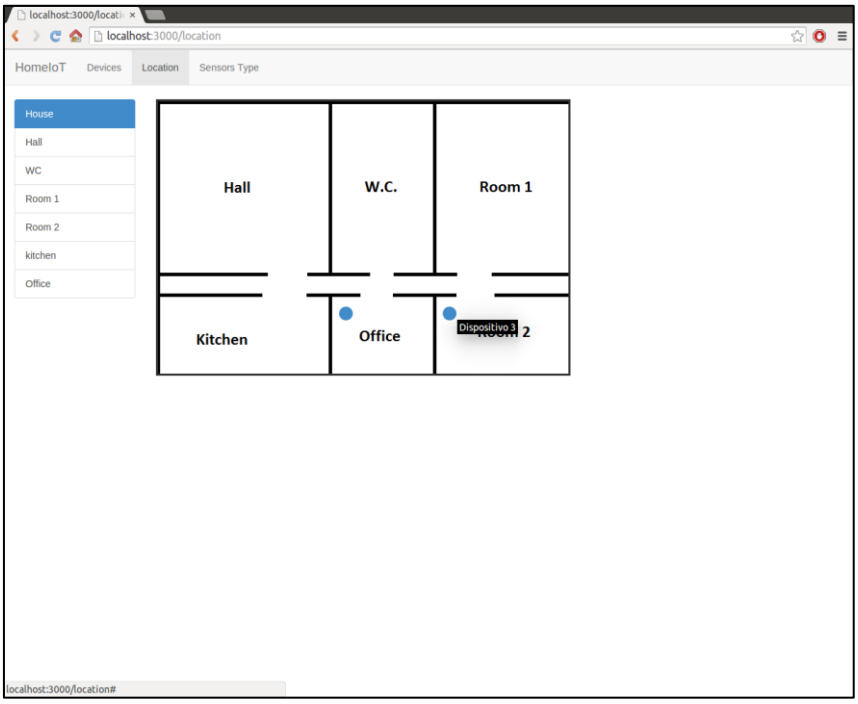

Figura 24 – Visualização dos dispositivos por localização no *browser*

Os pontos azuis que aparecem na planta da casa representam os dipositivos, ao passar com o rato sobre esses pontos é mostrado o nome do dispositivo. Ao carregar nos dispositivos é apresentada um formulário que lista os recursos do dispositivo e permite a sua manipulação.

A Figura 25 corresponde à listagem de dispositivos por localização em dispositivos móveis onde o princípio de funcionamento é igual ao do *browser*.

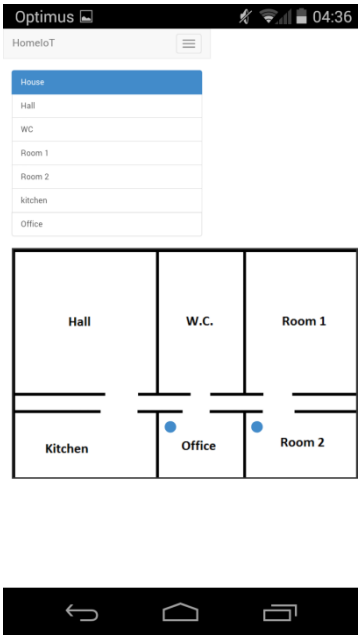

Figura 25 – Listagem dos dispositivos por localização em dispositivos móveis

6 Conclusões

Neste documento foi proposta a criação de uma aplicação que permita efetuar a monitorização de uma casa inteligente, constituída por vários dispositivos de baixa capacidade de processamento e energética. Devidas às características destes dispositivos, foi necessário recorrer a um protocolo de comunicação que tenha em consideração essas limitações e que permita a realização de trocas de mensagens de uma forma eficiente. O protocolo utilizado para esse fim foi o *Constrained Application Protocol* (CoAP).

De momento nenhum *browser* consegue suportar nativamente pedidos CoAP, devido a isso foi necessário efetuar um mapeamento de mensagens HTTP, suportadas pelos browsers, para mensagens CoAP e vice-versa. Para resolver este problema foi criada uma API REST que recebe pedidos HTTP e efetua pedidos CoAP a uma rede de dispositivos limitados.

Para ser possível uma monitorização eficiente e intuitiva de uma casa inteligente, foi desenvolvida uma aplicação web que faz os pedidos à API referida anteriormente. Esta aplicação permite efetuar descoberta de dispositivos numa rede limita e descobrir todos os recursos que cada dispositivo possui. Com este conhecimento de recursos, a aplicação efetua pedidos que permitem obter dados para a monitorização.

6.1 Avaliação e Trabalho Futuro

Devido ao CoAP ser um protocolo relativamente recente, houve uma dificuldade inicial na compreensão do mesmo. A falta de documentação e informação para a resolução de dúvidas foi um processo difícil que levou algum tempo. Devido a essa demora na resolução de problemas não foi possível a implementação de algumas funcionalidades desejadas. Essas funcionalidades ficam propostas como trabalho futuro.

- Tirar proveito do método de observe para efetuar a atualização da base de dados sem a necessidade do utilizador efetuar esses pedidos. Esta funcionalidade poderia ser implementada recorrendo a pedidos periódicos da aplicação à base de dados mas não seria uma solução viável pois, os sensores recebiam uma quantidade enorme de pedidos que podia levar a uma redução do tempo médio de vida das suas baterias. O observe permite que os sensores enviem a informação essencial para a aplicação evitando trocas de mensagens desnecessárias, otimizando o tempo de vida das suas baterias.
- Permiti que o utilizador pudesse definir tarefas, ou seja, programar o sistema para que este responda de uma forma automática a alterações do meio tornando assim a casa inteligente.

Referências

[Alan Ott, 2012]	Alan Ott. Wireless Networking with IEEE 802.15.4 and 6LoWPAN
[Dave Evans, 2011],	Dave Evans. The Internet of Things - How the Next Evolution of the Internet is Changing Everything
[IEEE Communication Magazine, 1997]	Brian P. Crow, Indra Widjaja, Jeong Geun Kim, Prescott T.. Sakai IEEE 802.11 Wireless Local Area Network
[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 Pique, Ignacio Almazan, Daniel Garcia, 2010]	Jose Pique, Ignacio Almazan, Daniel Garcia. BLUETOOTH
[Kevin Ashton, 2009]	Kevin Ashton. That 'Internet of Things' Thing, http://www.rfidjournal.com/articles/view?4986 [último acesso: Out 2014]
[M. Collina, 2013]	https://www.npmjs.org/package/coap
[M. Ruta, F. Scioscia, A. Pinto, E. Di Sciascio, F. Gramegna, S. Ieva, G. Loseto, 2013].	M. Ruta, F. Scioscia, A. Pinto, E. Di Sciascio, F. Gramegna, S. Ieva, G. Loseto, Resource annotation, dissemination and discovery in the Semantic Web of Things: a CoAP-based framework
[Microsoft, 2005].	Microsoft. The TCP/IP model, http://technet.microsoft.com/en-us/library/cc786900(v=ws.10).aspx [último acesso: Out 2014] The TCP/IP model: TCP/IP
[Mirko Rossini, 2011]	Mirko Rossini. Design e implementazione di un proxy HTTP/CoAP
[node.js,2014]	node.js. noejs.org [ultimo acesso: Out 2014]
[T. Capan, 2013]	Tomislav Capan, http://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js [último acesso: Out 2014]
[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