

**UNIVERSIDADE DO ESTADO DO AMAZONAS - UEA**  
**ESCOLA SUPERIOR DE TECNOLOGIA**  
**ENGENHARIA DE COMPUTAÇÃO**

DOUGLAS LIMA DANTAS

**ARQUITETURA DE IOT PARA CIDADES  
INTELIGENTES**

Manaus

2017

**DOUGLAS LIMA DANTAS**

**ARQUITETURA DE IOT PARA CIDADES INTELIGENTES**

Trabalho de Conclusão de Curso apresentado  
à banca avaliadora do Curso de Engenharia  
de Computação, da Escola Superior de  
Tecnologia, da Universidade do Estado do  
Amazonas, como pré-requisito para obtenção  
do título de Engenheiro de Computação.

Orientador: Prof. Dr. Carlos Maurício Seródio Figueiredo

Manaus

2017

**Universidade do Estado do Amazonas - UEA**

**Escola Superior de Tecnologia - EST**

*Reitor:*

**Cleinaldo de Almeida Costa**

*Vice-Reitor:*

**Mario Augusto Bessa de Figueiredo**

*Diretor da Escola Superior de Tecnologia:*

**Roberto Higino Pereira da Silva**

*Coordenador do Curso de Engenharia de Computação:*

**Raimundo Corrêa de Oliveira**

*Coordenador da Disciplina Trabalho de Conclusão de Curso:*

**Raimundo Corrêa de Oliveira**

*Banca Avaliadora composta por:*

*Data da Defesa: \_\_\_/\_\_\_/\_\_\_.*

**Prof. Dr. Carlos Maurício Seródio Figueiredo (Orientador)**

**Prof. Dra. Elloá Batteto Guedes da Costa**

**Prof. Dr. Raimundo Corrêa de Oliveira**

## **CIP - Catalogação na Publicação**

D192a

DANTAS, Douglas

Arquitetura de IoT para Cidades Inteligentes/ Douglas Dantas; [orientado por] Prof. Dr. Carlos Maurício Seródio Figueiredo - Manaus: UEA, 2017.

240 p.: il.; 30cm

Inclui Bibliografia

Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação). Universidade do Estado do Amazonas, 2017.

CDU: 004.3

**DOUGLAS LIMA DANTAS**

**ARQUITETURA DE IOT PARA CIDADES INTELIGENTES**

Trabalho de Conclusão de Curso apresentado à banca avaliadora do Curso de Engenharia de Computação, da Escola Superior de Tecnologia, da Universidade do Estado do Amazonas, como pré-requisito para obtenção do título de Bacharel em Engenharia de Computação.

**Aprovado em: \_\_\_/\_\_\_/\_\_\_.**

BANCA EXAMINADORA

---

**Prof. Carlos Mauricio Seródio Figueiredo, Doutor**

*UNIVERSIDADE DO ESTADO DO AMAZONAS*

---

**Prof. Elloá Barreto Guedes da Costa, Doutora**

*UNIVERSIDADE DO ESTADO DO AMAZONAS*

---

**Prof. Raimundo Corrêa de Oliveira, Doutor**

*UNIVERSIDADE DO ESTADO DO AMAZONAS*

## Dedicatória

*Dedico este trabalho a D-US, o Primeiro e Eterno Engenheiro, e a meus pais, Ivanilda Lima e Ciro Dantas, e a meus irmãos, Dennys e Denylson Dantas. Vocês são os meus maiores amores nesse mundo.*

# Agradecimentos

*“Nada temos que recear quanto ao futuro, a menos que esqueçamos a maneira em que o Senhor nos tem guiado, e os ensinos que nos ministrou no passado.” - Ellen G. White*

Primeiramente, agradeço a D-US pela paciência que teve comigo e por ter permitido que eu concluisse esse curso com saúde física e mental, provendo todos os meios para que isso fosse possível.

À minha mãe, Ivanilda, por todo o amor, paciência, apoio, dedicação, a educação que me deu e por poder dividir com ela minhas alegrias e lamentar minhas frustrações. Sei que foi-lhe ruim a minha ausência. A meu pai, Ciro, pela ajuda que me deu, inclusive financeira, e por ser meu exemplo de homem trabalhador e pacífico. Aos gêmeos Dennys e Denylson, por serem esses irmãos maravilhosos e tornarem a minha vida mais alegre.

Ao meu orientador, Prof. Dr. Carlos Maurício, pela ótima orientação e apoio, pela paciência com minhas dificuldades e pela solicitude em me ajudar, sempre respondendo às mensagens, até mesmo em fins-de-semana e feriados. Ao Samsung Ocean e ao Prof. Antenor Ferreira, por todas as oportunidades, pelo ótimo estágio e pela experiência, capacitação e confiança que adquiri, além de fornecer os recursos para que o projeto deste TCC fosse executado. Agradeço também aos professores Ismael Júnior e Allan Bezerra por todo o apoio. Mais do que colegas de trabalho, vocês são meus amigos.

A todos os demais professores e, em especial, ao Prof. Dr. Raimundo Oliveira por toda a ajuda, seja como coordenador do curso, seja pessoalmente. Com certeza sua ajuda foi crucial para que eu terminasse este curso antes do tempo previsto.

A todos os colegas de curso e amigos, por juntos termos caminhado, comemorado, sofrido, nos alegrado e rido mutualmente de nossos infortúnios. Em especial, a Diego, Eduardo, Abda e Branca. Sem vocês, essa caminhada não teria a mesma graça.

## Resumo

Diante do crescimento urbano, surge a preocupação com a qualidade de vida nas cidades. Paralelo a isso, a tecnologia tem avançado e gerado recursos que podem preencher algumas dessas lacunas. Este trabalho traz uma proposta de arquitetura baseada em Internet of Things para monitorar dados em ambientes urbanos e disponibilizá-los com uma visualização amigável, além de prover uma API para que um nicho de aplicações utilizem os dados. Tudo isso utilizando tecnologias acessíveis que, em conjunto, ajudam a solucionar alguns desses problemas. A principal contribuição deste trabalho é um sistema de baixo custo, altamente escalável e personalizável para uso com infra-estrutura urbana.

Palavras-chave: *Internet of Things*, Arduino, Android, Node-RED, Freeboard, Nós sensores, Cidades Inteligentes.

# Abstract

Faced with urban growth, there is a concern for a quality of life in cities. Parallel to this, the technology has advanced and generated features that can fill some of these gaps. This work brings a proposal of Internet of Things based architecture to monitor data in urban environments and make them available with a friendly view. All this using accessible technologies that together help solve some problems, and provide an API for a set of applications to use the data. All this using affordable technologies that together help solve some of these problems. The main contribution of this work is a low-cost, highly scalable and customizable system for use with urban infrastructure.

Keywords: Internet of Things, Arduino, Android, Node-RED, Freeboard, Sensor nodes, Smart cities.

# Sumário

<b>Lista de Tabelas</b>	<b>x</b>
<b>Lista de Figuras</b>	<b>xi</b>
<b>Lista de Códigos</b>	<b>xii</b>
<b>1 Introdução</b>	<b>1</b>
1.1 Descrição do Problema . . . . .	1
1.2 Objetivos . . . . .	2
1.2.1 Objetivo geral . . . . .	2
1.2.2 Objetivos específicos . . . . .	2
1.3 Justificativa . . . . .	2
1.4 Metodologia . . . . .	3
1.5 Organização do Trabalho . . . . .	5
<b>2 Fundamentos Teóricos e Trabalhos Relacionados</b>	<b>6</b>
2.1 Fundamentos Teóricos . . . . .	6
2.1.1 Principais Protocolos Utilizados em IoT . . . . .	6
2.1.2 Plataformas de Hardware . . . . .	8
2.1.3 Plataformas de Software . . . . .	11
2.1.4 Sensores . . . . .	13
2.2 Trabalhos e Projetos Relacionados . . . . .	16
2.2.1 Nós sensores . . . . .	16
2.2.2 Disponibilização de Dados de Sensores . . . . .	18
<b>3 A Arquitetura Projetada</b>	<b>19</b>
3.1 O Nó Sensor . . . . .	19
3.1.1 Módulo CPU . . . . .	21

3.1.2	Módulo de Alimentação . . . . .	21
3.1.3	Módulo de Comunicação . . . . .	22
3.1.4	Módulo de Sensoriamento . . . . .	22
3.1.5	Montagem . . . . .	22
3.2	MQTT <i>broker</i> . . . . .	23
3.3	Aplicação Node-RED . . . . .	24
3.4	<i>Dashboard</i> Freeboard . . . . .	24
3.5	Aplicativo móvel . . . . .	25
<b>4</b>	<b>Testes e Resultados</b>	<b>29</b>
4.1	Estudos de Casos . . . . .	29
4.1.1	Exposição em Ambiente . . . . .	29
4.1.2	Experimentos Isolados . . . . .	30
4.2	Resultados . . . . .	37
<b>5</b>	<b>Considerações Finais</b>	<b>38</b>
5.1	Conclusão . . . . .	38
5.2	Dificuldades Encontradas . . . . .	39
5.3	Trabalhos futuros . . . . .	39
<b>Referências Bibliográficas</b>		<b>41</b>
<b>Apêndices</b>		<b>45</b>
<b>A</b>	<b>Apêndice A Códigos-fonte</b>	<b>46</b>
A.1	Módulo CPU (Arduino) . . . . .	46
A.2	Módulo de Comunicação (NodeMCU) . . . . .	49
A.3	Artigo Aceito para Publicação no ENCOSIS 2017 . . . . .	65

# **Lista de Tabelas**

2.1	Tabela comparativa entre plataformas de hardware. . . . .	11
2.2	Comparação entre os trabalhos relacionados e o nó projetado neste. . . . .	18
3.1	Discriminação dos custos com o nó sensor. . . . .	21

# Listas de Figuras

1.1	Visão geral da arquitetura proposta pelo trabalho . . . . .	4
2.1	Representação dos níveis de QoS do MQTT [HiveMQ2017]. . . . .	8
2.2	Modelo Arduino UNO R3 com suas diferentes partes em destaque. [NTU2015]	9
2.3	Esquema representativo de um software produzido pelo Arduino IDE. . . .	9
2.4	Aspecto do NodeMCU [Arduino e Cia2016]. . . . .	10
2.5	Aspecto do editor do Node-RED. [JS Foundation2013]. . . . .	12
2.6	Visão em camadas do Android. [Android2017]. . . . .	14
2.7	Aspecto do sensor DHT11 [D-ROBOTICS2010]. . . . .	14
2.8	Ilustração do LDR e esquema de sua montagem juntamente com o resistor auxiliar. . . . .	15
2.9	Ilustração do sensor de fumaça e gases inflamáveis MQ-2 e seus pinos. . . .	16
2.10	Aspecto de dois dos nós sensores apresentados nos trabalhos relacionados. .	17
2.11	Diagrama esquematizando o FIWARE Lab [Fiware2016]. . . . .	18
3.1	Visão geral da arquitetura proposta pelo trabalho . . . . .	20
3.2	Divisão do nó sensor em módulos. . . . .	20
3.3	Fluxograma descrevendo o funcionamento do software embarcado na CPU.	21
3.4	Diagrama esquemático do circuito. . . . .	23
3.5	Gabinete confeccionado com auxílio de impressora 3D. . . . .	23
3.6	Serviço Web feito usando Node-RED para disponibilizar dados. . . . .	24
3.7	Interfaces WEB e móvel desenvolvidas para visualização amigável e em tempo real dos dados. . . . .	26
3.8	Diagrama de classes do aplicativo. . . . .	27
3.9	Diagrama de sequência do aplicativo. . . . .	28
4.1	Gráfico da temperatura. . . . .	31
4.2	Gráfico da umidade. . . . .	31

4.3	Gráfico de concentração de gases inflamáveis e fumaça. . . . .	32
4.4	Gráfico da luminosidade. . . . .	32
4.5	Visão no aplicativo e Freeboard da variação da concentração de gases detectada pelo nó sensor. . . . .	33
4.6	Visão no aplicativo e Freeboard da variação de temperatura detectada pelo nó sensor. . . . .	34
4.7	Visão no aplicativo e Freeboard da variação de luminosidade detectada pelo nó sensor. . . . .	35
4.8	Visão no aplicativo e Freeboard da variação de umidade detectada pelo nó sensor. . . . .	36

# **Lista de Códigos**

A.1	Código-fonte destinado ao Arduino. . . . .	46
A.2	Código-fonte destinado ao NodeMCU. . . . .	49
A.3	Código-fonte do aplicativo Android desenvolvido para visualização dos dados dos sensores. . . . .	53
A.4	Código-fonte do layout do aplicativo Android desenvolvido para visualização dos dados dos sensores. . . . .	58

# **Capítulo 1**

## **Introdução**

### **1.1 Descrição do Problema**

A globalização, urbanização e industrialização têm sido as principais mudanças da humanidade no século XXI [Lee et al.2014]. Além disso, a população mundial aumentará de 7 bilhões para 9 bilhões até 2050, e 70% das pessoas viverão nas áreas urbanas [Marchal et al.2012]. Há muitos problemas que afetam a qualidade de vida dos urbanos, sendo alguns deles a poluição em suas várias vertentes (atmosférica, sonora, visual, etc.), mudanças climáticas, ilhas de calor, etc. Sendo assim, torna-se imprescindível trazer soluções para mitigá-los.

Uma forma de combater esse problema é tornar as cidades mais inteligentes, obtendo dados do ambiente constantemente com o fim de tomar decisões e ter um maior controle. Faz-se necessário desenvolver soluções de monitoramento que informem dados do ambiente com precisão suficiente e dentro de uma taxa aceitável de tempo. No entanto, tal desafio é cercado de várias restrições, como acesso instável à Internet, providência de fontes energéticas, etc. Nesse contexto é que entra o ramo da “Internet das Coisas” (*Internet of Things - IoT*) [Xia et al.2012]. IoT trata da interconexão em rede de objetos do quotidiano, que são muitas vezes equipados com microcontroladores, provendo computação pervasiva e sensível ao contexto. Isto traz uma vasta gama de possibilidades de desenvolvimento de novos projetos que prometem melhorar a qualidade de vida das pessoas. Além disso, no

ramo da IoT considera-se muitas vezes restrições tais como as que existem nesse problema.

A solução proposta nesse trabalho traz os princípios de IoT para contornar essas restrições. A vantagem é utilizar uma nova abordagem que mais se adequa ao problema do que a anterior, com maior facilidade de projeto, gerenciamento e baixo custo. Essa rede de sensores pode ser utilizada, feitas as devidas adaptações, nos mais diversos tipos de ambiente, como em indústrias, ruas, parques e laboratórios, monitorando dados como temperatura, concentração de gás, umidade e luminosidade, fazendo uso do protocolo *MQ Telemetry Transport* (MQTT) [Banks and Gupta2014], bastante leve e ideal para ambientes críticos [Hunkeler et al.2008]. Além disso, conterá uma API disponível em formato JSON e um *dashboard* amigável para análise dos dados.

## 1.2 Objetivos

### 1.2.1 Objetivo geral

Projetar, implementar e experimentar uma arquitetura de IoT para aplicações de monitoramento de cidades, compreendendo dispositivos embarcados, comunicação e visualização de dados.

### 1.2.2 Objetivos específicos

- Avaliar comparativamente tecnologias de IoT;
- Disponibilizar um protótipo funcional do sistema;
- Desenvolver um estudo de caso de monitoramento para dados de temperatura, umidade, luminosidade e concentração de gases.

## 1.3 Justificativa

A criação de estruturas que tornem as cidades mais inteligentes tem sido uma das principais preocupações dos governos. Um exemplo disso, é o fato do Ministério das Co-

municações ter publicado a portaria nº 2.111, de 11 de maio de 2016, que prevê ações na área de IoT. Para isso, haverá implantação de infraestrutura e utilização de tecnologia para monitorar serviços como iluminação, trânsito e segurança. O objetivo é obter dados que permitam uma gestão mais eficiente desses setores por parte das prefeituras [BRASIL2016].

Em 2012 havia 143 projetos de cidades inteligentes, sendo 35 na América do Norte, 11 na América do Sul, 47 na Europa, 40 na Ásia e 10 na África e Médio Oriente. Na Ásia e Oriente Médio prevalecem iniciativas associadas à construção de cidades a partir do zero, como são os casos de Masdar, nos Emirados Árabes Unidos e de Songdo, na Coreia do Sul. Na Europa e na América do Norte imperam projetos de renovação urbana inteligente, de que são exemplos “*Amsterdam Smart City*” na Holanda e “*SmartSantander*” na Espanha [Lee et al.2014].

IoT é um paradigma novo que está ganhando terreno rapidamente no cenário das modernas telecomunicações sem fio. A idéia básica deste conceito é a presença generalizada de uma variedade de coisas ou objetos - como *Radio-Frequency IDentification* (RFID), sensores, atuadores, telefones celulares, etc., que, através de esquemas de endereçamento único, são capazes de interagir uns com os outros e cooperar com seus vizinhos para alcançar objetivos comuns [Atzori et al.2010].

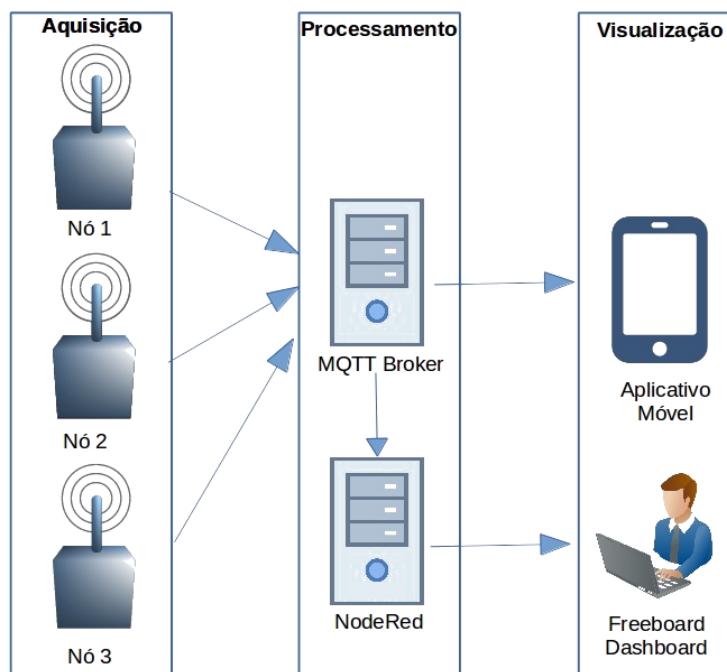
Vê-se que são muitas as preocupações com a qualidade de vida das cidades. A IoT tem potencial para mitigar esses problemas, tornando cidades mais inteligentes. Esse é o objetivo deste trabalho, criar uma arquitetura eficaz e de baixo custo para monitoramento de dados das cidades.

## 1.4 Metodologia

O processo consiste em uma sequência de passos para construção da arquitetura, que consistirá num protótipo entregável ao final do trabalho. Ao fim de cada etapa são geradas peças e documentações. As etapas são descritas à seguir:

- **Pesquisa bibliográfica sobre IoT e cidades inteligentes:** foi conduzida uma pesquisa nas principais fontes sobre tecnologias IoT e as necessidades das cidades inteligentes, para que houvesse maior noção das necessidades;

- **Projeto do sistema:** Fez-se um projeto do sistema embarcado e seu software , da topologia de rede e *dashboard* de visualização dos dados. Uma visão geral do projeto é apresentado na Figura 1.1;
- **Implementação do protótipo:** A partir do projeto foi construído um protótipo que foi implantada em um local escolhido para esse propósito;
- **Testes e avaliação:** Houve uma série de testes, verificando o funcionamento da arquitetura, sua qualidade e análise de dados;
- **Documentação:** Após todas as etapas anteriores concluídas, fez-se uma documentação explicando o funcionamento do protótipo, para que possa ser consultado sempre que necessário.



**Figura 1.1:** Visão geral da arquitetura proposta pelo trabalho

## 1.5 Organização do Trabalho

Este trabalho está dividido desta forma: O Capítulo 2 contém a base teórica deste trabalho, citando as principais tecnologias utilizadas em IoT, algumas usadas no projeto e outras não, além dos trabalhos relacionados. Já o Capítulo 3 contém o desenvolvimento do projeto. O Capítulo 4 apresenta os resultados obtidos e discute acerca a contribuição deles. Por fim, o Capítulo 5 mostra as considerações finais.

