UNIVERSIDADE FEDERAL DE SANTA MARIA

CENTRO DE TECNOLOGIA

CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Vinícius Cilião Guilherme

SISTEMA DE AUTOMAÇÃO RESIDENCIAL BASEADO EM INTERNET DAS COISAS

Santa Maria, RS

2017

Vinícius Cilião Guilherme

SISTEMA DE AUTOMAÇÃO RESIDENCIAL BASEADO EM INTERNET DAS COISAS

Trabalho de conclusão de curso apresentado ao Curso de Engenharia de Controle e Automação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Bacharel em Engenharia de Controle e Automação.**

Orientador: Prof. Dr. Frederico Menine Schaf

Santa Maria, RS

2017

Vinícius Cilião Guilherme

SISTEMA DE AUTOMAÇÃO RESIDENCIAL BASEADO EM INTERNET DAS COISAS

Trabalho de conclusão de curso apresentado ao Curso de Engenharia de Controle e Automação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Bacharel em Engenharia de Controle e Automação.**

Aprovado em 05 de setembro de 2017:

Frederico Menine Schaf, Dr. (UFSM)
(Presidente/Orientador)

Carlos Henrique Barriquello, Dr. (UFSM)

Claiton Moro Franchi, Dr. (UFSM)

Santa Maria. RS

2017

RESUMO

Este trabalho apresenta um estudo sobre *Internet of Things* (IoT) e o desenvolvimento e implementação de um sistema de automação residencial baseado em Internet das Coisas. O sistema utiliza Wi-Fi como meio de comunicação entre os dispositivos, e deve permitir que usuários possam controlar e supervisionar a residência remotamente com um *smartphone* pela Internet. A central do sistema foi implementada utilizando a plataforma de desenvolvimento embarcado Raspberry Pi 3, e os dispositivos utilizam o chip ESP8266 que integra interface de comunicação Wi-Fi e um microcontrolador Tensilica L106. Ainda, foi desenvolvido o protocolo de comunicação utilizado entre os dispositivos, visando um protocolo compacto, que possibilite um processo simples de adesão de novos dispositivos. O sistema conta com uma interface de configuração que permite, aliada a características do protocolo de comunicação, a integração do sistema e a identificação e classificação dos dispositivos nos diferentes ambientes da residência. O trabalho apresenta também os testes realizados no sistema com auxilio do *software* Wireshark para análise da comunicação. O sistema apresentou um baixo custo, com total de R$ 295,79.

ABSTRACT

This work presents a study about the Internet of Things (IoT), and the development and implementation of an IoT based home automation system. The system uses a Wi-Fi communication interface between the devices, and should allow users to remotely control and monitor the house using a smartphone via the Internet. The system center device was implemented using the embedded development platform Raspberry Pi 3, and the devices use the ESP8266 chip which integrates Wi-Fi communication interface and a Tensilica L106 microcontroller. In addition, the communication protocol for the devices was developed, aiming a compact protocol that enables an easy way to add new devices to the system. The system has a configuration interface that, allied to the communication protocol characteristics, allows the system integration and the identification and classification of the devices within the home environments. This work presents the tests made with the system with the aid of the software Wireshark to analyze the communication. The system presented low cost, with a total of R$ 295,79.

LISTA DE ILUSTAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BLE	*Bluetooth Low Energy*
BHS	*Bluetooth High Speed*
BR/EDR	*Basic Rate/Enhanced Data Rate*
CoAP	*Constrained Application Protocol*
DDNS	*Dynamic Domain Name System*
GUI	*Graphical User Interface*
GPIO	*General Purpose Input/Output*
HTTP	*Hypertext Transfer Protocol*
IEEE	*Institute of Electrical and Electronics Engineers*
IP	*Internet Protocol*
IoT	*Internet of Things*
M2M	*Machine-to-Machine*
MQTT	*Message Queue Telemetry Transport*
OSI	*Open Systems Interconnection*
PAN	*Personal Area Network*
PLC	*Power Line Communication*
RF	Rádio frequência
SSID	*Service Set Identifier*
TCP	*Tranport Control Protocol*
UDP	*User Datagram Protocol*
VoIP	*Voice over IP*
XMPP	*Extensible Messaging and Presence Protocol*