# **Capítulo 2**

## **Fundamentos Teóricos e Trabalhos Relacionados**

Este capítulo tem duas seções principais: Fundamentos e Trabalhos Relacionados. A primeira aborda sobre as tecnologias utilizadas no projeto, ao passo que a segunda apresenta um comparativo entre projetos relacionados a este trabalho.

### **2.1 Fundamentos Teóricos**

#### **2.1.1 Principais Protocolos Utilizados em IoT**

O advento da IoT, com seu paradigma que geralmente utiliza sistemas embarcados conectados em rede, trouxe uma série de necessidades: tecnologias com baixo consumo de energia e capazes de trabalhar com uma baixa disponibilidade de banda, com conexões muitas vezes instáveis. Na área de redes, isso ocasionou o desenvolvimento de vários protocolos, desde as camadas mais baixas até as mais altas na pilha de protocolos. Dentre os principais protocolos na camada de aplicação são utilizados em aplicações de IoT, podemos citar o CoAP [Shelby et al.2014] e o MQTT [Hunkeler et al.2008] [ISO/IEC20922 2016].

## Constrained Application Protocol (CoAP)

O CoAP [Shelby et al.2014] foi desenvolvido pelo CoRE (Constrained Resource Environments), um grupo de trabalho da IETF. Ele é um protocolo de transferência de documentos semelhante ao HTTP, porém é muito leve, tanto em gasto de energia quanto em consumo de banda. Isso se deve ao fato de executar sobre UDP, fazendo tentativas de conexão e reordenamento de pacotes na camada de aplicação. Ele segue o modelo cliente/servidor.

O CoAP tem quatro tipos de mensagens, sendo eles:

- **Confirmable (CON):**

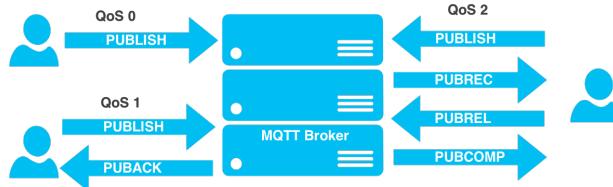
Mensagem que exige que a cada pacote enviado haja uma resposta (ACK) para confirmar a entrega e para que se envie o próximo pacote.

- **Non-confirmable(NON):** Não exige confirmação de entrega.
- **Acknowledgement (ACK):** Mensagem enviada para confirmar o recebimento de uma CON.
- **Reset:** Mensagem enviada quando se duvida da integridade de uma CON recebida, solicitando o seu re-envio.

## Message Queuing Telemetry Transport (MQTT)

O MQTT [Banks and Gupta2014] é um protocolo aberto extremamente leve para comunicação entre máquinas. Foi inicialmente criado pela IBM. Tem uma arquitetura cliente servidor, onde cada nó é um cliente e comunica-se com um servidor chamado *broker*. Ele utiliza TCP, em contraste ao uso de UDP pelo CoAP, e é baseado no padrão *publish/subscriber*, onde um nó pode publicar mensagens e/ou sobrescrever em um ou mais endereços chamados de tópicos. Uma funcionalidade interessante deste protocolo é o *Quality of Service* (QoS), onde pode-se configurar entre três níveis de garantia de entrega das mensagens. A Figura 2.1 apresenta uma representação dos três níveis de QoS. O nível QoS 0 é o mais básico, e funciona de forma semelhante ao UDP. Ele utiliza apenas um verbo responsável por publicar a mensagem, todavia sem confirmação de entrega e da ordem correta dos pacotes. Já o segundo nível, o QoS 1, utiliza dois verbos, um para publicar e outro para confirmar a entrega, mas ainda sem garantia de ordem de chegada. Por fim, o último nível (QoS 2)

utiliza quatro verbos, publicando e garantindo a entrega dos pacotes e ordem correta de entrega.



**Figura 2.1:** Representação dos níveis de QoS do MQTT [HiveMQ2017].

O MQTT foi o protocolo escolhido para ser utilizado no projeto descrito neste trabalho. As razões para isso são a sua arquitetura simples, com pequenas mensagens, juntamente com o modelo *publish/subscriber*, tornando o mesmo mais adequado às necessidades.

### 2.1.2 Plataformas de Hardware

O bom funcionamento de um projeto de IoT depende da escolha da plataforma de hardware utilizada. Com a diminuição dos custos de produção e melhorias nas tecnologias de fabricação de semicondutores, surgiram várias plataformas baratas e simples de programar, democratizando a fabricação de dispositivos e criando toda uma comunidade em volta disso. Dentre as plataformas mais populares podemos citar o Arduino [Arduino2005], ESP8266 [Espressif2014] e Raspberry Pi [RPF2012]. A Tabela 2.1.2 lista a diferença entre as principais plataformas de hardware.

#### Arduino

É uma plataforma de prototipagem eletrônica, de código e hardware aberto, projetada utilizando um microcontrolador da família Atmel AVR, com suporte a entrada, saída e conexão serial via USB, operando em 5V. Foi criado tendo um baixo custo e pensando em usuários iniciantes como seu público-alvo, tais como designers e artistas [Arduino2005]. Na sua versão UNO R3 utiliza o microcontrolador Atmel 328p, sendo o modelo utilizado neste trabalho e representado na Figura 2.2.

Contém saídas digitais, analógicas e é utilizado desde em projetos simples, como pequenas artes, quanto projetos complexos, como impressoras 3D. Esta foi a placa utilizada no

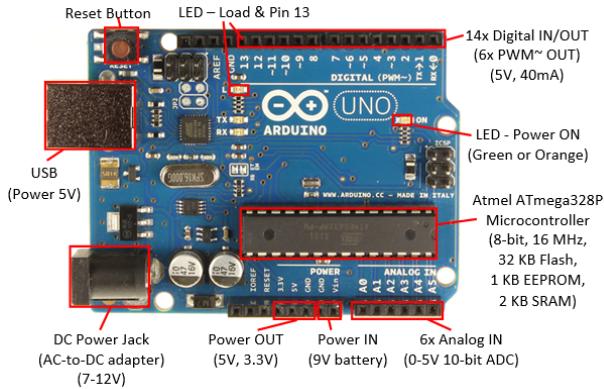


Figura 2.2: Modelo Arduino UNO R3 com suas diferentes partes em destaque. [NTU2015]

projeto proposto por esse trabalho, devido ao seu baixo custo, além várias saídas digitais e analógicas e um grande número de módulos compatíveis.

Ele pode ser programado em C/C++ utilizando o Arduino IDE, um *software* multi-plataforma para o qual estão disponíveis muitas bibliotecas. Suas regras de programação podem ser conferidas em [Arduino2005].

A Figura 2.3 ciclo de programação no Arduino IDE contém duas funções: *setup()*, que é executada uma única vez e onde se faz as configurações de inicialização, e *loop()*, que é um laço infinito contendo uma rotina que é a função do programa em si.

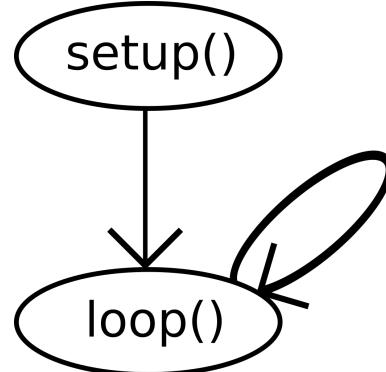


Figura 2.3: Esquema representativo de um software produzido pelo Arduino IDE.

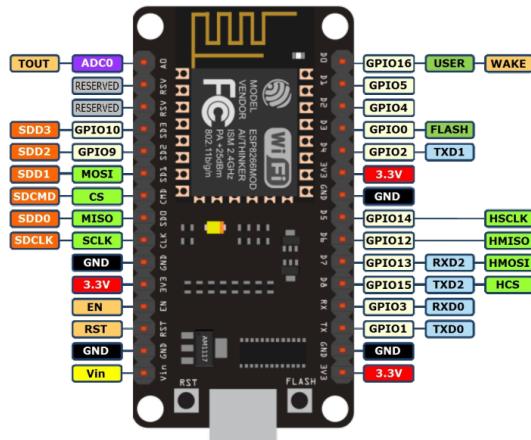
## ESP8266

O ESP8266 é um microcontrolador desenvolvido pela empresa Espressif Systems. No mercado desde 2014, ele já vem com comunicação WiFi 802.11b/g/n integrado, possui um

baixo consumo de energia, opera em 3,3V e pode ser utilizado tanto como módulo WiFi por outro microcontrolador, através de comunicação serial, quanto de maneira independente. Contém portas digitais GPIO com o número variando de acordo com o modelo.

Além disso, ele pode ser programado utilizando várias linguagens, como LUA, Python, C/C++ e através do Arduino IDE, utilizando muitas bibliotecas compatíveis com Arduino e facilitando a programação.

Existem vários *kits* de desenvolvimento baseado nesse *chip*, tal como o NodeMCU, que baseia-se no ESP8266-12 ou ESP8266-12E, sendo o último utilizado neste projeto como módulo de comunicação. A Figura 2.4 apresenta uma imagem do mesmo, com suas várias portas digitais e uma analógica.



**Figura 2.4:** Aspecto do NodeMCU [Arduino e Cia2016].

## Raspberry Pi

O Raspberry Pi é um computador SoC (*System on a Chip*) de baixo custo, com um processador baseado em ARM. Ele opera em 3,3V e contém entradas e saídas digitais, entradas Ethernet, USB e saídas de áudio e vídeo (AV e HDMI), variando de acordo com o modelo. Foi lançado pela Raspberry Pi Fundation em 2012, com o propósito de introduzir princípios de Ciência da Computação no ensino de base. Executa sistemas operacionais baseados em Linux e tem a capacidade de executar vários funções de um computador pessoal, como interface gráfica, suítes de escritório, etc. [Lawler2012] [Cellan-Jones2012]. Potencialmente pode ser programado utilizando qualquer linguagem, mas as

mais comuns são Python, Java, C, C++ e *Shell Script*. Tendo como seu ponto forte sua grande potência, em termos de processamento em memória, é utilizado em muitos projetos tais como celulares, drones, consoles de jogos, CPU de controle para outras plataformas, etc. O motivo de não utilizar-se no trabalho proposto foi o fato de não ter portas analógicas, necessárias para o uso de vários tipos de sensores, além de ter um custo consideravelmente alto em relação às plataformas anteriores, e do seu processamento muito avançado para uma tarefa simples. Estes fatos inviabilizaram o uso desta plataforma neste projeto.

Característica	Arduino	ESP8266	Raspberry PI
Modelo	Uno R3	12E	B
Processador	Atmega328	Tensilica Xtensa LX106	ARM11
Clock	16MHz	80MHz	700MHz
E/S Digital	Sim	Sim	Sim
E/S Analógica	Sim	Sim	Não
Tensão	5V	3,3V	3,3V
RAM	2KB	160KB	512MB
Preço	R\$30,00	R\$25,00	R\$225,00

**Tabela 2.1:** Tabela comparativa entre plataformas de hardware.

### 2.1.3 Plataformas de Software

No desenvolvimento IoT faz-se o uso de diferentes plataformas de software em várias categorias, onde pode-se destacar a programação, visualização de dados e sistemas operacionais.

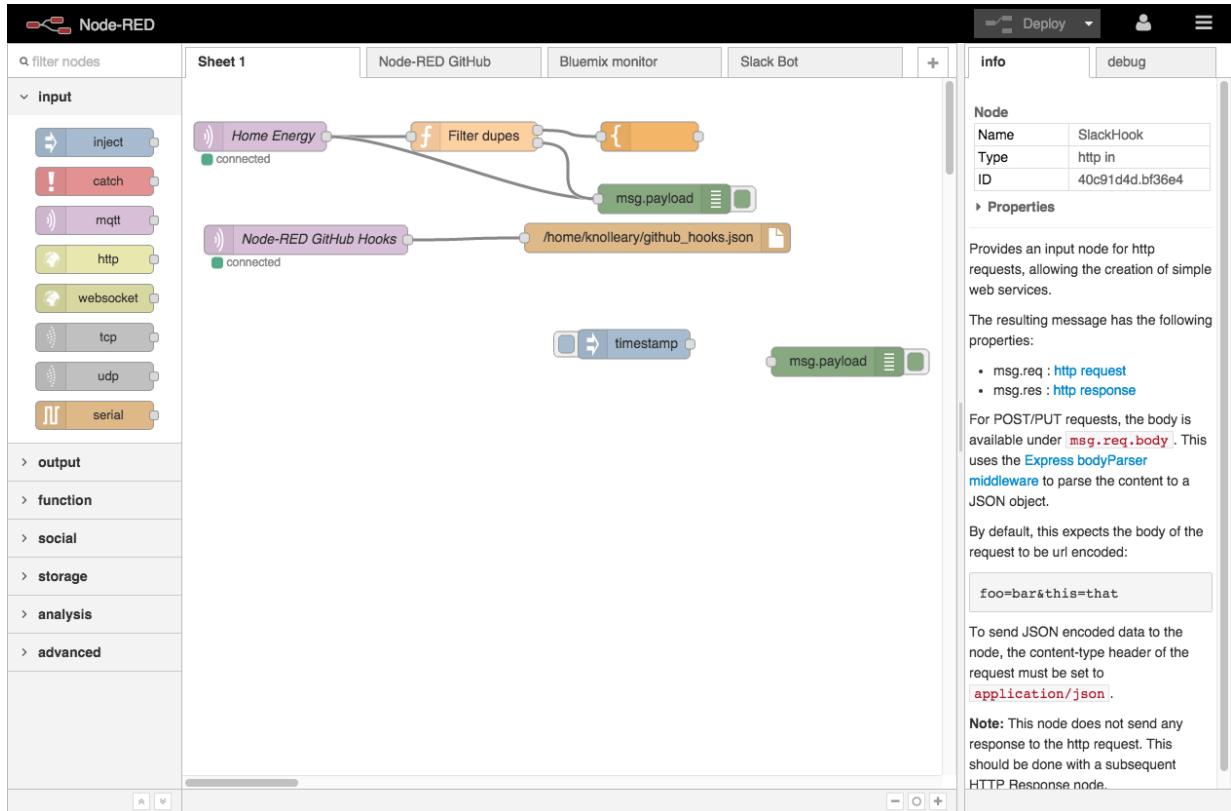
#### Node-RED

O Node-RED [JS Foundation2013] foi desenvolvido pela JS Foundation e é uma ferramenta visual baseada em fluxo de blocos para programar rapidamente a comunicação entre APIs, dispositivos, disponibilização de dados, etc., com o fim de desenvolver aplicações IoT rapidamente. Ele foi produzido utilizando o NodeJS, um *framework* de desenvolvimento Web baseado em Javascript.

Node-RED tem um editor de fluxos para ser utilizado em navegadores. Nele é possível criar uma aplicação arrastando blocos da paleta para a área de trabalho e então ligá-los. Ao fim, clica-se em *DEPLOY* para executar o fluxo. A paleta pode ser expandida instalando-se

novos blocos criados pela comunidade, que são disponibilizados em formato JSON. Dentre suas principais funcionalidades tem-se a conexão MQTT, HTTP, TCP, UDP, depuração, escrita em arquivos, etc. A Figura 2.5 apresenta um aspecto do editor Node-RED com sua paleta e área de trabalho.

Neste trabalho ele foi utilizado para criar rapidamente um serviço Web que disponibilizasse em formato JSON os dados recebidos via MQTT, tornando possível o uso por várias aplicações diferentes.



**Figura 2.5:** Aspecto do editor do Node-RED. [JS Foundation2013].

## Freeboard

Freeboard é um mecanismo de *dashboards* baseado em HTML. Com ele é possível criar painéis amigáveis para visualização de dados gerados, dispondo de oito tipos de gráficos diferentes. Ele obtém estes dados a partir de uma fonte definida pelo usuário, chamada de *datasource*, podendo ser vários tipos de serviço Web diferentes.

Ele pode ser tanto baixado e instalado em um servidor quanto utilizado através do serviço de nuvem gratuito provido pelo fabricante. Na segunda opção, cada usuário pode criar seu próprio *dashboard*, que ficará público e disponível para consulta de todos os outros usuários. Neste trabalho optou-se pelo serviço em nuvem, com o fim de disponibilizar os dados recolhidos através dos sensores para a comunidade do Freeboard.

## Android

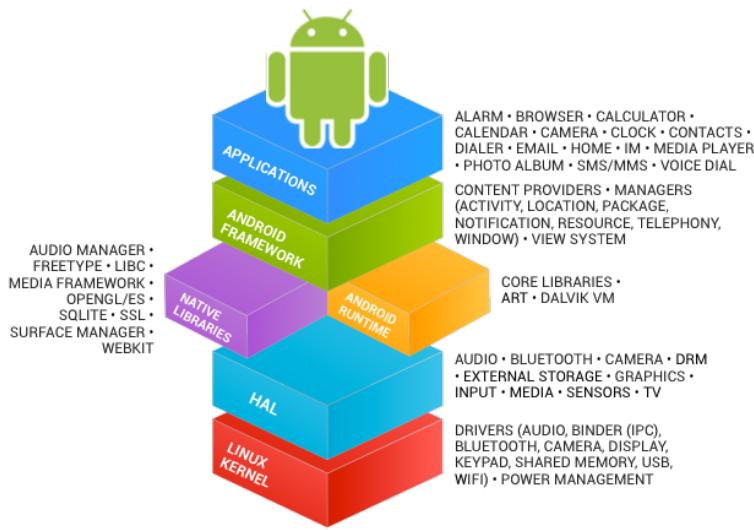
O Android [Android2017] é um sistema operacional de código aberto e baseado em Linux criado para o mercado de dispositivos móveis e com suporte a diferentes tipos de hardware. Foi originado por um consórcio de empresas chamado *Open Handset Alliance* (OHA), liderado pelo Google. Cada empresa fabricante que faça uso dele pode produzir sua versão personalizada e distribuí-lo, além de haver uma grande comunidade de desenvolvedores independentes que também disponibiliza versões. A Figura 2.6 apresenta as diferentes camadas do sistema Android, onde é possível observar que ele traz desde o *Kernel Linux* até um conjunto de aplicações para uso direto do usuário. Até agosto de 2015, 83,8% dos *smartphones* utilizavam Android, dominando a maior parte da fatia de mercado [Gartner, Inc.2015].

A ferramenta principal de desenvolvimento Android é o IDE Android Studio, utilizando um conjunto de bibliotecas chamadas Java SKD, baseado na Java, utilizando uma máquina virtual diferenciada chamada Dalvik Virtual Machine. Existe ainda o Android NDK, que é a programação nativa em C/C++, além de outras opções.

Devido a todas essas características, foi a plataforma escolhida para que se fizesse uma aplicação móvel com um painel para uma visualização amigável e portátil dos dados.

### 2.1.4 Sensores

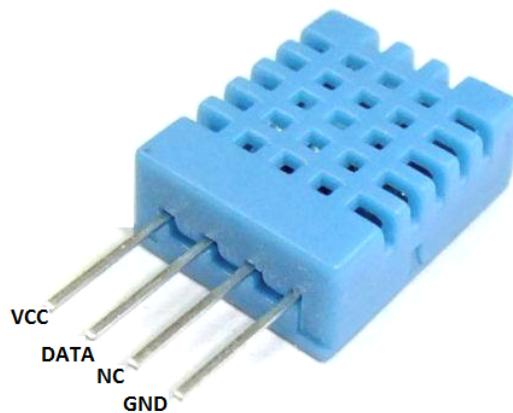
Sensor é um dispositivo capaz de converter estímulos do ambiente em sinais elétricos, sejam eles químicos, físicos, luminosos, etc., tornando-os interpretáveis a um dispositivo computacional. Neste projeto utilizou-se sensores de temperatura, umidade, luminosidade e de gases.



**Figura 2.6:** Visão em camadas do Android. [Android2017].

### Sensor de umidade relativa do ar e temperatura

O sensor utilizado para medir a umidade relativa do ar e temperatura do ambiente foi o DHT11 [D-ROBOTICS2010], cujo as medições apresentam margens de erro de respectivamente  $\pm 5\%$  e  $\pm 20^{\circ}\text{C}$ . Há uma biblioteca disponível para Arduino que abstrai seu uso. A Figura 2.7 apresenta um aspecto do sensor juntamente com o nome de seus pinos. Ele se comunica com o microcontrolador através de uma saída digital, onde uma sequência de pulsos pode ser decodificada para obter os dados.



**Figura 2.7:** Aspecto do sensor DHT11 [D-ROBOTICS2010].

## Sensor de luminosidade

O sensor de luminosidade utilizado neste trabalho é um fotoresistor (LDR) [EMANT2004]. Seu funcionamento baseia-se na variação da sua resistência em função da luz. Se há um ambiente de alta escuridão, sua resistência tende ao infinito, ao passo que quanto maior a luminosidade, sua resistência tende a 0. Logo, necessita-se de um resistor auxiliar em série com o LDR para limitar a corrente no circuito. Sua leitura é feita através de porta analógica e conversão direta do sinal de saída em medida de iluminância. A Equação 2.1 relaciona a resistência de um típico LDR a iluminância do ambiente, que é medida em lux (lx). A Equação 2.2 relaciona a tensão de saída de um LDR em função do resistor auxiliar, da tensão de alimentação e da resistência atual do LDR. A Equação 2.3 é a união de 2.1 e 2.2, relacionando a iluminância a resistência auxiliar, tensão de alimentação e tensão de saída LDR. A Figura 2.8 apresenta uma ilustração do LDR e seu esquemático juntamente com o resistor auxiliar.

$$R_{LDR} = \frac{500}{iluminancia} \quad (2.1)$$

$$V_{saída} = V_{alimentacao} \times \frac{R_{LDR}}{R_{LDR} + R_{auxiliar}} \quad (2.2)$$

$$iluminancia = \left( \frac{500}{R_{auxiliar}} \right) \left( \frac{V_{alimentacao}}{V_{saída}} - 1 \right) \quad (2.3)$$

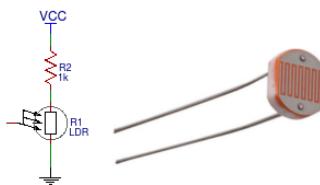


Figura 2.8: Ilustração do LDR e esquema de sua montagem juntamente com o resistor auxiliar.

## Sensor de Gases Inflamáveis e Fumaça

O sensor utilizado neste trabalho para detectar gases inflamáveis e fumaça foi o MQ-2 [SANDBOX ELECTRONICS2016]. Ele é capaz de detectar o nível LPG, propano,

metano, hidrogênio, gás carbônico, monóxido de carbono e outros gases num intervalo de concentração entre 300ppm e 10000ppm. Tem duas saídas de comunicação, a digital, que emite um sinal booleano que indica a presença ou não de gás e ajustável por um potenciômetro no módulo, e a analógica, que emite uma tensão diretamente proporcional à concentração de gases, sendo a utilizada neste trabalho. A Figura 2.9 apresenta um aspecto de um módulo MQ-2 e os nomes de seus pinos.



**Figura 2.9:** Ilustração do sensor de fumaça e gases inflamáveis MQ-2 e seus pinos.

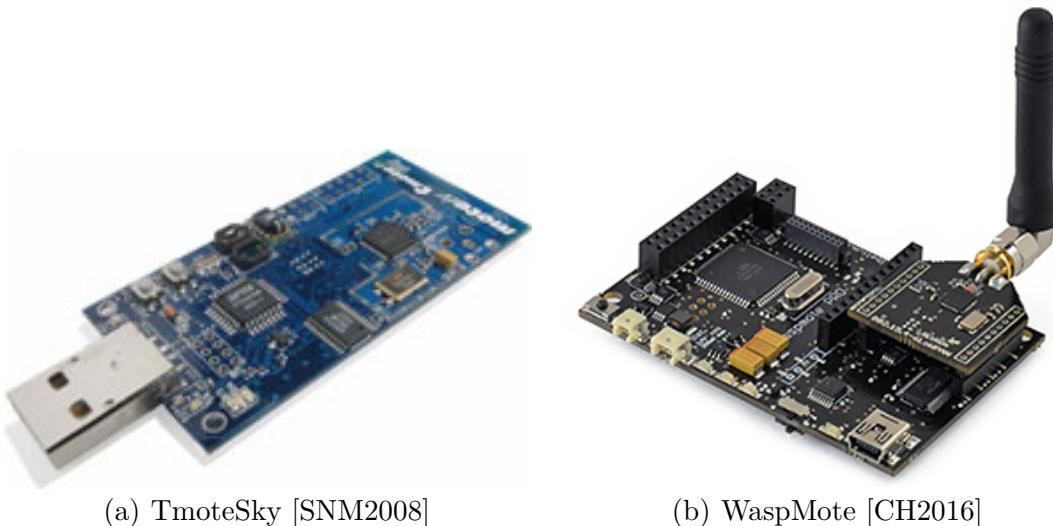
## 2.2 Trabalhos e Projetos Relacionados

Vários projetos de nós sensores e arquiteturas de IoT comerciais estão disponíveis no mercado. Há, também, inúmeros artigos na Internet que ensinam aos usuários como fazer seus próprios nós sensores. Além disso, há trabalhos acadêmicos que desenvolveram propostas de arquiteturas semelhantes a apresentada neste trabalho.

### 2.2.1 Nós sensores

Um trabalho relacionado é o *Forest Fire Sensor* [Santos and Figueiredo2016], uma Rede de Sensores Sensores Sem Fio (RSSF) que também utiliza a plataforma Arduino e alguns sensores idênticos aos utilizados neste trabalho, porém utiliza outros protocolos de comunicação além do WiFi, como o XBee, por exemplo. No entanto, o trabalho é voltado para monitoramento de ambientes como florestas, indústrias, parques, etc., enquanto

a arquitetura proposta neste tem como fim monitoramento em cidades para utilizar disponibilizar os dados em formato amigável para uma boa visualização, além de uma API para consumo em outros sistemas. Os mesmos citam e fazem uma análise em seu trabalho *Tmote Sky* [Advanticsys2015] e o *Wasp mote* [Liberium2012], dois nós sensores disponíveis no mercado bastante utilizados. A Figura 2.10 apresenta imagens de ambos os nós sensores.



**Figura 2.10:** Aspecto de dois dos nós sensores apresentados nos trabalhos relacionados.

O Tmote Sky utilizado em projetos RSSF. Tendo menor custo em relação a outros do mercado, de €80, o que equivale a R\$269,60 (cotação do dia 21 de abril de 2017), é utilizado em trabalhos onde haja um risco maior de perda do equipamento. Seu custo/benefício compensa em casos onde não seja necessário um alto poder de processamento. No entanto, caso necessite-se de um processamento maior e transmissões na faixa de quilômetros compensa mais utilizar o WaspMote, embora seja mais caro, já que este dispõe de um hardware mais potente e maiores pinos de expansão.

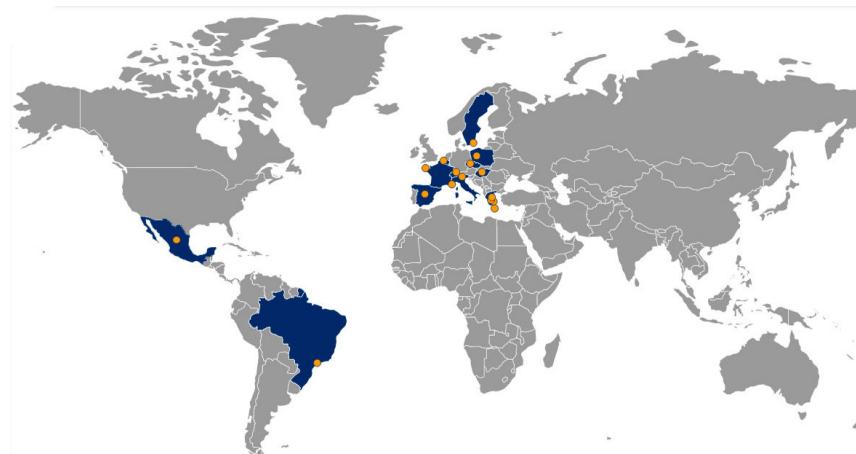
A Tabela 2.2 apresenta uma análise comparativa entre os três nós sensores e o projetado neste trabalho. É interessante notar que o nó sensor projetado neste trabalho é o que possui menor custo.

Característica	Forest Fire Sensor	TmoteSky	WaspMote	Projetado
Transmissão	WiFi	WiFi, Bluetooth, etc.	WiFi, Zigbee, Bluetooth, etc.	WiFi, 3G, SigFox, etc.
Microcontrolador	ATMega2560	MSP430	ATMega1281	ATMega328p
Arquitetura	8 bits	16 bits	8 bits	8 bits
Velocidade	16MHz	8MHz	8MHz	16MHz
Memória de Programa	256kB	48kB	128kB	32kB
Memória de Dados	8kB	10kB	8kB	2kB
Memória de Armazenamento	2GB	1MB	2GB	1kB
Preço	R\$ 143,00	R\$ 169,60	R\$ 768,30	R\$92,00

**Tabela 2.2:** Comparaçāo entre os trabalhos relacionados e o nō projetado neste.

### 2.2.2 Disponibilizaçāo de Dados de Sensores

Há o projeto FIWARE Lab [Fiware2016], que trata-se de uma iniciativa sem fins lucrativos para disponibilizaçāo de dados de sensores de vários locais do mundo, onde indivíduos e empresas podem tanto publicar quanto utilizar essas informações para pesquisas e desenvolvimento de aplicações. Pode-se ver na figura nós enviando e recebendo dados para o FIWARE *Broker* utilizando diferentes protocolos para isso. Várias cidades já tornaram seus dados públicos através desse projeto. Particularmente, este projeto pode ser facilmente integrado a essa plataforma. A Figura 2.11 mostra um mapa com os países que contém nós enviando dados para o FIWARE Lab.



**Figura 2.11:** Diagrama esquematizando o FIWARE Lab [Fiware2016].

# Capítulo 3

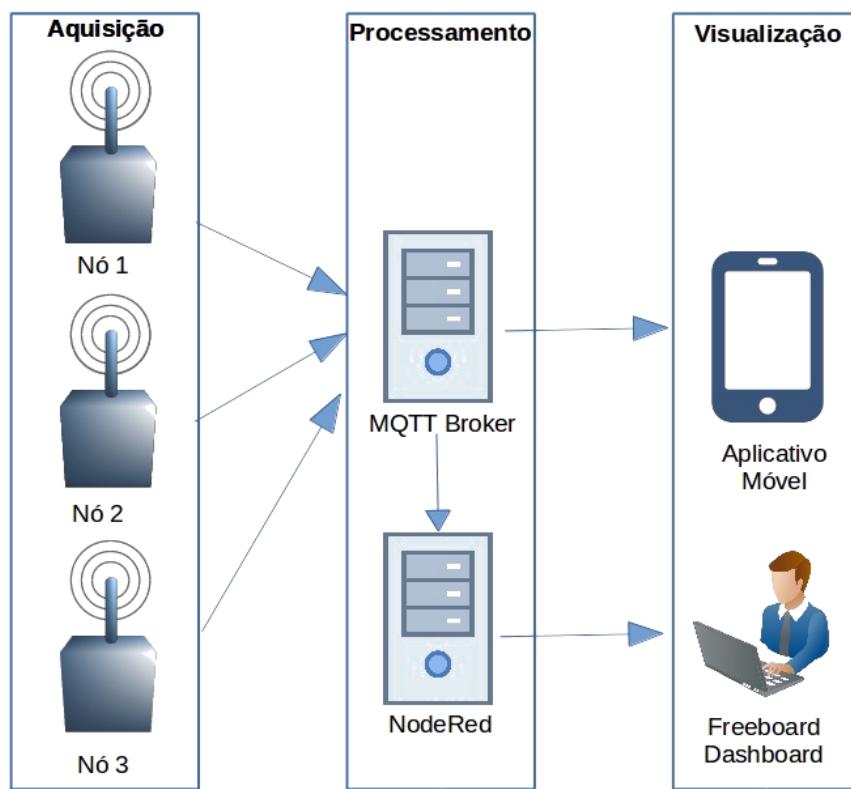
## A Arquitetura Projetada

A Figura 3.1 mostra uma visão geral da arquitetura de acordo com os itens abaixo, juntamente com o fluxo de informação.

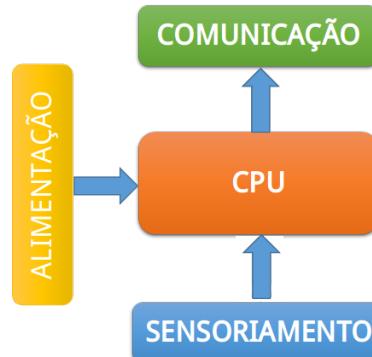
- Um ou mais nós sensores;
- Broker MQTT;
- Servidor com Node-RED instalado;
- Freeboard;
- Aplicativo móvel.

### 3.1 O Nó Sensor

O nó sensor é composto por quatro módulos distintos. A Figura 3.2 apresenta a visão em módulos do nó-sensor. Primeiramente, tem-se a CPU, onde todo o processamento, gerenciamento de memória e execução de programas acontece. Logo depois, vem o módulo de alimentação, responsável por fornecer energia ao equipamento. Existe também o módulo de comunicação que permite ao nó conectar-se a um roteador WiFi. Por fim, há o módulo de sensoriamento, que contém os sensores de concentração de gás, luminosidade, umidade e



**Figura 3.1:** Visão geral da arquitetura proposta pelo trabalho



**Figura 3.2:** Divisão do nô sensor em módulos.

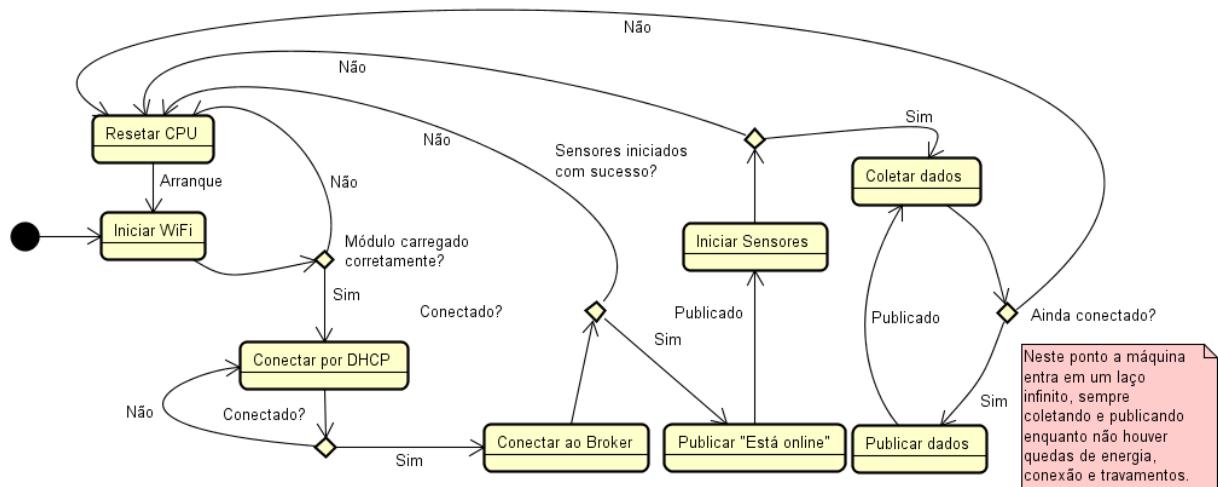
temperatura. O único custo econômico neste projeto foi gerado com as peças do nô sensor e são discriminados na Tabela 3.1.

Itens	Valor
Arduino UNO Rev3	R\$29,00
MQ-2	R\$9,50
DHT11	R\$7,50
LDR	R\$1,00
NodeMCU	R\$45,00
Total	R\$92,00

**Tabela 3.1:** Discriminação dos custos com o nô sensor.

### 3.1.1 Módulo CPU

O módulo CPU consiste numa placa Arduino UNO R3, que utiliza o microcontrolador ATMega 328p. Ele é responsável pela execução do programa, controlando as entradas e saídas digitais e analógicas. A Figura 3.3 mostra o diagrama que representa o software embarcado no microcontrolador. Basicamente, o sistema faz a inicialização do módulo WiFi e se conecta ao *broker*. Após isso, entra em laço onde dados são lidos dos sensores e enviados ao broker. Em caso de queda de conexão ou travamentos o software reinicia.



**Figura 3.3:** Fluxograma descrevendo o funcionamento do software embarcado na CPU.

### 3.1.2 Módulo de Alimentação

O módulo de alimentação é composto por duas entradas: uma *JACK*, de até 12V, e uma entrada USB de 5V. O sistema pode ser alimentado por uma delas. Neste trabalho, utilizou-se uma fonte de alimentação externa que reduz a tensão de 127V para 5V e contém

uma entrada USB que serve, além da alimentação, para conexão serial. O nó sensor pode facilmente ser alimentado por baterias.

### 3.1.3 Módulo de Comunicação

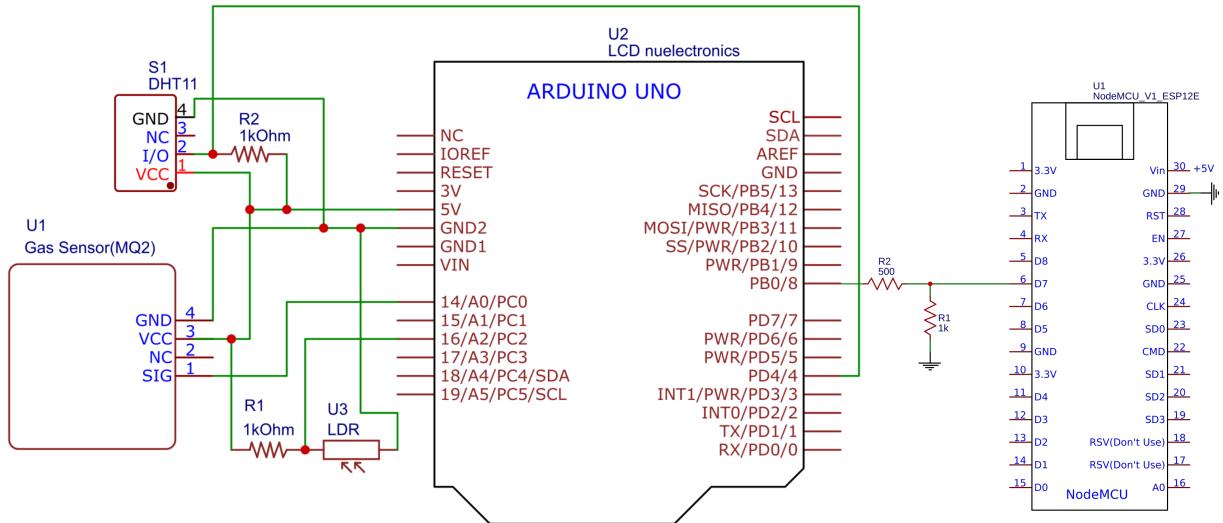
Para comunicação utilizou-se o NodeMCU para fazer a enviar os dados via MQTT. O Arduino faz a leitura dos sensores os envia ao NodeMCU por serial, que publica os dados em tópicos utilizando o QoS nível 0. O mesmo pode ser facilmente substituído por um *shield* Ethernet, 3G ou GSM para operação em campo, por exemplo.

### 3.1.4 Módulo de Sensoriamento

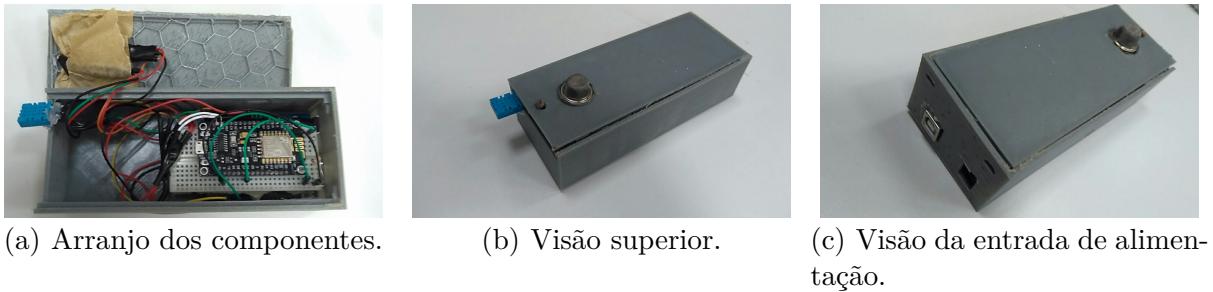
O módulo de sensoriamento é composto por três sensores: umidade e temperatura, luminosidade e concentração de gases. Essa camada pode ser expandida adicionando-se mais sensores utilizando as portas analógicas e digitais disponíveis, além de se fazer ajustes no software embarcado no Arduino.