SUMÁRIO

1 INTRODUÇÃO

O avanço tecnológico trouxe nas ultimas décadas varias melhorias nos mais diversos campos. O surgimento da Internet, e a evolução vertiginosa da microeletrônica possibilitaram o surgimento de tecnologias que caminham constantemente para um senário cada vez mais conectado. Neste contexto surge a Internet das Coisas (*Internet of Things - IoT*), que tem possibilitado novas formas de interação entre o ser humano e os equipamentos e tecnologias que fazem parte de seu dia a dia. “A Internet das Coisas pode ser entendida como a expansão dos serviços de Internet a partir da interação entre pessoas e dispositivos, e entre dispositivos”. (RIBEIRO, 2016 *apud* TRIPATHI, 2014, p 44).

Hoje vive-se em uma nova sociedade, conectada. Neste novo cenário, percebe-se cada vez mais a introdução de tecnologias nas residências, de forma a atender as necessidades de seus moradores de acesso a informações digitais, de conforto e de segurança. A automação residencial possibilita a otimização das residências, seu principal papel é integrar sistemas tecnológicos que permitam suprir estas necessidades.

Com o aumento de tecnologias conectadas nas casas, e com o acesso à Internet tão ubíquo, a residência pode se beneficiar de um sistema de automação baseado em Internet das Coisas, que possibilite ao usuário o controle remoto de cargas de forma a gerenciar o consumo de energia, e o monitoramento remoto para aumento da segurança.

1.1 OBJETIVOS

Este projeto tem como objetivo o desenvolvimento de um sistema de automação residencial baseado em Internet das Coisas. O sistema deve possibilitar que o usuário controle e supervisione a casa remotamente pela Internet. O projeto visa ainda o desenvolvimento do protocolo de comunicação a ser utilizado entre os dispositivos, que a viabilize um processo simples de configuração de novos dispositivos.

2 INTERNET DAS COISAS

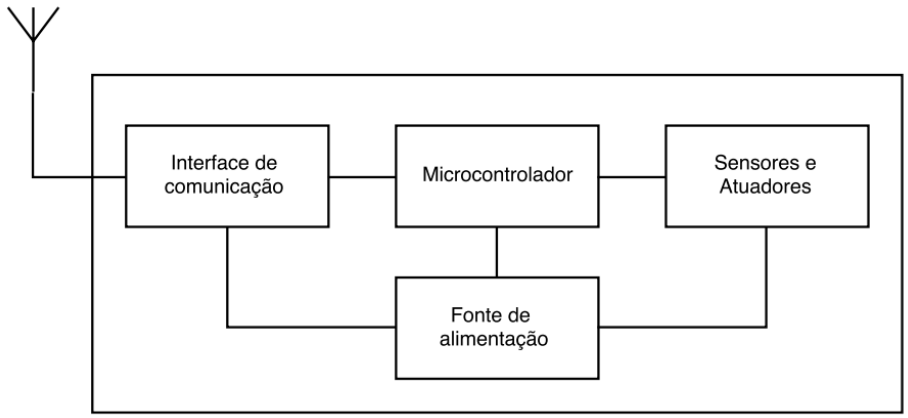
A Internet das Coisas surgiu dos avanços de diversas áreas como sistemas embarcados, microeletrônica, comunicação e tecnologia de informações. Pode ser considerada a extensão da Internet, ao proporcionar que objetos e equipamentos do dia a dia se conectem a Internet, esta conexão viabiliza que estes objetos sejam acessados como provedores de serviços. As novas habilidades dos objetos e equipamentos comuns geram um grande número de oportunidades no âmbito da automação residencial. (SANTOS, 2016).