### 3.1.5 Montagem

Primeiramente, a montagem foi realizada utilizando-se uma placa de prototipagem. Então, após verificar-se a corretude dos circuitos e o bom funcionamento do protótipo, confeccionou-se um gabinete utilizando uma impressora 3D, no qual instalou-se o projeto. A Figura 3.4 apresenta o esquema de montagem utilizado para o circuito. A Figura 3.5(a) mostra o gabinete impresso com os componentes eletrônicos arranjados e, por fim, a Figura 3.5(c) exibe o nó sensor finalizado.



**Figura 3.4:** Diagrama esquemático do circuito.



**Figura 3.5:** Gabinete confeccionado com auxílio de impressora 3D.

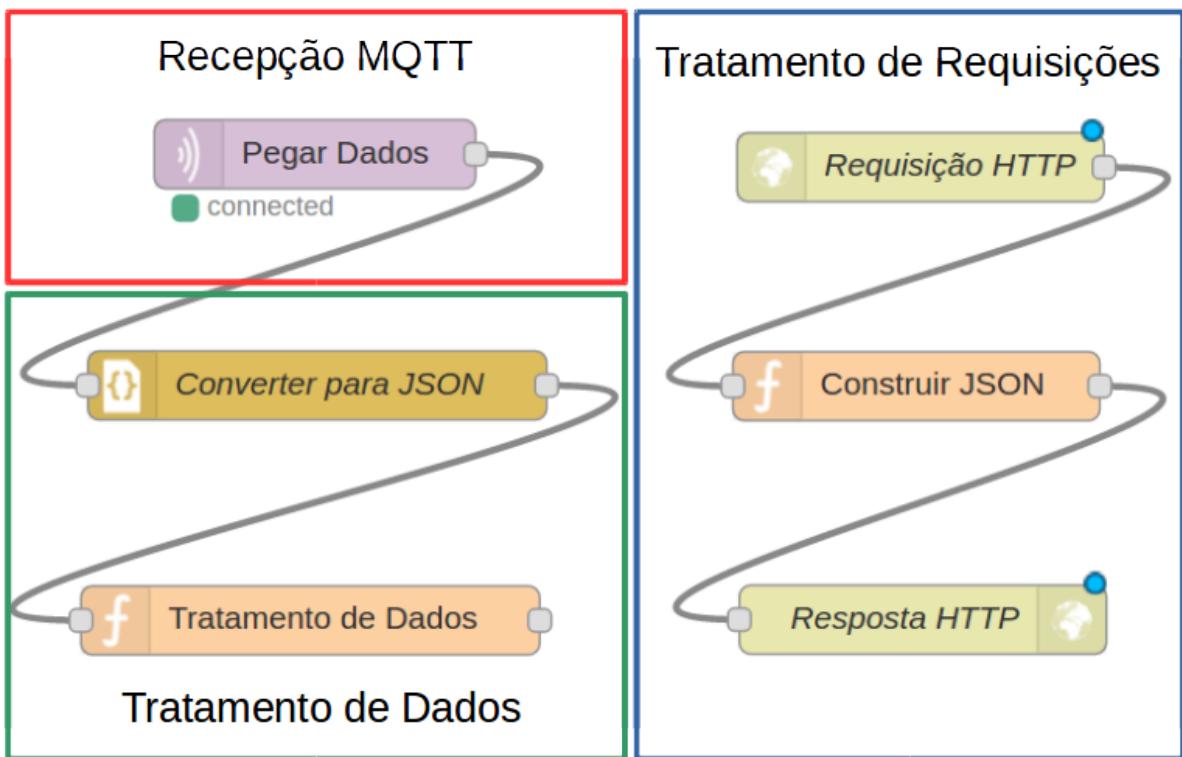
## 3.2 MQTT broker

O MQTT broker utilizado no projeto foi o Mosquitto 1.4.9 [Eclipse2006]. Foi disponibilizado um tópico por nó sensor onde se publica os dados dos sensores em uma cadeia de caracteres no formato JSON, e outro tópico para mensagens de *status* do sistema. São eles:

- /ocean/<nome\_nó\_sensor>/dados
- /ocean/<nome\_nó\_sensor>/debug;

### 3.3 Aplicação Node-RED

A Figura 3.6 mostra o programa criado com Node-RED, onde os dados dos sensores são recebidos nos blocos da “Recepção MQTT”, são convertidos para JSON em “Tratamento dos Dados” e disponibilizados para serem acessados via *GET* em “Tratamento de Requisições”, tornando-se um serviço WEB. Neste trabalho disponibilizou-se o endereço no formato `http://endereço:porta/dados` que apresenta uma *string* JSON no formato deste exemplo: `{"temperatura":23,"umidade":48,"gas":8.8,"luminosidade":2.03,"timestamp":z1492217284180}`.



**Figura 3.6:** Serviço Web feito usando Node-RED para disponibilizar dados.

### 3.4 *Dashboard Freeboard*

A Figura 3.7(a) mostra uma imagem dos painéis criados para o projeto na plataforma Freeboard, onde o usuário tem uma visão geral amigável em tempo real dos dados. Como é possível ver, criou-se quatro painéis com gráficos do tipo gauge, um para representar cada

um dos sensores. Não há um limite para adicionar elementos aos painéis, que podem ser de oito tipos diferentes, à saber: texto, gauge, *sparkline*, *pointer*, *picture*, indicator light, mapa do Google e HTML. Tal ferramenta permite que qualquer usuário possa fazer uso dos dados disponibilizados pelo Node-RED para sua aplicação específica.

### 3.5 Aplicativo móvel

A Figura 3.7(b) mostra a tela do aplicativo desenvolvido utilizando para Android utilizando a biblioteca Paho [Eclipse2012], que assina os tópicos MQTT utilizados pelo nó sensor publica, recebendo os dados assincronamente e disparando-os no visor. É composto de uma única tela, que foi dividida em quatro partes, cada uma contendo um ícone correspondente ao dado enviado pelo sensor, tendo ao lado o nome da variável e o seu valor, juntamente com as respectivas unidades de medida. Tem a função de painel de visualização de dados, semelhante ao Freeboard, onde o usuário pode analisar os dados amigavelmente. A vantagem dessa alternativa é que mostra que a arquitetura, ao usar tecnologias populares, torna fácil a integração de novas aplicações. Vale ressaltar que este aplicativo não utiliza dados formatados pelo Node-RED, mas sim os recebidos pelo *broker*. Todavia, não há impedimentos para que um celular possa consumir os dados formatados.

As Figuras 3.8 e 3.9 apresentam o diagrama de classes e sequência, respectivamente, do aplicativo. A classe principal é a MainActivity, que herda AppCompatActivity. Classes filhas de *Activity* representam telas, manipulando seu visual e guardando sua lógica [Lecheta2015]. Neste aplicativo, ele relaciona-se com ImageView e TextView, ambas classes do Android SDK responsáveis por representar objetos de exibição de imagens e textos na tela, respectivamente, além das classes MqttAndroidClient e IMqttToken, da biblioteca Paho. MainActivity inicializa o *layout*, faz a conexão MQTT, assina o tópico da arquitetura e recebe os dados, atualizando os elementos da tela. Isto se repete em um laço infinito.

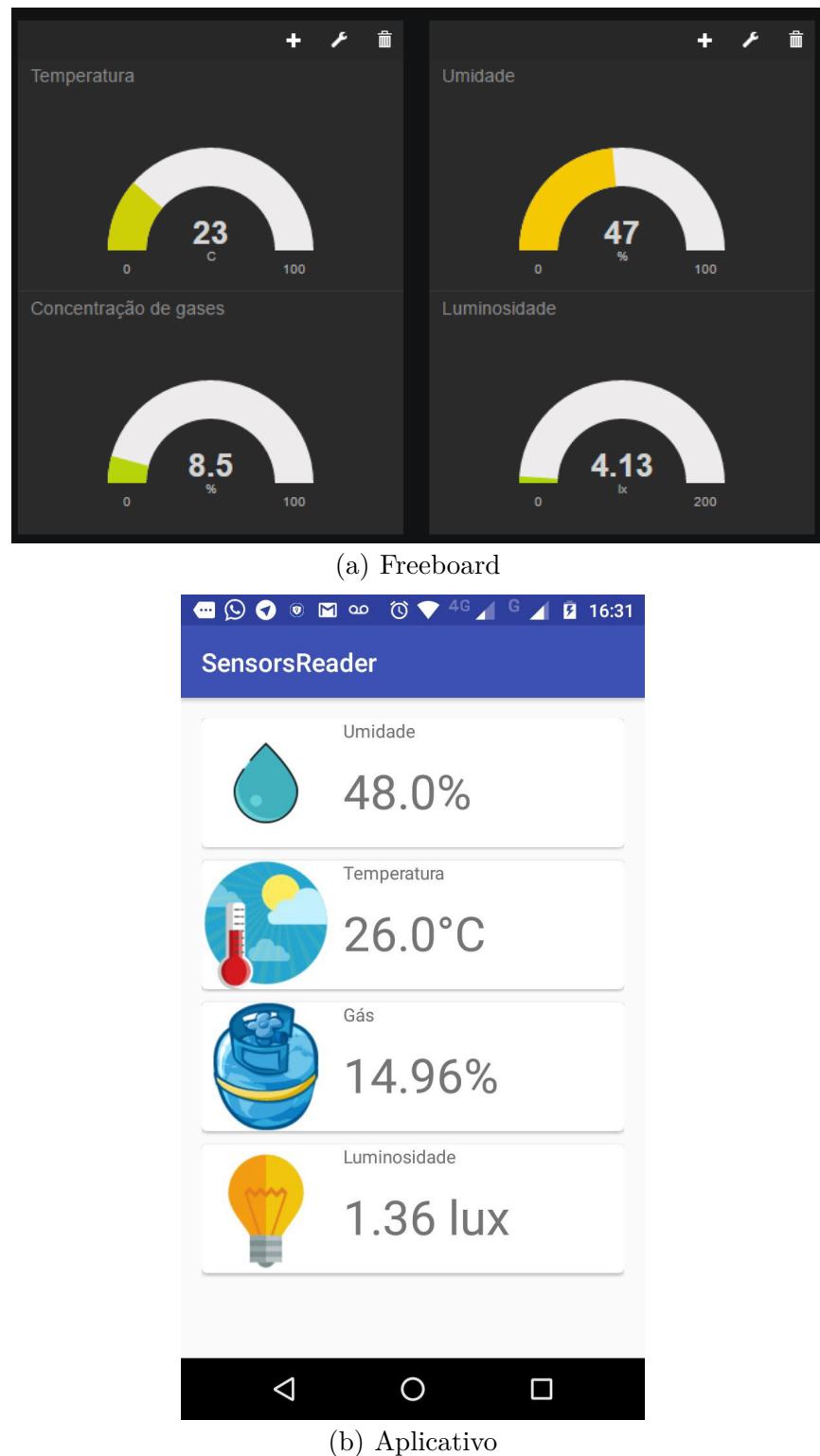


Figura 3.7: Interfaces WEB e móvel desenvolvidas para visualização amigável e em tempo real dos dados.

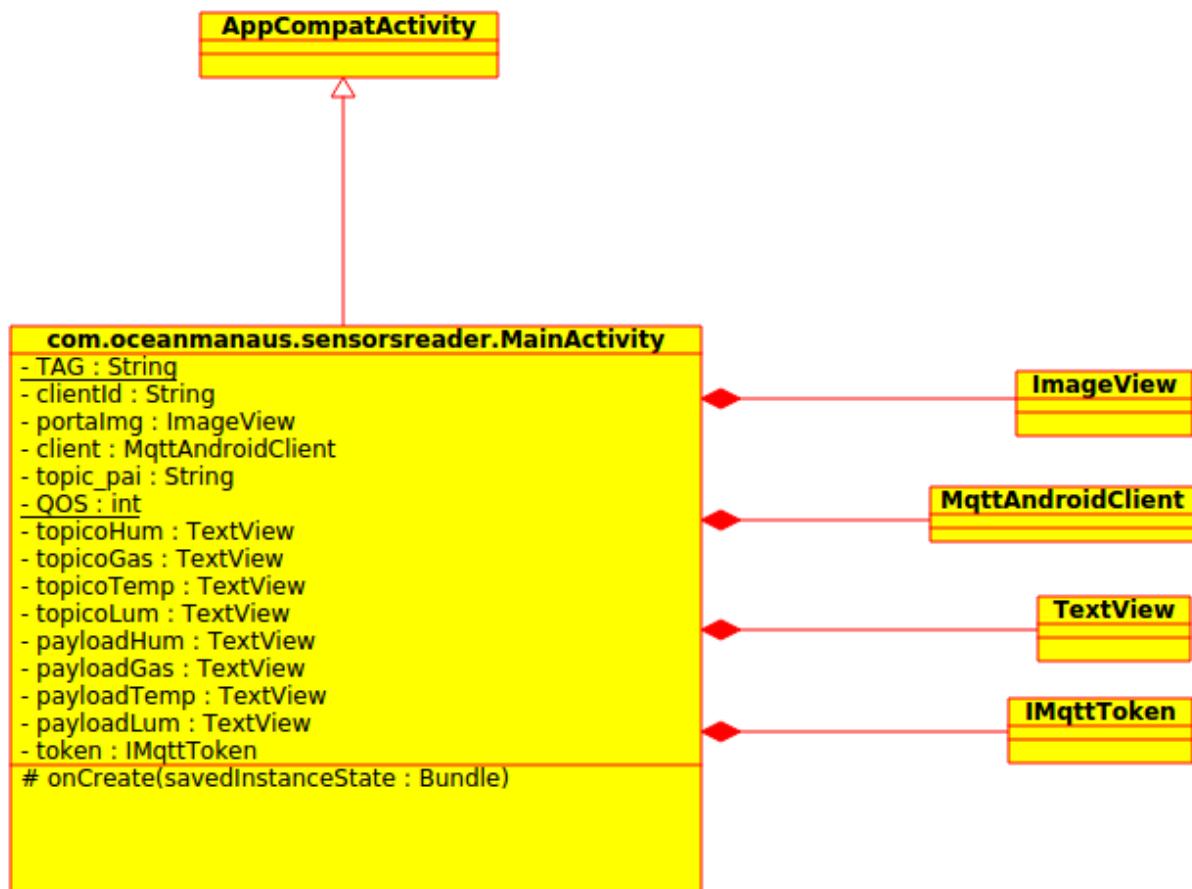
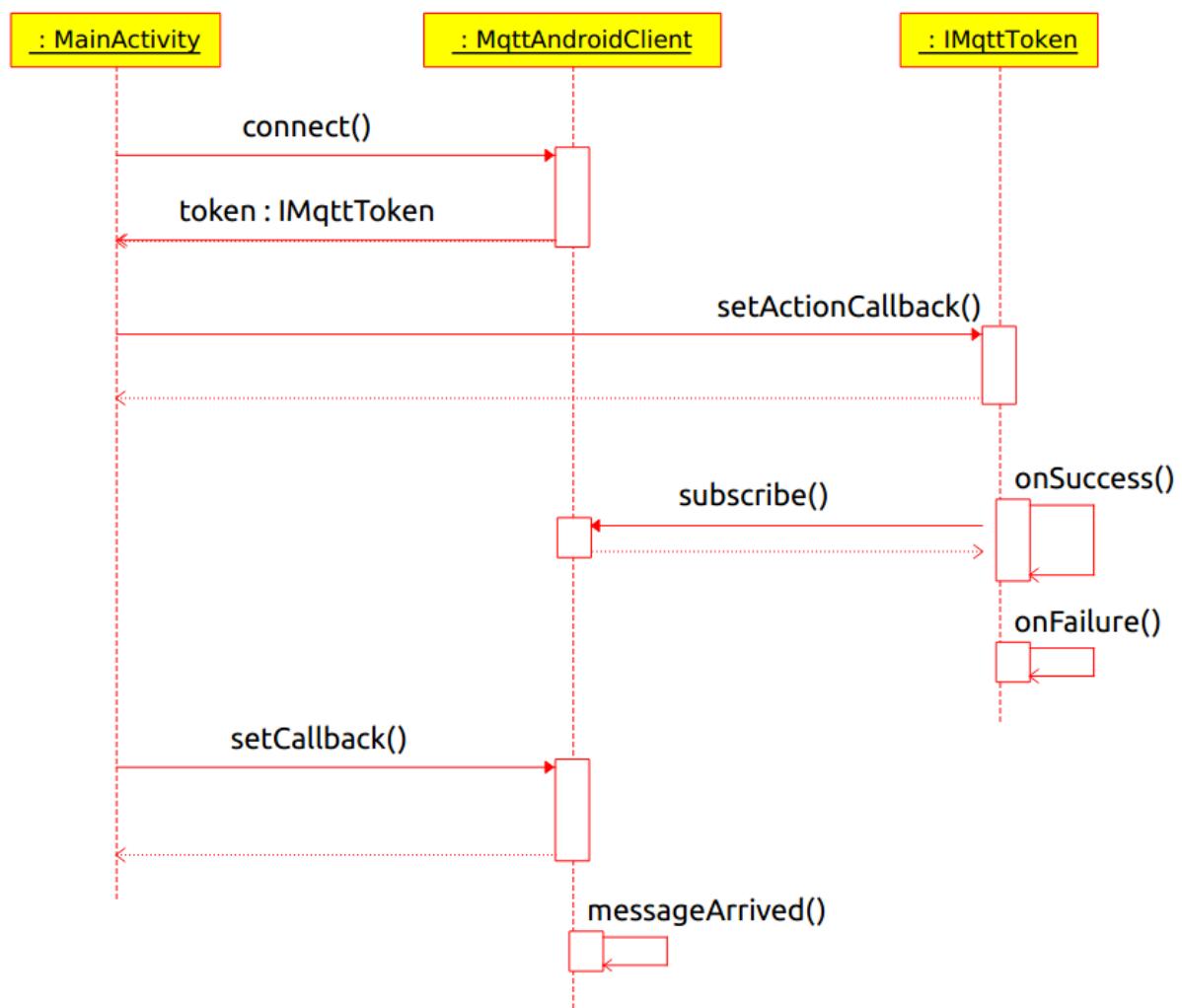


Figura 3.8: Diagrama de classes do aplicativo.



**Figura 3.9:** Diagrama de sequência do aplicativo.

# **Capítulo 4**

## **Testes e Resultados**

### **4.1 Estudos de Casos**

Para testar a arquitetura, dois tipos de testes foram realizados. No primeiro, expôs-se o nó sensor a um ambiente para avaliação em campo, considerando todos os sensores ao mesmo tempo, ao passo que no segundo verificou-se isoladamente cada sensor, com experimentos controlados causando situações para que verificar as respostas dos mesmos a essas variações.

#### **4.1.1 Exposição em Ambiente**

Neste teste, uma unidade de nó sensor foi implantada num ambiente residencial, em um lugar aberto e coberto, para protegê-lo de possíveis intempéries, e foram analisados dados em um período começando as 21h03min do primeiro dia do teste a 4h4min do terceiro dia, totalizando 111.671,762s, exposto a variações de incidência solar, temperatura, umidade e concentração de gases. As Figuras 4.1 a 4.4 apresentam os gráficos gerados durante este período de observação, onde verifica-se que os sensores detectaram mudanças no ambiente e os gráficos correlacionam-se entre si. Um exemplo disso, é o fato de que por volta dos 60.000s a temperatura e a luminosidade estão em seus ápices, ao passo que a umidade encontra-se em seu vale. Isso deveu-se a um momento de incidência solar direta, onde o sol entrou pela janela por volta de 12h50min do segundo dia. O sensor de gás detectou

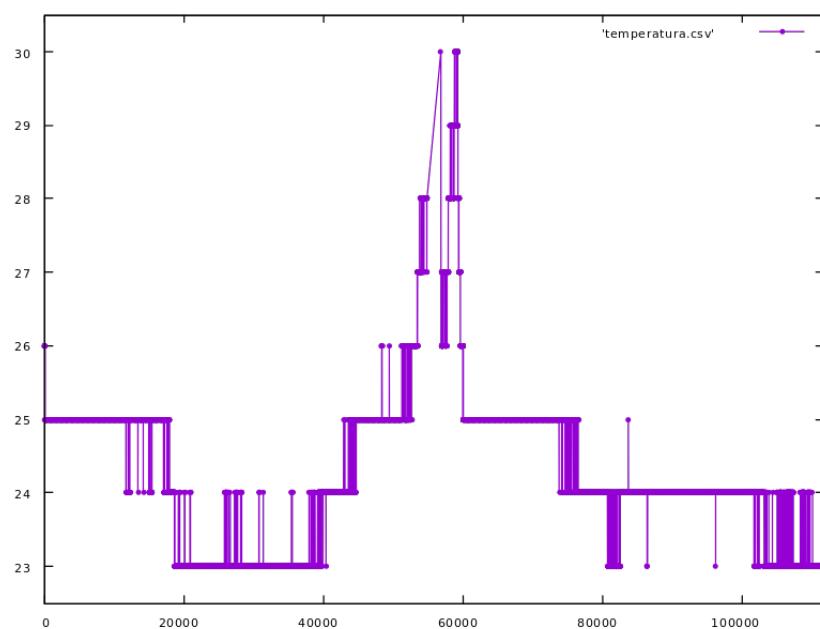
somente níveis gasosos residuais, provavelmente devido a poluição e ruídos em geral.

Durante o período testado, o nó sensor transmitiu 19.553 mensagens, apresentando um *delay* médio de aproximadamente 5,71s entre cada envio. No entanto, para as aplicações as quais este trabalho se propõe isto provavelmente não representaria um problema, uma vez que geralmente não se exige um grau de precisão maior do que este nesses tipos de aplicações. O aplicativo fez seus recebimentos assincronamente, apresentando os dados conforme o proposto. O JSON disponível pelo NodeRED estava constantemente atualizado, assim como os painéis do Freeboard.

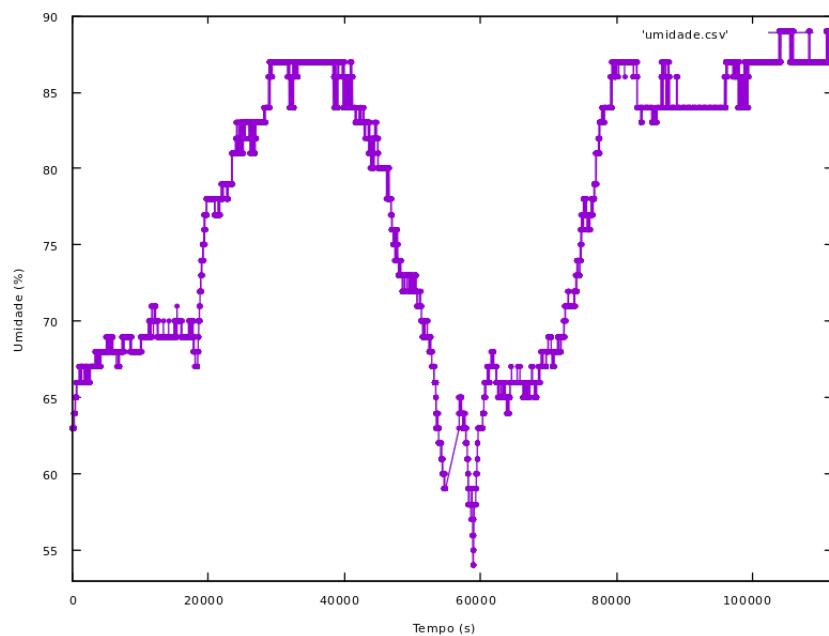
#### 4.1.2 Experiments Isolados

Dentro de um ambiente fechado, fez-se uma série de experimentos para verificar se os sensores estão respondendo adequadamente aos estímulos do ambiente. A Figura 4.5 mostra a observação dos dados pelos módulos clientes. O primeiro deles, utilizou-se um ferro de solda e estanho, gerando uma fumaça que estimulou o sensor de concentração de gases, causando um aumento do valor detectado pelo mesmo de 11,82%. Em seguida, a Figura 4.6 mostra-se o segundo teste, onde colocou-se o ferro de solda aquecido em um suporte em forma de mola, o que fez com a região próxima fosse aquecida, e aproximou-se o sensor de temperatura do mesmo, notando-se um aumento de 5°C. Logo depois, a Figura 4.7 mostra um experimento utilizando a lanterna de um celular focada próximo ao sensor de luminosidade, onde notou-se um aumento de 6,09lx. Por fim, a Figura 4.8 retrata o último experimento, onde um pano úmido foi colocado no sensor de umidade, e verificou-se um aumento de 15% da umidade relativa do ar.

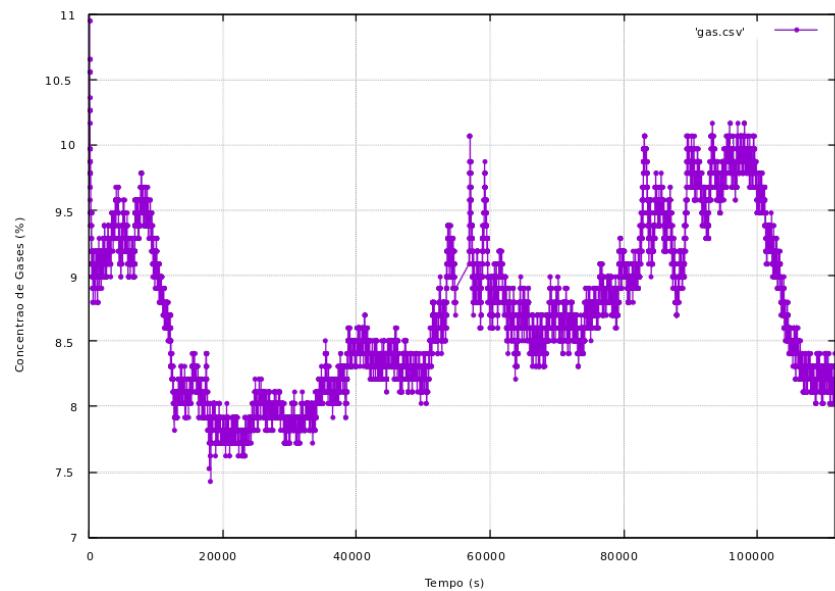
Nota-se que, embora cada experimento tivesse o objetivo e influenciar um sensor de cada vez, outros eram influenciados em menor escala, o que é esperado, dado a proximidade dos sensores. Esses testes revelam que a arquitetura responde suficientemente bem dentro da faixa de precisão exigida por aplicações de cidades inteligentes.



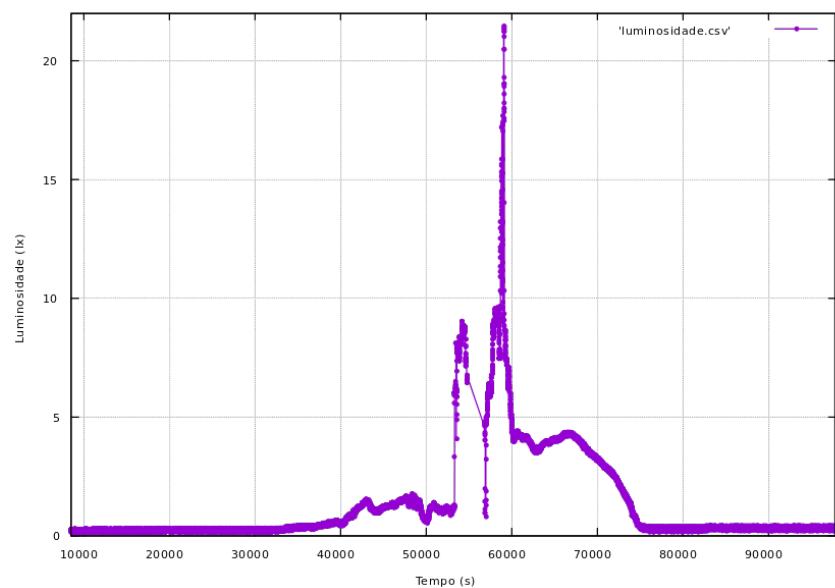
**Figura 4.1:** Gráfico da temperatura.



**Figura 4.2:** Gráfico da umidade.



**Figura 4.3:** Gráfico de concentração de gases inflamáveis e fumaça.



**Figura 4.4:** Gráfico da luminosidade.

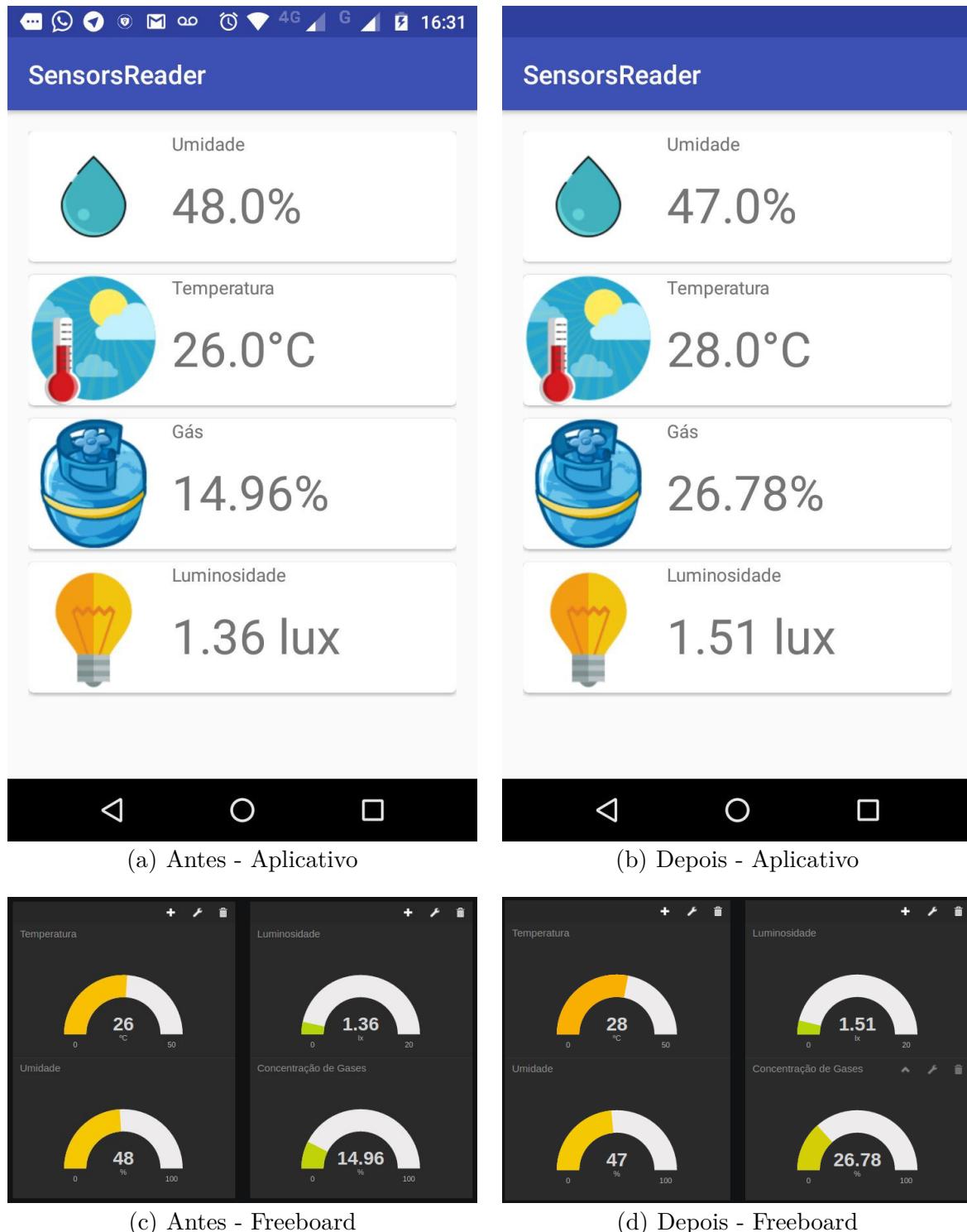


Figura 4.5: Visão no aplicativo e Freeboard da variação da concentração de gases detectada pelo nó sensor.

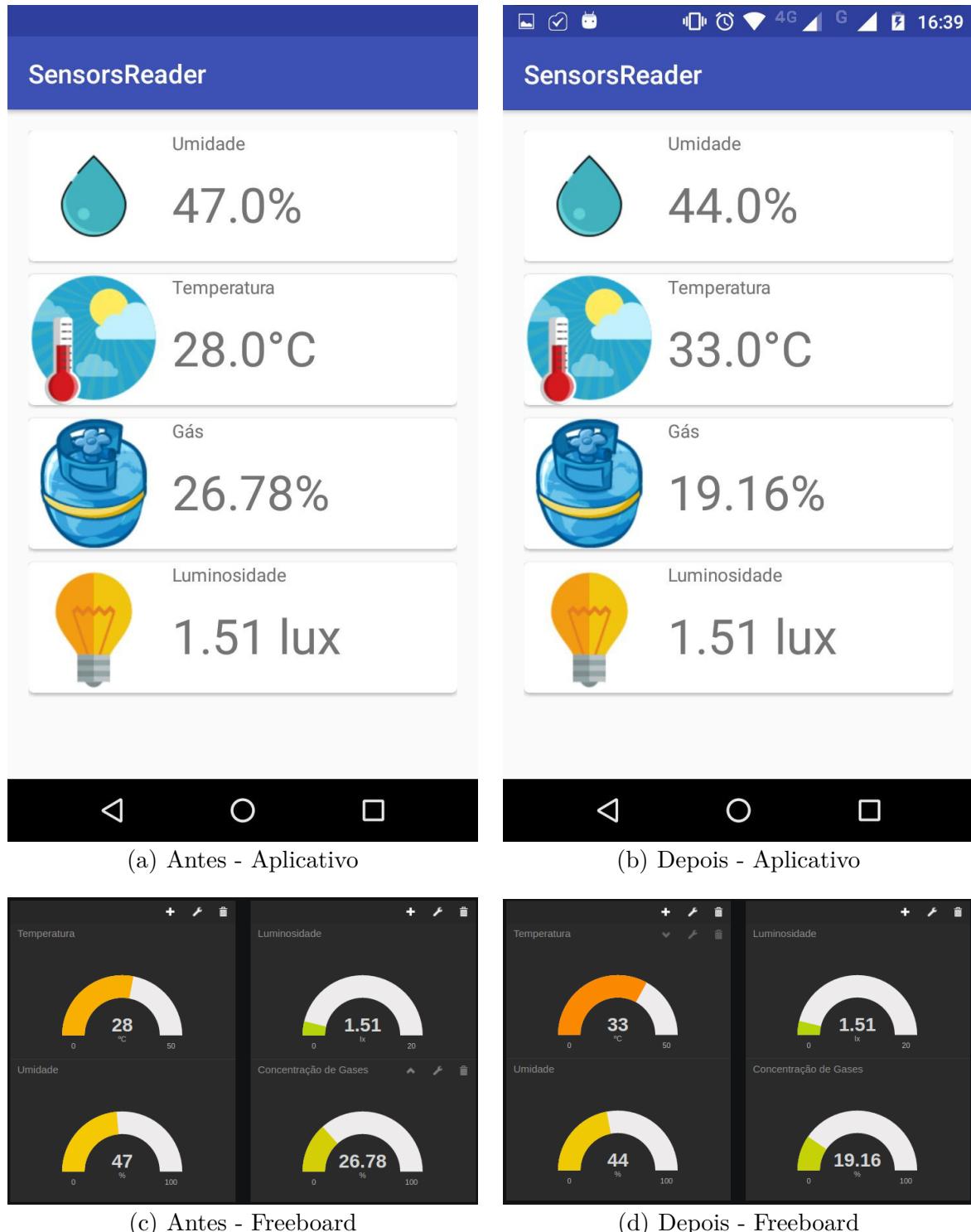


Figura 4.6: Visão no aplicativo e Freeboard da variação de temperatura detectada pelo nó sensor.

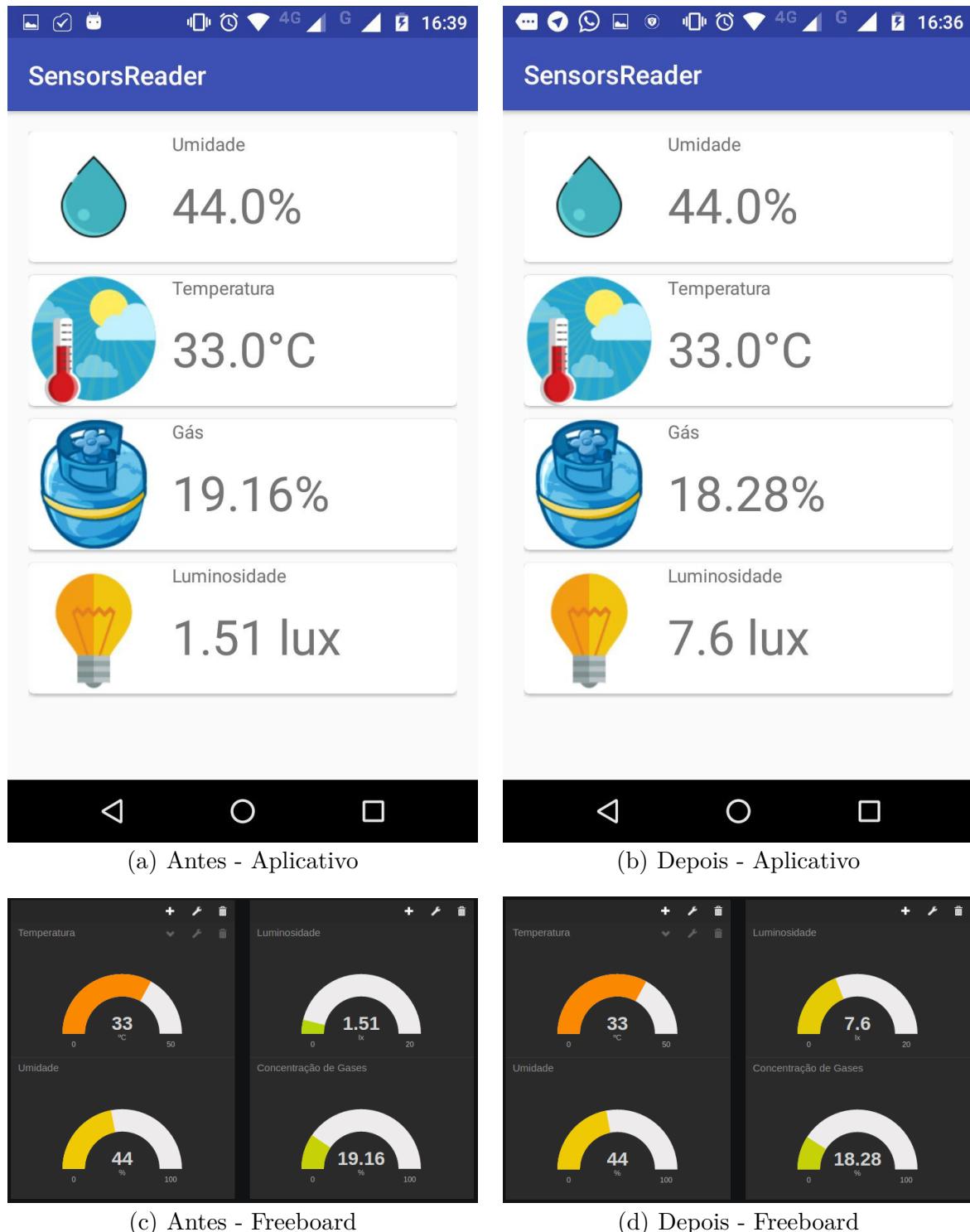


Figura 4.7: Visão no aplicativo e Freeboard da variação de luminosidade detectada pelo nó sensor.