2.1 DISPOSITIVOS

As coisas da Internet das Coisas são dispositivos inteligentes, ou seja, sistemas embarcados baseados em microcontroladores, capazes de transmitir e receber informações através de uma rede (MICRIUM, 2015), bem como interagir com o ambiente através de sensores e atuadores.

A arquitetura de *hardware* de um dispositivo é composta basicamente, como pode ser visto na Figura 2.1, de quatro componentes.

Figura 2.1 – Composição do hardware dos dispositivos



Fonte: (Adaptado de VASSEUR, 2010).

A interface de comunicação dá ao dispositivo a capacidade de enviar e receber dados através da rede. Para um dispositivo sem fio, este utiliza tipicamente um rádio transceptor,

enquanto dispositivos com fio se utilizam de meios de comunicação tais como Ethernet ou *Power Line Communication* (PLC) (VASSEUR, 2010).

O microcontrolador dá ao dispositivo sua inteligência. Este componente é um microprocessador dotado de memória, *timers,* entradas e saídas e conversor analógico-digital sendo assim capaz de fazer a leitura de sensores e controle de atuadores.

Sistemas embarcados podem utilizar microcontroladores de 8, 16 ou 32 *bits*. Graças a redução de custo de processadores de 32 *bits*, estes têm ganhado preferência no desenvolvimento de dispositivos utilizados em IoT por sua maior capacidade de processamento. Os principais fabricantes são Intel e ARM com processadores de baixo consumo energético direcionados a IoT (MICRIUM, 2015).

O dispositivo interage com o ambiente ao qual está inserido através de sensores e atuadores. Sensores possibilitam a leitura de variáveis físicas do ambiente como temperatura e umidade. Atuadores permitem o acionamento de cargas como lâmpadas e motores.

A fonte de alimentação é responsável por energizar todos os componentes do dispositivo, e deve ser definida conforme a aplicação do mesmo. Para um dispositivo fixo, este pode ser alimentado por um conversor conectado à rede elétrica, já para aplicações de difícil acesso, ou com dispositivos móveis, a fonte pode ser uma bateria ou painel fotovoltaico.

2.2 TECNOLOGIAS DE COMUNICAÇÃO

A escolha do meio de comunicação em um sistema IoT é muito importante, pois esta afetará diretamente nos dispositivos, seu *hardware* e consequentemente seu custo. A seguir são apresentadas algumas das principais tecnologias de comunicação.

2.2.1 Ethernet

O padrão Ethernet (IEEE 802.3) implementa uma comunicação cabeada e está presente em muitas das redes implementadas atualmente, por sua simplicidade, facilidade de adaptação e manutenção e por seu baixo custo. Há diferentes tipos de cabos que permitem diferentes velocidades de transmissão podendo chegar a 10 Gbps com a variante *10-Gigabit Ethernet*.

Este padrão oferece vantagens como o baixo consumo energético e sua alta vazão de dados, porém, por ser uma conexão cabeada, dispositivos permanecem fixos. Esta característica pode limitar seu uso em aplicações de IoT que exijam dispositivos com mobilidade. (SANTOS, 2016).

2.2.2 PLC

Power Line Communication é uma tecnologia que possibilita o envio de dados através da rede elétrica. Para dispositivos IoT o PLC pode ser atrativo devido a grande disponibilidade e facilidade de acesso a rede elétrica, e por já fornecer uma fonte de energia.

PLC apresenta a mesma de desvantagem da Ethernet, seu uso implica em dispositivos fixos. Além disso, como a rede de distribuição elétrica não é projetada para transferir sinais de alta frequência, o equipamento pode adicionar ruído a quaisquer sinais transferidos pela rede, de forma que dispositivos devem ser capazes de gerenciar a perda de dados. (VASSEUR, 2010).

2.2.3 ZigBee

O padrão ZigBee é baseado na especificação IEEE 802.15.4 que mira tecnologias de radio para aplicações de baixo consumo energético e baixa taxa de transmissão de dados. IEEE 802.15.4 especifica uma taxa de dados máxima de 250 kbps, porém, na prática, o padrão ZigBee alcança apenas taxas inferiores. O ZigBee opera na frequência 2.4GHz e é capaz de operar também nas frequências 868MHz e 915MHz. (SANTOS, 2016; VASSEUR, 2010).