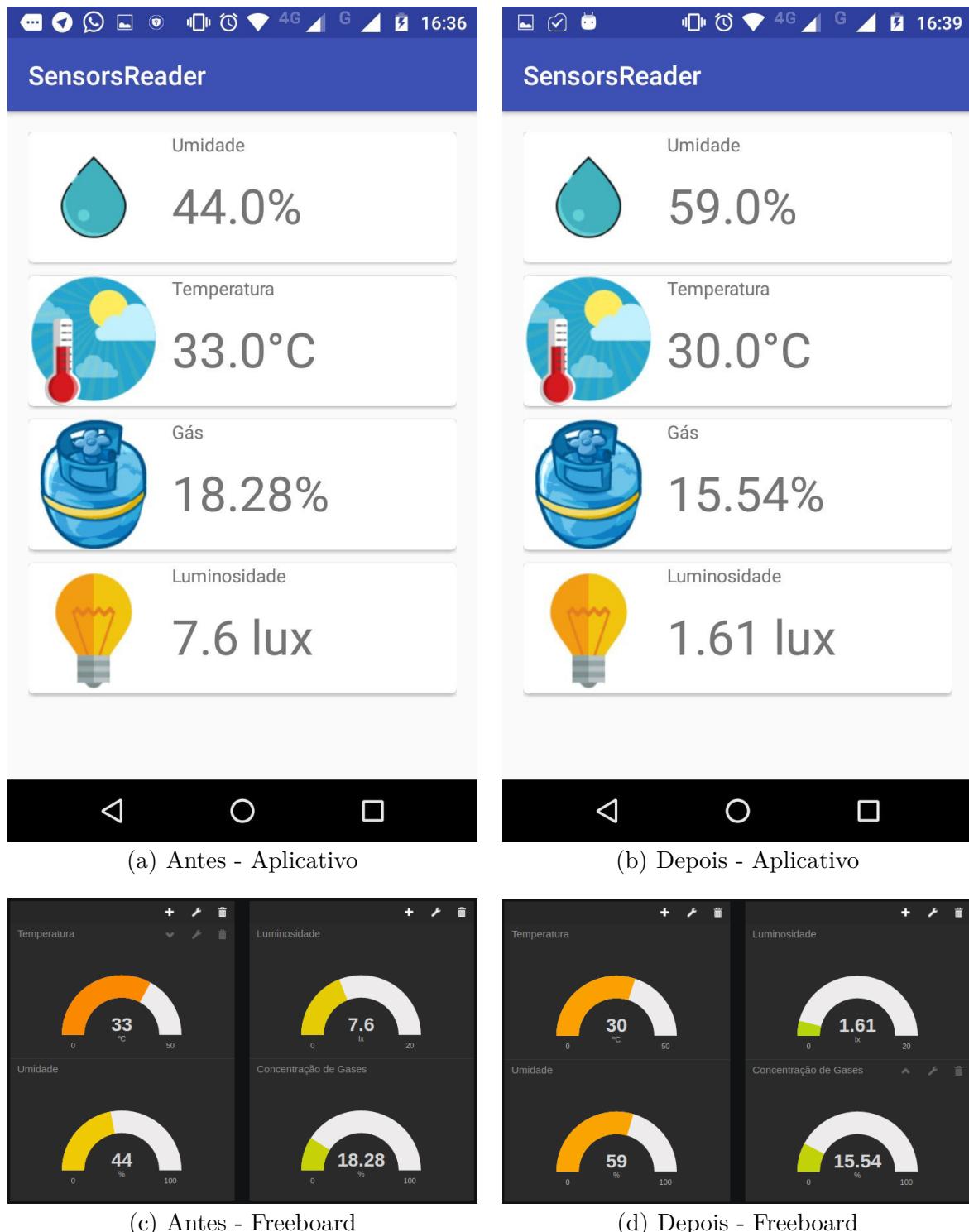


Figura 4.8: Visão no aplicativo e Freeboard da variação de umidade detectada pelo nó sensor.

## 4.2 Resultados

Os resultados produzidos por este trabalho incluem:

- Um nó sensor com quatro sensores (umidade, temperatura, luminosidade e gases) conectável à Internet via WiFi, juntamente com seu código-fonte;
- Um serviço Web que fornece os dados coletados em tempo real no formato JSON;
- Um aplicativo móvel para Android com um *dashboard* para acompanhamento em tempo real dos dados juntamente com seu código-fonte;
- Um *dashboard* Web Freeboard para visualização *desktop* em tempo real;

Todos estes resultados juntos integram um sistema escalável, uma vez que ele suporta um grande número de nós sensores, limitado pelo número máximo de clientes do *Broker*, e um grande número de serviços consumindo a API disponibilizada, também limitado pelo número máximo de clientes suportados pelo servidor Web. O funcionamento do sistema pode ser visto no vídeo anexo a este trabalho.

# Capítulo 5

## Considerações Finais

### 5.1 Conclusão

Com o crescimento das cidades uma série de problemas surgiu, sendo um dos principais deles a poluição. Neste cenário, o IoT oferece recursos para minimizar esse problema, fazendo uso de tecnologias de sistemas embarcados conectados à Internet. Uma aplicação muito útil é a criação de arquiteturas extensíveis que possam monitorar dados do ambiente que sejam de interesse e disponibilizá-los através de um serviço Web e em painéis de visualização amigável.

Este trabalho apresentou uma solução de arquitetura de IoT para monitoramento de ambientes urbanos, de baixo custo, com tecnologias abertas e simples, podendo ter um grande número de nós sensores, dois *dashboards* para visualização sendo um Web e outro móvel, além de um sistema que disponibiliza os dados em JSON, permitindo que qualquer aplicação possa fazer requisições e utilizar os dados para criar novas funcionalidades, funcionando como extensões. Isto torna o sistema versátil. É uma arquitetura de baixa custo e de fácil implementação, podendo ser aplicada em infraestrutura pública e estendida a diversas aplicações de interesse. Além disso, é importante enfatizar a alta escalabilidade do sistema, uma vez que os nós sensores podem estar em diferentes pontos de acesso a Internet, o *broker* aceita vários clientes de acordo com o número máximo definido em suas configurações, e várias aplicações podem consumir seus dados, abrindo margem para todo

um ecossistema de aplicações e podendo fazer uso, inclusive, de atuadores. É possível criar uma aplicação que através do serviço Web aguarde a concentração de gases chegar a um nível crítico e dispare um alarme, por exemplo.

De acordo com os testes realizados, a arquitetura elaborada neste trabalho conseguiu cumprir os objetivos definidos na proposta e dentro dos prazos definidos no cronograma. Alguns dos resultados foram publicados em Dantas and Figueiredo2017.

## 5.2 Dificuldades Encontradas

Ao longo do processo de desenvolvimento da arquitetura houve algumas dificuldades, porém todas foram contornadas, o que permitiu com que o sistema chegasse ao final. Dentre os principais empecilhos, os principais foram:

- De início, ao invés de utilizar-se o NodeMCU como módulo de comunicação, utilizou o CC3000 da Sparkfun, um módulo WiFi. No entanto, ele apresentou um série de instabilidades que estavam minando a qualidade do projeto;
- Houve um esforço para encontrar o modelo de gabinete correto para a impressão em 3D;
- A plataforma online do Freeboard apresentou uma instabilidade relativa ao acesso de serviços de domínios diferentes, mas isso foi resolvido pela equipe técnica da empresa;

## 5.3 Trabalhos futuros

A arquitetura desenvolvida neste trabalho pode ser ampliada e resulta em diversos trabalhos possíveis. Uma possibilidade seria a de construir vários nós, instalá-los em uma localidade, tais como um *campus*, e utilizar atuadores que tivessem funcionamento condicionado aos dados obtidos, utilizando aprendizagem de máquina ao reconhecer padrões em variações de temperatura, por exemplo. Outro possível trabalho seria instalar vários nós em vias estratégicas e medir a poluição do ar em um período, permitindo que a Prefeitura soubesse onde ficam as áreas mais estratégicas para tomar medidas de conscientização e

redução da poluição, diminuindo ilhas de calor. Além disso, no modelo atual para alterar a rede WiFi a qual deseja-se conectar é necessário alterar o código-fonte do NodeMCU e recompilá-lo, o que é um empecilho. Como o ESP8266 pode atuar como ponto de acesso, é possível criar um modo de configuração, onde um dispositivo conectar-se-ia ao seu roteador embutido e definiria o nome e a senha da rede a qual espera-se conectar, contornando o problema. Por fim, poderia-se criar uma plataforma online onde o usuário selecionaria numa lista os sensores que deseja utilizar e em que portas pretende conectá-lo, gerando um *firmware* online, que poderia ser baixado e instalado, permitindo que um usuário sem conhecimentos em programação possa utilizá-lo facilmente.

# Referências Bibliográficas

- [Advanticsys2015] Advanticsys (2015). <http://www.advanticsys.com/shop/mtmcm5000msp-p-14.html>. Acesso em 22/10/2016.
- [Android2017] Android (2017). The android source code. <https://source.android.com/source/>. Acesso em 04/17/2017.
- [Arduino2005] Arduino (2005). <https://www.arduino.cc/>. Acesso em 22/10/2016.
- [Arduino e Cia2016] Arduino e Cia (2016). Arduino e cia: Web server com o módulo esp8266 nodemcu e dht22. <http://www.arduinoecia.com.br/2016/02/web-server-esp8266-nodemcu-dht22.html>. Acessado em 04/17/2017.
- [Atzori et al.2010] Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer networks*, 54(15):2787–2805.
- [Banks and Gupta2014] Banks, A. and Gupta, R. (2014). MQTT Version 3.1.1. <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>. Acesso em 22/10/2016.
- [BRASIL2016] BRASIL (2016). Ministério abre processo de seleção para o minha cidade inteligente. <http://www2.mc.gov.br/sala-de-imprensa/todas-as-noticias/institucionais/40119-ministerio-abre-selecao-para-o-minha-cidade-inteligente>. Acesso em 05/09/2016.
- [Cellan-Jones2012] Cellan-Jones (2012). A 15 pound computer to inspire young programmers. [http://www.bbc.co.uk/blogs/thereporters/rorycellanjones/2011/05/a\\_15\\_computer\\_to\\_inspire\\_your](http://www.bbc.co.uk/blogs/thereporters/rorycellanjones/2011/05/a_15_computer_to_inspire_your)

- [CH2016] CH (2016). Wasp mote - wireless sensor networks open source platform. <https://www.cooking-hacks.com/documentation/tutorials/wasp mote/>. Acessado em 04/21/2017.
- [D-ROBOTICS2010] D-ROBOTICS (2010). DHT11 Humidity & Temperature Sensor. <http://www.micropik.com/PDF/dht11.pdf>. Acesso em 22/10/2016.
- [Dantas and Figueiredo2017] Dantas, D. and Figueiredo, C. M. (2017). Arquitetura de iot para cidades inteligentes. In *ENCOSIS 2017*.
- [Eclipse2006] Eclipse (2006). <https://mosquitto.org/>. Acesso em 22/10/2016.
- [Eclipse2012] Eclipse (2012). <https://eclipse.org/paho/>. Acesso em 22/10/2016.
- [EMANT2004] EMANT (2004). <http://emant.com/316002.page>. Acesso em 22/10/2016.
- [Espressif2014] Espressif (2014). <https://espressif.com/>. Acesso em 22/10/2016.
- [Fiware2016] Fiware (2016). <https://www.fiware.org/lab/>. Acesso em 22/10/2016.
- [Gartner, Inc.2015] Gartner, Inc. (2015). Gartner says worldwide smartphone sales recorded slowest growth rate since 2013. <http://www.gartner.com/newsroom/id/3115517>. Acesso em 04/17/2017.
- [HiveMQ2017] HiveMQ (2017?). Mqtt essentials part 6: Quality of service levels. <http://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels>. Acesso em 04/07/2017.
- [Hunkeler et al.2008] Hunkeler, U., Truong, H. L., and Stanford-Clark, A. (2008). Mqtts—a publish/subscribe protocol for wireless sensor networks. In *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*, pages 791–798. IEEE.
- [ISO/IEC20922 2016] ISO/IEC20922 (2016). Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1.
- [JS Foundation2013] JS Foundation (2013). Node-red : About. <http://nodered.org/about/>. Acessado em 04/17/2017.

- [Lawler2012] Lawler (2012). Raspberry Pi credit-card sized Linux PCs are on sale now, \$25 Model A gets a RAM bump. <https://www.engadget.com/2012/02/29/raspberry-pi-credit-card-sized-linux-pcs-are-on-sale-now-25-mo/>. Acesso em 22/10/2016.
- [Lecheta2015] Lecheta, R. (2015). *Google Android 4<sup>a</sup> edição: Aprenda a criar aplicações para dispositivos móveis com o Android SDK*. Novatec Editora.
- [Lee et al.2014] Lee, J. H., Hancock, M. G., and Hu, M.-C. (2014). Towards an effective framework for building smart cities: Lessons from seoul and san francisco. *Technological Forecasting and Social Change*, 89:80–99.
- [Liberium2012] Liberium (2012). <http://www.libelium.com/products/waspmove>. Acesso em 22/10/2016.
- [Marchal et al.2012] Marchal, V., Dellink, R., van Vuuren, D., Clapp, C., Château, J., Lanzi, E., Magné, B., and van Vliet, J. (2012). OECD environmental outlook to 2050: the consequences of inaction.
- [NTU2015] NTU (2015). Android basics and user interfaces. <https://www3.ntu.edu.sg/home/ehchua/programming/arduino/arduino.html>. Acesso em 04/19/2017.
- [RPF2012] RPF (2012). <https://www.raspberrypi.org/>. Acesso em 22/10/2016.
- [SANDBOX ELECTRONICS2016] SANDBOX ELECTRONICS (2016). MQ-2 Smoke/LPG/CO Gas Sensor Module. <http://sandboxelectronics.com/?p=165>. Acesso em 22/10/2016.
- [Santos and Figueiredo2016] Santos and Figueiredo (2016). Rede de Sensores Sem Fio Para Monitoramento de Dados Ambientais: Projeto de Nô Sensor de Baixo Custo. *Encontro Regional de Computação e Sistemas de Informação*.
- [Shelby et al.2014] Shelby, Z., Hartke, K., and Bormann, C. (2014). The constrained application protocol (coap). RFC 7252, RFC Editor. <http://www.rfc-editor.org/rfc/rfc7252.txt>.

- [SNM2008] SNM (2008). The Sensor Network Museum : Projects - Tmote Sky browse.  
<http://www.snm.ethz.ch/Projects/TmoteSky>. Acesso em 04/21/2017).
- [Xia et al.2012] Xia, F., Yang, L. T., Wang, L., and Vinel, A. (2012). Internet of things.  
*International Journal of Communication Systems*, 25(9):1101.

# Apêndices

# Apêndice A

## Códigos-fonte

### A.1 Módulo CPU (Arduino)

```
*****  
Este software      instalado no Arduino. Ele l  
os sensores, cria uma string JSON com os dados  
e os envia por serial ao NodeMCU.  
*****/  
  
#include <DHT.h>  
#include <SoftwareSerial.h>  
  
#define DHTTYPE DHT11    // DHT 11  
#define DHTPIN 4          // Pino digital onde leremos os dados do DHT  
  
#define GAS_PIN 0  
#define LUM_PIN 2  
  
/* Essa formula poder ser melhorada.  
Aqui se pega a tensao da porta analogica */
```

```
float pegaTensao(int porta) {
    int sensorValue = analogRead(porta);
    return sensorValue * (5.0 / 1023.0);
}

// Mede a luminosidade do ambiente em LUX.
// R = resistencia do divisor de tensao em KOhms
float pegaLuminosidade (int porta, float r) {
    float tensao = pegaTensao(porta);
    return ((2500 / tensao) - 500) / r;
}

// A tensao na porta aumenta linearmente em relacao a
// concentracao de gas. Assim podemos medir em porcentagem.
float pegaGas (int porta) {
    float tensao = pegaTensao(porta);
    return (tensao / 5) * 100;
}

DHT dht(DHTPIN, DHTTYPE);

int vez;

SoftwareSerial mySerial(7, 6); // RX, TX

void setup(void)
{
    vez = 0;
    Serial.begin(57600);
```

```
dht.begin();  
  
Serial.println(F("Olá, "));  
mySerial.begin(4800);  
}  
  
void loop(void) {  
  
    // importante deixarmos um curto delay entre as leituras  
    // de sensores e também entre as publicações  
  
    char strTemp[10];  
    char strHum[10];  
    float h = dht.readHumidity();  
    delay(50);  
    float t = dht.readTemperature();  
    Serial.print("Vez: "); Serial.println(vez++);  
    Serial.print("Temperatura: "); Serial.println(t);  
    Serial.print("Hum: "); Serial.println(h);  
    if (isnan(t) || isnan(h))  
    {  
        Serial.println("Faltou ao ler o sensor DHT");  
    }  
    char gas_buf[10];  
    char lum_buf[10];  
    float nivelGas = pegaGas(GAS_PIN);  
    dtostrf(nivelGas, 5, 2, gas_buf);  
    float nivelLuz = pegaLuminosidade(LUM_PIN, 10);  
    dtostrf(nivelLuz, 5, 2, lum_buf);
```

```

dtosstrf(t, 5, 2, strTemp);
dtosstrf(h, 5, 2, strHum);
char v[10];
sprintf(v, "Vez:%d", vez);

char json[100];

sprintf(json, "{\"temperatura\":%s,\"umidade\":%s,\"
\"gas\":%s,\"luminosidade\":%s}\n",
        strTemp, strHum, gas_buf, lum_buf, vez);

mySerial.write(json);
Serial.println(json);

//while (mySerial.available()) mySerial.read();
Serial.print("AvalFim:");
Serial.println(mySerial.available());

delay(5000);
}

```

**Código A.1:** Código-fonte destinado ao Arduino.

## A.2 Módulo de Comunicação (NodeMCU)

```
*****
Este software instalado no NodeMCU. Ele recebe
a string JSON do Arduino via serial e publica
em um t pico MQTT.
*****
```

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
```

```
#include <PubSubClient.h>
#include <SoftwareSerial.h>

#define TX D7
#define RX D6
#define BROKER "iot.eclipse.org"
#define PORTA 1883

char mySSID [20], f_ssid [50] = "Ocean\u2014Convidado",
f_pwd [50] = "ocean2222";
WiFiClient wclient;
PubSubClient client(wclient);

SoftwareSerial mySerial(RX, TX); // RX, TX

void setup ( void ) {
    pinMode(LED_BUILTIN, OUTPUT);
    WiFi.mode(WIFI_STA);
    Serial.begin(57600);
    while (!Serial) {
        ;
    }
    Serial.println("Ol\u00f5 sou\u2014o\u2014NodeMCU!");
    mySerial.begin(4800);
```

```
mySerial.setTimeout(5000);

client.setServer(BROKER, PORTA);

}

using namespace std;

void loop (void) {

    if (WiFi.status() != WL_CONNECTED) {
        Serial.print("Connecting\u2014to\u2014");
        Serial.print(f_ssid);
        Serial.println("... ");
        WiFi.hostname(mySSID);
        WiFi.begin(f_ssid, f_pwd);

        if (WiFi.waitForConnectResult() != WL_CONNECTED) {
            Serial.print(".");
            return;
        }
        Serial.println();
        Serial.println("WiFi\u2014connected");
        Serial.print ("IP\u2014address:\u2014");
        Serial.println ( WiFi.localIP() );
    }

    if (WiFi.status() == WL_CONNECTED) {

        while (!client.connected()) {
```

```
String c = (String("no") + String(ESP.getChipId()));  
String debugTopic = String("/ocean/no") +  
    ESP.getChipId() + String("/debug");  
String listenerTopic = String("/ocean/no") +  
    ESP.getChipId() + String("/listener");  
String debugMSG = String("no") +  
    String(ESP.getChipId()) + String(" test online!");  
  
if (client.connect(c.c_str())) {  
    client.publish(debugTopic.c_str(), debugMSG.c_str());  
    client.subscribe(listenerTopic.c_str());  
}  
}  
  
if (client.connected()) {  
  
    while (!mySerial.available()) { Serial.println("waiting");}  
  
    client.publish((String("/ocean/no") +  
        ESP.getChipId()).c_str(), s.c_str());  
    Serial.println(s);  
    digitalWrite(LED_BUILTIN, HIGH);  
    delay(500);  
    digitalWrite(LED_BUILTIN, LOW);  
    while (mySerial.available()) {  
        Serial.write(mySerial.read());  
    }  
  
    client.loop();  
}
```

```
    }  
}
```

**Código A.2:** Código-fonte destinado ao NodeMCU.