O foco da especificação IEEE 802.15.4 é permitir a concepção de transceptores de rádio de baixo custo e baixa complexidade, o que faz do ZigBee uma tecnologia de comunicação atrativa para dispositivos inteligentes. Seu alcance máximo nominal é de algumas dezenas de metros e por se implementar uma comunicação sem fios este padrão deve permitir a concepção de dispositivos móveis dentro de seus limites de alcance.

2.2.4 Bluetooth

Bluetooth é uma das tecnologias mais utilizadas, em redes sem fios, para *Personal Area Netwoks* (PAN), e pode ser vista na grande maioria dos *smartphones* e *laptops*. Possui diferentes versões: *Basic Rate/Enhanced Data Rate* (BR/EDR) e *Bluetooth High Speed* (HS) correspondem às versões 3.0 ou inferiores e miram uma maior taxa de transmissão e velocidade e, portanto, implicam em um maior consumo energético. Já as versões 4.0 ou superiores correspondem ao *Bluetooth Low Energy* (BLE), este é voltado ao baixo consumo de energia, possibilitando que dispositivos possam ser alimentados por baterias para aplicações móveis (SANTOS, 2016).

2.2.5 Wi-Fi

Wi-Fi é uma tecnologia de comunicação sem fio baseada no padrão IEEE 802.11. Muito utilizada em residências e escritórios para proporcionar conexão sem fio de computadores à Internet. A maioria dos *laptops* e *smartphones* de hoje em dia possuem tecnologia de conexão Wi-Fi embarcada, bem como muitos roteadores e modem DSL comerciais integram esta interface de comunicação (VASSEUR, 2010).

Por ser tão utilizada e estar presente em todo lugar, esta tecnologia se apresenta como uma excelente candidata para aplicações de IoT. Além de sua onipresença, podem ser notadas as vantagens de proporcionar uma comunicação sem fios, permitindo mobilidade aos dispositivos, e ainda, como quase toda casa ou escritório com acesso a Internet possui um roteador Wi-Fi, o sistema dispensa a necessidade de um *gateway* para que dispositivos possam enviar e receber dados pela Internet.

Porém, como o padrão IEEE 802.11 foi desenvolvido para transferência de dados em alta velocidade para computadores, esta tecnologia apresenta um alto consumo energético se comparado a transceptores IEEE 802.15.4, como o ZigBee (VASSEUR, 2010). Dispositivos posicionados em locais de difícil acesso a rede elétrica e alimentados por baterias não poderiam suportar esta necessidade energética. No entanto novas tecnologias tem possibilitado o desenvolvimento de soluções de baixo custo, e baixo consumo energético (MICRIUM, 2015).

2.3 PROTOCOLOS DE COMUNICAÇÃO

Dispositivos inteligentes devem executar suas tarefas, tais como transmissão de dados, no menor tempo possível com o objetivo de reduzir o consumo de energia. Isto significa que suas mensagens devem ser as menores possíveis, o que tem efeito na escolha ou desenvolvimento de um protocolo de comunicação apropriado (MICRIUM, 2015).

A Figura 2.2 apresenta o modelo de referência OSI (*Open Systems Interconnection*) que divide sistemas de intercomunicação em sete camadas. A camada física representa dispositivos reais como transmissores e cabeamento. A camada de enlace de dados proporciona meios para a transferência de dados entre os nodos. A camada de rede está relacionada com a transferência de pacotes da origem para o destino, é onde se encontra o endereçamento IP (*Internet Protocol*). As camadas seguintes apresentam os protocolos de transporte e comunicação das aplicações embarcadas e são estudados nas seções a seguir.