```
/**********************************************************/  
Este codigo e referente a inteligencia do  
aplicativo. Ele recebe as mensagens MQTT e atualiza  
os valores.  
*****/*
```

```
package com.oceanmanaus.sensorsreader;  
  
import android.os.Handler;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.util.Log;  
import android.widget.ImageView;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import org.eclipse.paho.android.service.MqttAndroidClient;  
import org.eclipse.paho.client.mqttv3.IMqttActionListener;  
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;  
import org.eclipse.paho.client.mqttv3.IMqttToken;  
import org.eclipse.paho.client.mqttv3.MqttCallback;  
import org.eclipse.paho.client.mqttv3.MqttClient;  
import org.eclipse.paho.client.mqttv3.MqttException;  
import org.eclipse.paho.client.mqttv3.MqttMessage;  
import org.eclipse.paho.client.mqttv3.MqttPersistenceException;
```

```
import org.w3c.dom.Text;

import java.io.UnsupportedEncodingException;

public class MainActivity extends AppCompatActivity {

    public static final String TAG = "OCEAN_MQTT_SUBSCRIBER";
    String clientId;

    ImageView portaImg;
    MqttAndroidClient client;
    String topic_pai = "/ocean/#";
    public static final int QOS = 0;

    TextView topicoHum, topicoGas, topicoTemp,
    topicoLum, payloadHum, payloadGas, payloadTemp, payloadLum;

    IMqttToken token;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        topicoHum = (TextView) findViewById(R.id.topico_hum);
        topicoGas = (TextView) findViewById(R.id.topico_gas);
        topicoTemp = (TextView) findViewById(R.id.topico_temp);
        topicoLum = (TextView) findViewById(R.id.topico_light);

        payloadHum = (TextView) findViewById(R.id.payload_hum);
```

```
payloadGas = ( TextView ) findViewById ( R . id . payload_gas );
payloadTemp = ( TextView ) findViewById ( R . id . payload_temp );
payloadLum = ( TextView ) findViewById ( R . id . payload_light );

final String clientId = MqttClient . generateClientId ();
client =
    new MqttAndroidClient ( this . getApplicationContext () ,
    "tcp://iot.eclipse.org:1883" ,
    clientId );

try {
    token = client . connect ();
    token . setActionCallback ( new IMqttActionListener () {
        @Override
        public void onSuccess ( IMqttToken asyncActionToken ) {

            Log . d ( TAG , "onSuccess" );
            String payload = clientId + " esta "+ online ! ;

            byte [ ] encodedPayload = new byte [ 0 ];

            try {
                encodedPayload = payload . getBytes ( "UTF-8" );

                MqttMessage message =
                    new MqttMessage ( encodedPayload );
                client . publish ( "/ocean/debug" , message );
            }

            IMqttToken subToken =
                client . subscribe ( topic_pai , QOS );
        }
    } );
}
```

```
        subToken.setActionCallback (
            new IMqttActionListener() {
                @Override
                public void onSuccess(
                    IMqttToken asyncActionToken) {
                    Log.d(TAG, topic_pai + " sobrescrito");
                }

                @Override
                public void onFailure(
                    IMqttToken asyncActionToken,
                    Throwable exception) {

                }
            });

        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (MqttPersistenceException e) {
            e.printStackTrace();
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onFailure(IMqttToken asyncActionToken,
        Throwable exception) {
```

```
        Log.d(TAG, "onFailure");

    }

});

client.setCallback(new MqttCallback() {
    @Override
    public void connectionLost(Throwable cause) {

    }

    @Override
    public void messageArrived(String topic,
                               MqttMessage message) throws Exception {
        Log.i(TAG, topic + " --> " + message.toString());

        if (topic.contains("umidade")) {
            topicoHum.setText(topic);
            payloadHum.setText(message.toString() + "%");
        } else if (topic.contains("gas")) {
            topicoGas.setText(topic);
            payloadGas.setText(message.toString() + "%");
        } else if (topic.contains("luminosidade")) {
            topicoLum.setText(topic);
            payloadLum.setText(message.toString() + " lux");
        } else if (topic.contains("temperatura")) {
            topicoTemp.setText(topic);
            payloadTemp.setText(message.toString() + " C");
        }
    }
})
```

```
    @Override
    public void deliveryComplete(IMqttDeliveryToken token) {

    }
});

} catch (MqttException e) {
    e.printStackTrace();
}

}
}
```

Código A.3: Código-fonte do aplicativo Android desenvolvido para visualização dos dados dos sensores.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:orientation="vertical"
    tools:context="com.oceanmanaus.sensorsreader.MainActivity">

    <android.support.v7.widget.CardView
        xmlns:card_view="http://schemas.android.com/apk/res-auto"
        android:id="@+id/card_view_hum"
        android:layout_gravity="center"
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="10dp"
    card_view:cardCornerRadius="4dp">
<RelativeLayout
    android:layout_width="wrap_content"
    android:layout_height="match_parent">

    <ImageView
        android:id="@+id/icon_hum"
        android:layout_alignParentLeft="true"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_marginRight="10dp"
        android:src="@drawable/humidity" />

    <TextView
        android:id="@+id/topico_hum"
        android:layout_toRightOf="@id/icon_hum"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="T pico" />

    <TextView
        android:id="@+id/payload_hum"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/topico_hum"
        android:layout_toRightOf="@id/icon_hum"
        android:layout_marginTop="12dp"
        android:textSize="18pt"
```

```
        android:text="Payload" />

    </RelativeLayout>

</android.support.v7.widget.CardView>

<android.support.v7.widget.CardView
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view_temp"
    android:layout_gravity="center"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="10dp"
    card_view:cardCornerRadius="4dp">
    <RelativeLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent">

        <ImageView
            android:id="@+id/icon_temp"
            android:layout_alignParentLeft="true"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:layout_marginRight="10dp"
            android:src="@drawable/temperature" />

        <TextView
            android:id="@+id/topico_temp"
            android:layout_toRightOf="@+id/icon_temp"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
```

```
        android:text="Topico "/>

    <TextView
        android:id="@+id/payload_temp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/topico_temp"
        android:layout_toRightOf="@id/icon_temp"
        android:layout_marginTop="12dp"
        android:textSize="18pt"
        android:text="Payload "/>

</RelativeLayout>

</android.support.v7.widget.CardView>

<android.support.v7.widget.CardView
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view_gas"
    android:layout_gravity="center"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="10dp"
    card_view:cardCornerRadius="4dp">
    <RelativeLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent">

        <ImageView
            android:id="@+id/icon_gas"
            android:layout_alignParentLeft="true"
```

```
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_marginRight="10dp"
        android:src="@drawable/gas" />

<TextView
    android:id="@+id/topico_gas"
    android:layout_toRightOf="@id/icon_gas"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text=" T pico "/>

<TextView
    android:id="@+id/payload_gas"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/topico_gas"
    android:layout_toRightOf="@id/icon_gas"
    android:layout_marginTop="12dp"
    android:textSize="18pt"
    android:text=" Payload "/>

</RelativeLayout>

</android.support.v7.widget.CardView>

<android.support.v7.widget.CardView
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view_light"
    android:layout_gravity="center"
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:layout_marginBottom="10dp"
    card_view:cardCornerRadius="4dp">
<RelativeLayout
    android:layout_width="wrap_content"
    android:layout_height="match_parent">

    <ImageView
        android:id="@+id/icon_light"
        android:layout_alignParentLeft="true"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_marginRight="10dp"
        android:src="@drawable/light" />

    <TextView
        android:id="@+id/topico_light"
        android:layout_toRightOf="@id/icon_light"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text=" T pico" />

    <TextView
        android:id="@+id/payload_light"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/topico_light"
        android:layout_toRightOf="@id/icon_light"
        android:layout_marginTop="12dp"
        android:textSize="18pt"
        android:text=" Payload" />
```

```
</RelativeLayout>

</android.support.v7.widget.CardView>

</LinearLayout>
```

Código A.4: Código-fonte do layout do aplicativo Android desenvolvido para visualização dos dados dos sensores.

### **A.3 Artigo Aceito para Publicação no ENCOSIS 2017**

O artigo que segue foi escrito pelos mesmos autor e orientador desta monografia e foi aceito para publicação no Encontro Regional de Computação e Sistemas de Informação (ENCOSIS) 2017, sendo defendido em 25 de maio de 2017.

# Arquitetura de IoT para Cidades Inteligentes

Douglas Lima Dantas<sup>1,2</sup>, Carlos Maurício Seródio Figueiredo<sup>1,2</sup>

<sup>1</sup>Escola Superior de Tecnologia – Universidade do Estado do Amazonas (UEA)  
Manaus – AM – Brasil

<sup>2</sup>Samsung Ocean Center (OCEAN)  
Manaus – AM – Brasil

{dld.eng, cfigueiredo}@uea.edu.br

**Abstract.** *Faced with urban growth, there is a concern for a quality of life in cities. Parallel to this, the technology has advanced and generated features that can fill some of these gaps. This work brings a proposal of Internet of Things based architecture to monitor data in urban environments and make them available with a friendly view. All this using accessible technologies that together help solve some problems.*

**Resumo.** *Diante do crescimento urbano, surge a preocupação com a qualidade de vida nas cidades. Paralelo a isso, a tecnologia tem avançado e gerado recursos que podem preencher algumas dessas lacunas. Este trabalho traz uma proposta de arquitetura baseada em Internet of Things para monitorar dados em ambientes urbanos e disponibilizá-los com uma visualização amigável. Tudo isso utilizando tecnologias acessíveis que, em conjunto, ajudam a solucionar alguns desses problemas.*

## 1. Introdução

A globalização, urbanização e industrialização têm sido as principais mudanças da humanidade no século XXI [Lee et al. 2014]. Além disso, a população mundial aumentará de 7 bilhões para 9 bilhões até 2050, e 70% das pessoas viverão nas áreas urbanas [Marchal et al. 2012]. Há muitos problemas que afetam a qualidade de vida dos urbanos, sendo alguns deles a poluição em suas várias vertentes (por exemplo, atmosférica, sonora e visual), mudanças climáticas e ilhas de calor. Sendo assim, torna-se imprescindível trazer soluções para mitigá-los.

Uma forma de combater esse problema é tornar as cidades mais inteligentes, obtendo dados do ambiente constantemente com o fim de tomar decisões e ter um maior controle. Faz-se necessário desenvolver soluções de monitoramento que informem com precisão suficiente e dentro de uma taxa aceitável de tempo. No entanto, tal desafio é cercado de várias restrições, como instável acesso à Internet e providência de fontes energéticas. Nesse contexto é que entra o ramo da *Internet of Things* (IoT) [Xia et al. 2012]. IoT trata da interconexão em rede de objetos do quotidiano, que são muitas vezes equipados com microcontroladores, provendo computação pervasiva, sensível ao contexto. Isto traz uma vasta gama de possibilidades de desenvolvimento de novos projetos que prometem melhorar a qualidade de vida das pessoas. Além disso, no ramo da IoT considera-se muitas vezes restrições tais como as que existem nesse problema.

A solução proposta neste trabalho traz os princípios de IoT para contornar essas restrições. A vantagem é utilizar uma nova abordagem que mais se adequa ao problema, com maior facilidade de projeto, gerenciamento e baixo custo. Esta rede de sensores pode ser utilizada, feitas as devidas adaptações, nos mais diversos tipos de ambiente tais como indústrias, ruas, parques e laboratórios., monitorando dados como temperatura, concentração de gases, umidade e luminosidade, fazendo uso do protocolo *Message Queue Telemetry Transport* (MQTT), bastante leve e ideal para ambientes críticos [Hunkeler et al. 2008]. Além disso, conterá uma API disponível em formato JSON e um *dashboard* como boa usabilidade para análise dos dados.

## 2. Tecnologias Utilizadas

O advento da IoT, com seu paradigma que geralmente utiliza sistemas embarcados conectados em rede, trouxe uma série de necessidades: tecnologias com baixo consumo de energia e capazes de trabalhar com uma baixa disponibilidade de banda, com conexões muitas vezes instáveis. Na área de redes, isso ocasionou o desenvolvimento de vários protocolos, desde as camadas mais baixas até as mais altas na pilha de protocolos. Dentro os principais protocolos na camada de aplicação são utilizados em aplicações de IoT, podemos citar o CoAP [Shelby et al. 2014] e o MQTT [ISO/IEC20922 2016], sendo o segundo utilizado neste trabalho.

Além disso, o bom funcionamento de um projeto de IoT depende da escolha da plataforma de hardware utilizada. Com a diminuição dos custos de produção e melhorias nas tecnologias de fabricação de semicondutores, surgiram várias plataformas baratas e simples de programar, democratizando a fabricação de dispositivos e criando toda uma comunidade em volta disso. Dentre as plataformas mais populares podemos citar o Arduino [Arduino 2005], utilizada neste projeto, o ESP8266 [Espressif 2014] e Raspberry Pi [RPF 2012].

### 2.1. Message Queuing Telemetry Transport (MQTT)

O MQTT [Banks and Gupta 2014] é um protocolo aberto extremamente leve para comunicação entre máquinas. Foi inicialmente criado pela IBM. Tem uma arquitetura cliente-servidor, no qual cada nó é um cliente e comunica-se com um servidor chamado *broker*. Ele utiliza TCP e é baseado em mensagens, utilizando o padrão *publish/subscriber*, onde um nó pode publicar mensagens e/ou sobreescriver em um ou mais endereços chamados de tópicos. Uma funcionalidade interessante deste protocolo é o *Quality of Service* (QoS), onde pode-se configurar entre três níveis de garantia de entrega das mensagens. O nível QoS 0 é o mais básico, e funciona de forma semelhante ao UDP. Ele utiliza apenas um verbo responsável por publicar a mensagem, todavia sem confirmação de entrega e da ordem correta dos pacotes. Já o segundo nível, o QoS 1, utiliza dois verbos, um para publicar e outro para confirmar a entrega, mas ainda sem garantia de ordem de chegada. Por fim, o último nível (QoS 2) utiliza quatro verbos, publicando e garantindo a entrega dos pacotes e ordem correta de entrega.

O MQTT foi o protocolo escolhido para ser utilizado no projeto descrito neste trabalho. As razões para isso são a sua arquitetura simples, com mensagens curtas e que são facilmente transmitidas mesmo em redes instáveis, juntamente com o modelo *publish/subscriber*, que facilita com o fato de vários nós-sensores poderem assinar um tópico automaticamente, tornando o mesmo mais adequado às necessidades.

## 2.2. Arduino

O Arduino [Arduino 2005] é uma plataforma de prototipagem eletrônica com código-fonte e hardware aberto, projetada utilizando um microcontrolador da família Atmel AVR, com suporte a entrada, saída e conexão serial via USB, operando em 5V. Foi criado tendo um baixo custo e tendo usuários iniciantes como seu público-alvo, tais como designers e artistas. Contém saídas digitais, analógicas e é utilizado tanto em projetos simples, como pequenas artes, quanto projetos complexos, como impressoras 3D. Esta foi a placa utilizada no projeto proposto por esse trabalho, devido ao seu baixo custo, além de várias saídas digitais, analógicas e um grande número de módulos chamados de *shields* e bibliotecas compatíveis, integrando diferentes sensores.

## 3. Trabalhos e projetos relacionados

Vários projetos de nós sensores e arquiteturas de IoT comerciais estão disponíveis no mercado. Há, também, inúmeros artigos na Internet que ensinam aos usuários como fazer seus próprios nós sensores. Além disso, há trabalhos acadêmicos que desenvolveram propostas de arquiteturas semelhantes a apresentada neste trabalho. Considerou-se trabalho ou projeto relacionado aqueles que têm como objetivo coletar dados de sensores ou disponibilizá-los na Internet através de um sistema de visualização.

### 3.1. Nós Sensores

Um trabalho relacionado é o *Forest Fire Sensor* [dos Santos and Figueiredo 2016], uma Rede de Sensores Sem Fio (RSSF) que também utiliza a plataforma Arduino e alguns sensores idênticos aos utilizados neste trabalho, porém utiliza outros protocolos de comunicação além do WiFi, como o XBee, por exemplo. No entanto, o trabalho é voltado para monitoramento de ambientes como, por exemplo, florestas, indústrias e parques, enquanto a arquitetura proposta neste artigo tem como fim monitoramento em cidades para disponibilizar os dados em formato amigável para uma boa visualização, além de uma API para consumo em outros sistemas. Os mesmos citam em seu trabalho *Tmote Sky* [Advanticsys 2015] e o *Waspmove* [Liberium 2012], dois nós sensores disponíveis no mercado bastante utilizados.

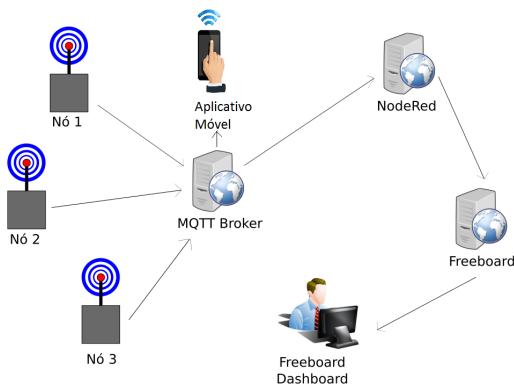
### 3.2. Disponibilização de Dados de Sensores

Há o projeto FIWARE Lab [Fiware 2016], que trata-se de uma iniciativa sem fins lucrativos para disponibilização de dados de sensores de vários locais do mundo, onde indivíduos e empresas podem tanto publicar quanto utilizar essas informações para pesquisas e desenvolvimento de aplicações. Várias cidades já tornaram seus dados públicos através desse projeto. Particularmente, o projeto aqui apresentado poderia integrar-se facilmente a esta iniciativa.

## 4. Arquitetura Projetada

A Figura 1 mostra uma visão geral da arquitetura de acordo com os itens abaixo, juntamente com o fluxo de informação.

- Um ou mais nós sensores;
- Broker MQTT;
- Servidor com NodeRED instalado;
- Freeboard;
- Aplicativo móvel.

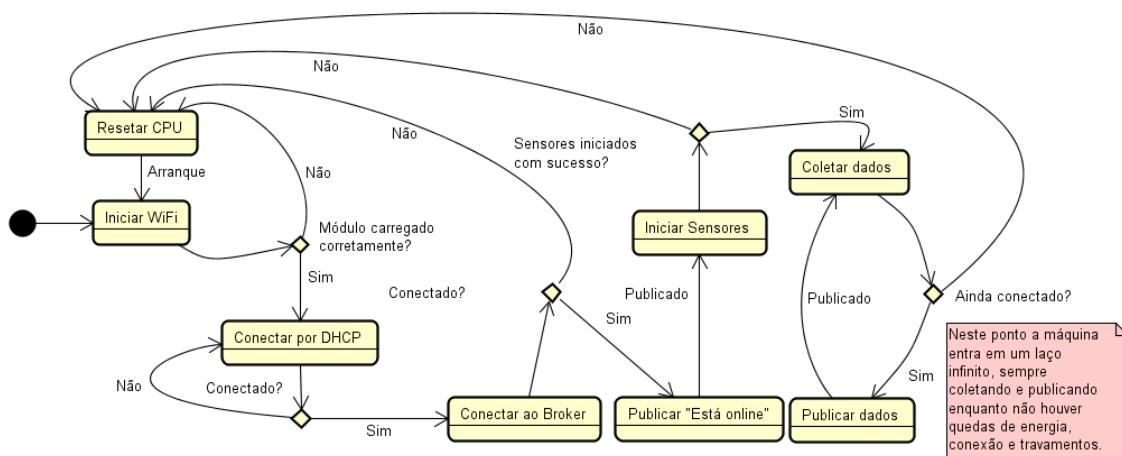


**Figura 1. Visão geral da arquitetura proposta pelo trabalho**

#### 4.1. O Nô Sensor

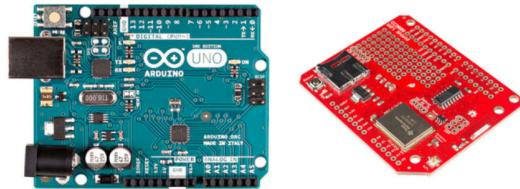
O nô sensor é composto por quatro módulos distintos. Primeiramente, tem-se a CPU, onde todo o processamento, gerenciamento de memória e execução de programas acontece. Logo depois, vem o módulo de alimentação, responsável por fornecer energia ao equipamento. Existe também o módulo de comunicação que permite ao nô conectar-se a um roteador WiFi. Por fim, há o módulo de sensoriamento, que contém os sensores de concentração de gás, luminosidade, umidade e temperatura. O único custo econômico neste projeto foi gerado com as peças do nô sensor e são discriminados na Tabela 4.1.

O módulo CPU consiste numa placa Arduino UNO R3, que utiliza o microcontrolador ATMega 328p. Ele é responsável pela execução do programa, controlando as entradas e saídas digitais e analógicas. A Figura 2 mostra o diagrama que representa o software embarcado no microcontrolador. Basicamente, o sistema faz a inicialização do módulo WiFi e se conecta ao broker. Após isso, entra em laço onde dados são lidos dos sensores e enviados ao broker. Em caso de queda de conexão ou travamentos o software reinicia. A Figura 3 mostra uma foto do Arduino utilizado.



**Figura 2. Máquina de estado finito do software embarcado na CPU.**

Já o módulo de alimentação é composto por duas entradas: uma *JACK*, de até 12V, e uma entrada USB de 5V. O sistema pode ser alimentado por uma delas. Neste

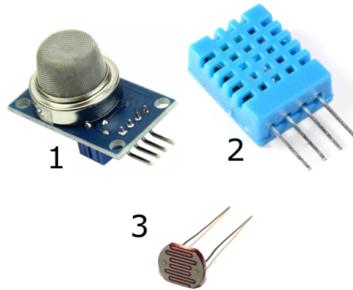


**Figura 3. Fotos mostrando o modelo de Arduino e shield WiFi utilizados no projeto.**

trabalho, utilizou-se uma fonte de alimentação externa que reduz a tensão de 127V para 5V e contém uma entrada USB que serve, além da alimentação, para conexão serial.

Para comunicação utilizou-se um *shield* WiFi CC3000 da Sparkfun para fazer a conexão com a Internet. Ele tem velocidade de 4Mb/s, segue o padrão IEEE 802.11 b/g e tem uma entrada para antena externa, que é opcional. A Figura 3 mostra uma foto do componente utilizado. O mesmo pode ser facilmente substituído por um *shield* 3G ou GSM para operação em campo.

O módulo de sensoriamento é composto por vários sensores. O primeiro deles é o MQ-2, sensor de concentração de gases com precisão para medir entre 300ppm e 10.000ppm vários tipos de gases tais como GLP, metano, propano, butano, hidrogênio, vapor de álcool e gás natural [HANWEI ELETRONICS CO.,LTD 2012]. O segundo é o sensor DHT11, que mede umidade relativa do ar e temperatura [D-ROBOTICS 2010]. O último é um fotoresistor, para medir luminosidade. Abaixo, as fórmulas utilizadas para obtenção das medidas de luminosidade e concentração de gases. A Figura 4 mostra imagens dos sensores utilizados, contendo o MQ-2 (1), DHT11 (2) e fotoresistor (3), respectivamente.



**Figura 4. Imagens dos sensores usados no trabalho.**

$$U(i) = \frac{5i}{1023} \quad (1)$$

$$L_{ux}(U) = \frac{\frac{2500}{U} - 500}{R} \quad (2)$$

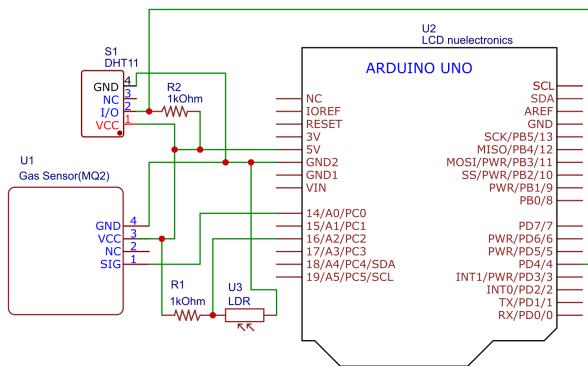
$$G(U) = 20U \quad (3)$$

Itens	Valor
Arduino UNO Rev3	R\$29,00
MQ-2	R\$9,50
DHT11	R\$7,50
LDR	R\$1,00
CC3000	R\$285,00
Total	R\$332,00

**Tabela 1. Discriminação dos custos com o nó sensor.**

A Equação 1 se refere a medição de tensão na porta analógica. O Arduino obtém um valor  $i \in [0, 1023]$  diretamente proporcional à tensão aplicada na entrada analógica, de acordo com a fórmula descrita na Equação 1. Já a Equação 2 é a fórmula para medir a luminosidade em Lux a partir da tensão aplicada no fotoresistor, onde  $R$  é o resistor do divisor de tensão utilizado [EMANT 2004]. Por fim, a Equação 3, onde diz que o MQ-2 tem sua tensão entre 0V e 5V de acordo com a concentração de gases, logo pode-se medir em porcentagem [HANWEI ELETRONICS CO.,LTD 2012].

Primeiramente a montagem foi realizada utilizando uma placa de prototipagem. Então, após verificar-se a corretude dos circuitos e o bom funcionamento do protótipo, confeccionou-se um gabinete utilizando uma impressora 3D, no qual instalou-se o projeto já sem a placa de prototipação. A Figura 5 apresenta o esquema de montagem utilizado para o circuito. A Figura 6(a) mostra o gabinete impresso com os componentes eletrônicos arranjados e, por fim, a Figura 6(b) exibe o nó sensor finalizado.



**Figura 5. Diagrama esquemático do circuito. Foram utilizados dois resistores de  $1k\Omega$  no projeto.**



(a) Arranjo dos componentes. (b) Nó sensor finalizado.

**Figura 6. Gabinete confeccionado com auxílio de impressora 3D.**

#### 4.2. MQTT broker

O MQTT *broker* utilizado no projeto foi o Mosquitto 1.4.9 [Eclipse 2006]. Foram disponibilizados seis tópicos para cada nó sensor, um para status do aparelho e quatro para recebimento dos dados de cada sensor, separadamente, e um quinto para uma *string* em formato JSON com todos os dados juntos, onde nome\_sensor representa o identificador único do nó. São eles:

- /ocean/sensores/debug;
- /ocean/sensores/<nome\_nó\_sensor>/temperatura;
- /ocean/sensores/<nome\_nó\_sensor>/umidade;
- /ocean/sensores/<nome\_nó\_sensor>/luminosidade;
- /ocean/sensores/<nome\_nó\_sensor>/gas
- /ocean/sensores/<nome\_nó\_sensor>/dados

#### 4.3. NodeRED

O NodeRED é uma ferramenta visual para programar rapidamente a comunicação entre APIs, dispositivos e disponibilização de dados. [Corbin 2016]. Ele foi utilizado no projeto para centralizar o recebimento de dados e publicá-los em um formato amigável JSON, para ser utilizado por qualquer outro projeto, como um *dashboard* ou um alarme, por exemplo. A Figura 7 mostra o programa criado com NodeRED, onde os dados dos sensores são recebidos nos blocos da Recepção MQTT, são convertidos para JSON em Tratamento dos Dados e disponibilizados para serem acessados via *GET* em Tratamento de Requisições, tornando-se um serviço WEB.

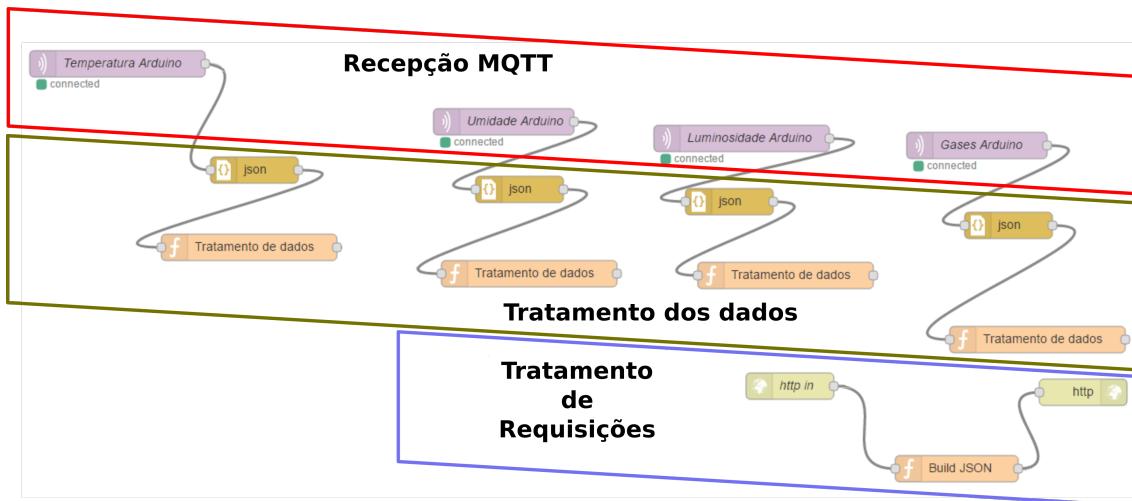


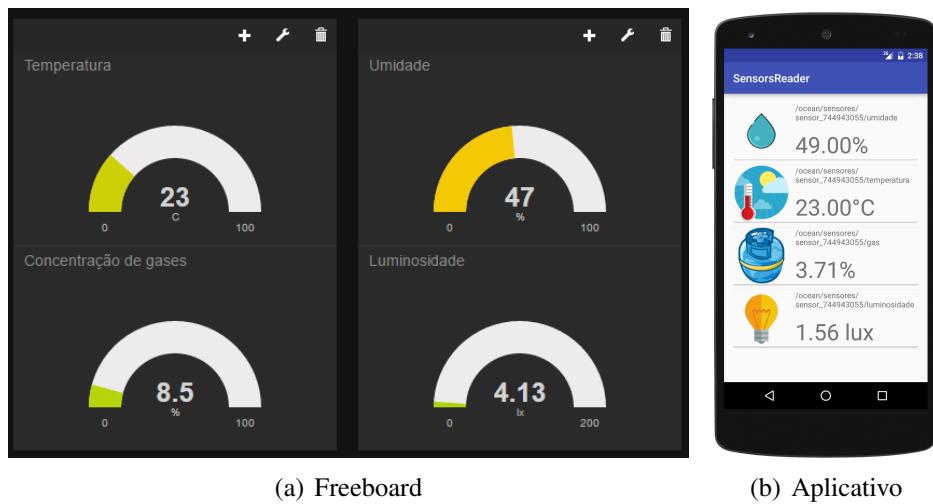
Figura 7. Programa gerenciador de dados feitos usando NodeRED.

#### 4.4. Freeboard

É uma plataforma de *dashboards* online, onde o usuário pode cadastrar-se, monitorar seus painéis com dados e disponibilizá-los, tudo de maneira simples e amigável [Freeboard 2014]. A Figura 8(a) mostra uma imagem dos painéis criados para o projeto, onde o usuário tem uma visão geral amigável em tempo real dos dados. Tal ferramenta permite que qualquer usuário possa fazer uso dos dados disponibilizados pelo NodeRED para sua aplicação específica.

#### 4.5. Aplicativo móvel

Foi desenvolvido um aplicativo para Android que tem a função de painel de visualização de dados, semelhante ao Freeboard, onde o usuário pode analisar os dados amigavelmente. A vantagem dessa alternativa é que mostra que a arquitetura, ao usar tecnologias populares, torna fácil a integração de novas aplicações. A Figura 8(b) mostra tela do aplicativo desenvolvido utilizando a plataforma *Android Studio*, com adição da biblioteca Paho [Eclipse 2012], que assina os tópicos MQTT utilizados pelo nó sensor publica, recebendo os dados assincronamente e disparando-os na tela.



**Figura 8. Interfaces WEB e móvel desenvolvidas para visualização amigável e em tempo real dos dados.**

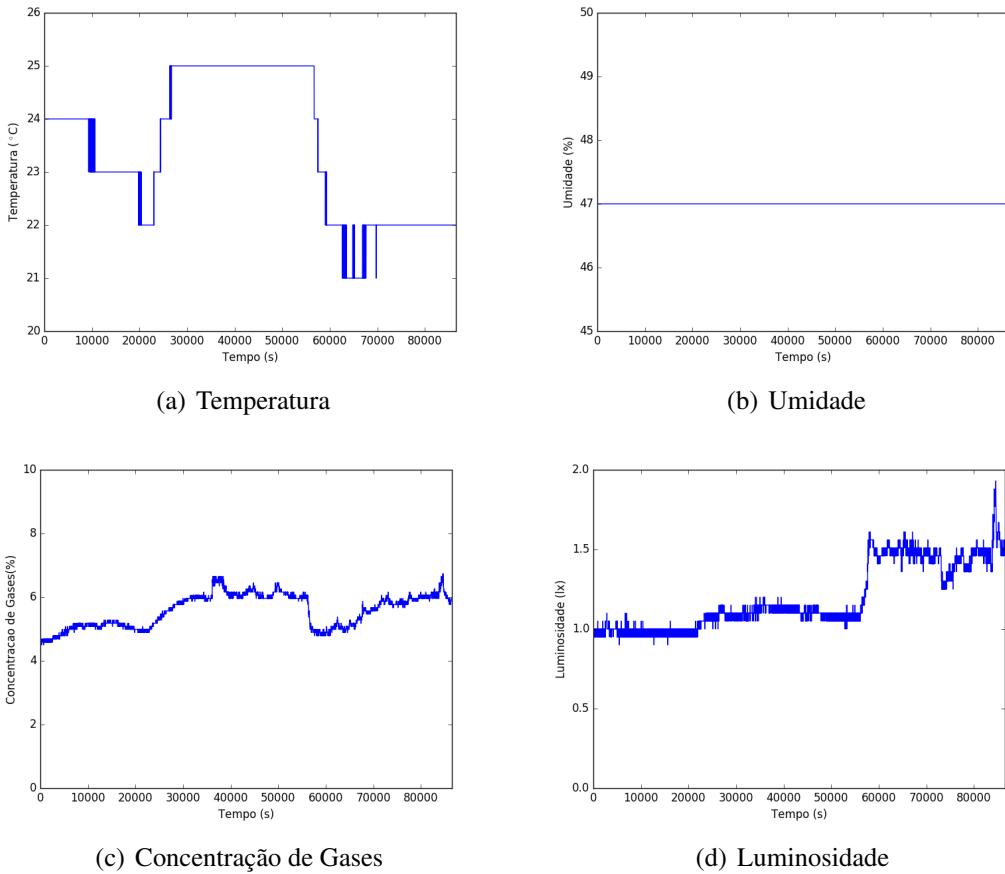
### 5. Estudo de caso

Para testar a arquitetura, uma unidade de nó sensor foi implantada no Samsung Ocean Center, local onde o mesmo foi projetado, e foram analisados dados por 24 horas. O mesmo foi exposto na sala principal, sujeito a variações de temperatura, em virtude dos condicionadores de ar, luminosidade indicando atividades no local. A Figura 5 apresenta os gráficos gerados durante este período de observação, onde se pode verificar que não houve mudança de umidade e os gases detectados foram concentrações baixíssimas.

Durante o período testado, o nó sensor transmitiu 16733 mensagens no tópico /oceaan/sensores/<nome\_nó\_sensor>/dados, apresentando um *delay* médio de 5,16s entre cada envio. O aplicativo fez seus recebimentos assincronamente, apresentando os dados conforme o proposto. O JSON disponível pelo NodeRED estava constantemente atualizado.

### 6. Conclusão

Este artigo apresentou uma solução de arquitetura de IoT para monitoramento de ambientes urbanos, de baixo custo, com tecnologias abertas e simples, tendo um ou mais nós sensores, um *dashboard* para visualização e um sistema que disponibiliza os dados em JSON, permitindo que qualquer aplicação possa fazer requisições e utilizar os dados para criar novas funcionalidades, funcionando como extensões. Isto torna o sistema versátil.



**Figura 9. Gráficos gerados a partir dos dados recebidos do nó sensor dentro de um período de 24 horas.**

É uma arquitetura de baixo custo e de fácil implementação, podendo ser aplicado em infraestrutura pública e extendido a diversas aplicações de interesse.

A arquitetura desenvolvida neste trabalho pode ser ampliada e resulta em diversos trabalhos possíveis. Uma possibilidade seria a de construir vários nós, instalá-los em uma localidade, tal como um *campus*, e utilizar atuadores que tivessem funcionamento condicionado aos dados obtidos, utilizando aprendizagem de máquina ao reconhecer padrões em variações de temperatura, por exemplo. Outro possível trabalho seria instalar vários nós em vias estratégicas e medir a poluição do ar em um período, permitindo que a Prefeitura soubesse onde ficam as áreas mais estratégicas para tomar medidas de conscientização e redução da poluição, diminuindo ilhas de calor.

## Agradecimentos

Os autores agradecem a Universidade do Estado do Amazonas pelo apoio. Os resultados apresentados nessa publicação foram obtidos por meio de atividades de Pesquisa e Desenvolvimento do projeto SAMSUNG OCEAN, patrocinado pela Samsung Eletrônica da Amazônia LTDA., apoiado pela SUFRAMA sob os termos da Lei Federal N° 8.248/91.

## Referências

- Advanticsys (2015). <http://www.advanticsys.com/shop/mtmcm5000msp-p-14.html>. 22/10/2016.
- Arduino (2005). <https://www.arduino.cc/>. 22/10/2016.
- Banks, A. and Gupta, R. (2014). MQTT Version 3.1.1. <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>.
- Corbin (2016). Node-RED: The fundamental, easy to use, open-source programming tool for IoT. <https://developer.ibm.com/dwblog/2016/node-red-programming-tool-open-source-iot/>. 22/10/2016.
- D-ROBOTICS (2010). DHT11 Humidity & Temperature Sensor. <http://www.micropik.com/PDF/dht11.pdf>. 22/10/2016.
- dos Santos, J. W. V. and Figueiredo, C. M. S. (2016). Rede de sensores sem fio para monitoramento de dados ambientais: Projeto de nó sensor de baixo custo. In *Anais do Encontro Regional de Computação e Sistemas de Informação*.
- Eclipse (2006). <http://mosquitto.org/>. 22/10/2016.
- Eclipse (2012). <https://eclipse.org/paho/>. 22/10/2016.
- EMANT (2004). Measure Light Intensity using Light Dependent Resistor (LDR). <http://emant.com/316002.page>. 22/10/2016.
- Espressif (2014). <https://espressif.com/>. 22/10/2016.
- Fiware (2016). <https://www.fiware.org/lab/>. 22/10/2016.
- Freeboard (2014). <https://freeboard.io/>. 22/10/2016.
- HANWEI ELETRONICS CO.,LTD (2012). Technical Data MQ-2 Gas Sensor. <http://sandboxelectronics.com/files/SEN-000004/MQ-2.pdf>. 22/10/2016.
- Hunkeler, U., Truong, H. L., and Stanford-Clark, A. (2008). Mqtt-s—a publish/subscribe protocol for wireless sensor networks. In *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*, pages 791–798. IEEE.
- ISO/IEC20922 (2016). Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1.
- Lee, J. H., Hancock, M. G., and Hu, M.-C. (2014). Towards an effective framework for building smart cities: Lessons from seoul and san francisco. *Technological Forecasting and Social Change*, 89:80–99.
- Liberium (2012). <http://www.libelium.com/products/wasp mote>. 22/10/2016.
- Marchal, V., Dellink, R., van Vuuren, D., Clapp, C., Château, J., Lanzi, E., Magné, B., and van Vliet, J. (2012). Oecd environmental outlook to 2050: the consequences of inaction.
- RPF (2012). <https://www.raspberrypi.org/>. 22/10/2016.
- Shelby, Z., Hartke, K., and Bormann, C. (2014). The constrained application protocol (coap). RFC 7252, RFC Editor. <http://www.rfc-editor.org/rfc/rfc7252.txt>.
- Xia, F., Yang, L. T., Wang, L., and Vinel, A. (2012). Internet of things. *International Journal of Communication Systems*, 25(9):1101.