Figura 2.2 – Modelo de Referência OSI

<table>
<tr><td>Camada de aplicação</td><td rowspan="3">HTTP, Websockets, XMPP, CoAP, MQTT</td></tr>
<tr><td>Camada de Apresentação</td></tr>
<tr><td>Camada de Sessão</td></tr>
<tr><td>Camada de Transporte</td><td>TCP, UDP</td></tr>
<tr><td>Camada de Rede</td><td>IP</td></tr>
<tr><td>Camada de Enlace de Dados</td><td>PPP, SLIP, Ethernet</td></tr>
<tr><td>Camada Física</td><td>Dispositivos Físicos</td></tr>
</table>

Fonte: (Adaptado de MICRIUM, 2015).

2.3.1 Protocolos de Transporte

Como visto no modelo OSI, os protocolos de transporte estão logo acima do protocolo IP. As aplicações não utilizam IP diretamente, mas sim protocolos de transporte para comunicação umas com as outras. São dois os protocolos de transportes mais utilizados: o *User Datagram Protocol* (UDP), e o *Transport Control Protocol* (TCP) (VASSEUR, 2010).

2.3.1.1 Transport Control Protocol

TCP é utilizado para a maioria das interações usuário-Internet, como e-mail e navegação via *Web-browser*. Este protocolo é recomendado para aplicações que necessitam um maior grau de confiabilidade, e o tempo de transmissão é menos importante. Confirmação de recebimento e retransmissões são características que o tornam confiável (MICRIUM, 2015). TCP é orientado a conexão, o que significa que antes que qualquer mensagem possa ser transferida, os dois dispositivos devem configurar, explicitamente, uma conexão (VASSEUR, 2010).

2.3.1.2 User Datagram Protocol

UDP é um protocolo mais simples, este não garante a entrega das mensagens, assim a confirmação deve ser implementada na aplicação. É ideal para aplicações que requerem uma transmissão mais rápida. UDP não exige que dispositivos estabeleçam conexão antes de trocar mensagens e permite envio em *multicast*.

UDP é um bom candidato para aplicações com dispositivos IoT por ser mais leve que o TCP, seu *header* é de apenas 8 *bytes* contra o *header* de 20 *bytes* do TCP, além disso é mais rápido por não ser orientado a conexão e não fazer recuperação de mensagens, porém, a confirmação da transmissão deve ser implementada na aplicação já que o protocolo não a faz, e mensagens podem se repetir e chegar fora de ordem, o que também deve ser tratado na aplicação (VASSEUR, 2010).

2.3.2 Protocolos IoT

Nas três camadas superiores do modelo de referência OSI está a aplicação embarcada, é onde se dá a padronização da comunicação. *Hypertext Transfer Protocol* (HTTP) e *Websockets* são protocolos de comunicação comuns da Internet, HTTP é a fundação de um modelo cliente-servidor usada para a Internet. O *Extensible Messaging and Presence Protocol* (XMPP) é um exemplo de protocolo que tem encontrado novo uso em IoT, este foi inicialmente desenvolvido como um protocolo de mensagens instantâneas, mas tem se estendido para aplicações como *Voice over IP* (VoIP) (MICRIUM, 2015; GREHS, 2016).

O problema destes protocolos é que eles são geralmente muito pesados para uso em aplicações IoT, os microprocessadores utilizados nos dispositivos IoT dispõem de pouca memória. Além disso, estes protocolos podem produzir mensagens muito grandes, com centenas ou até milhares de *bytes*, o que tem implicações no tempo de transmissão e consequentemente no consumo de energia. Em contrapartida, novos protocolos como o *Message Queue Telemetry Transport* (MQTT) e o *Constrained Application Protocol* (CoAP) são leves e produzem mensagens muito menores, com dezenas de *bytes*. (MICRIUM, 2015).

2.3.2.1 Message Queue Telemetry Transport

O MQTT é um protocolo de transporte de mensagens que segue o modelo cliente/servidor. Seu objetivo é coletar dados de diversos dispositivos e transportá-los a uma infraestrutura central. Assim, aplicações que necessitem dividir informações coletadas de vários dispositivos e disponibiliza-las ao usuário podem se beneficiar deste protocolo (GREHS, 2016).

Este protocolo mira redes com vários dispositivos que necessitam monitoramento ou controle a partir de um servidor na Internet. Pode suportar redes inseguras e com baixa largura de banda de forma a minimizar os requisitos de recursos dos dispositivos. O protocolo não suporta comunicação direta *device-to-device*, e nem *multicast.* (MICRIUM, 2015).

O MQTT opera sobre o protocolo de transporte TCP e implementa um padrão de troca de mensagens do tipo *publish/subscribe*. Os dispositivos clientes se conectam a um servidor capaz de concentrar todas as informações, o chamado *broker*, que é um intermediário no processo de comunicação. Os dispositivos *publishers* publicam informações para o *broker*, e

dispositivos que desejam receber uma determinada informação subscrevem, fazendo uma requisição ao *broker* (MARTINS, 2015)

2.3.2.2 Constrained Application Protocol

CoAP é um protocolo de transferência que mira dispositivos com recursos restritos, como pequena memória ou alimentados a bateria. O modelo de interconexão deste protocolo é similar ao modelo cliente/servidor do protocolo HTTP, porém CoAP suporta interações *machine-to-machine* (M2M) que resultam em uma implementação CoAP agindo tanto como cliente quanto como servidor (SHELBY, 2014).

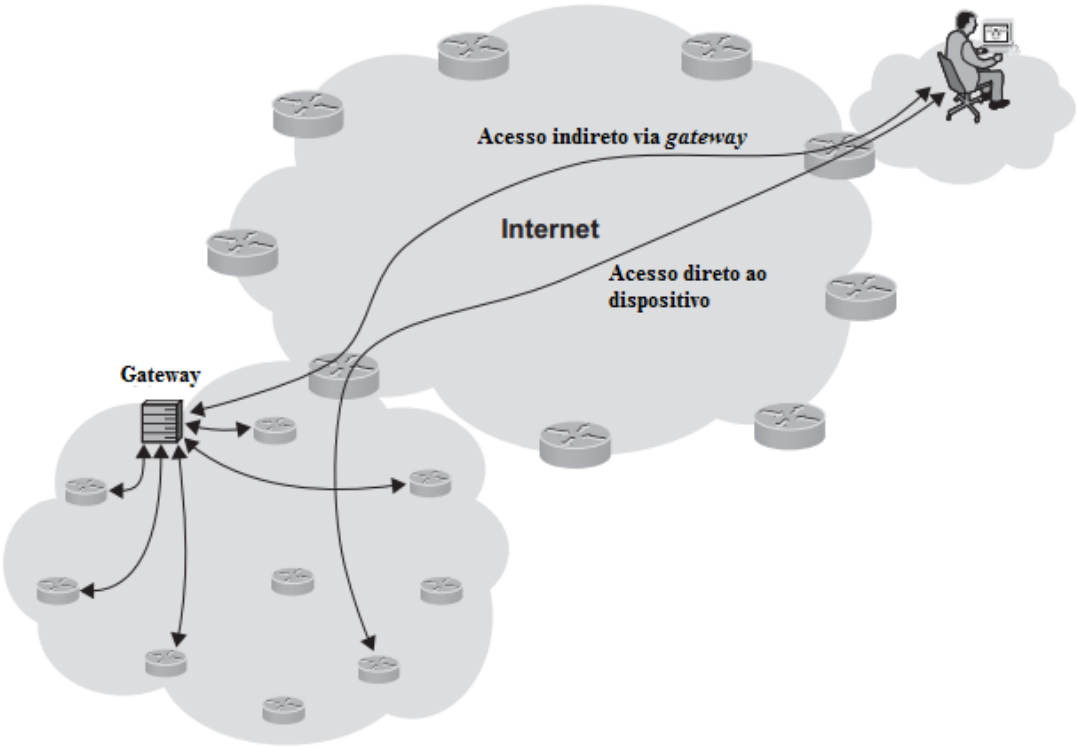
CoAP opera sobre o protocolo de transporte UDP, porém algumas características do TCP são replicadas, como mensagens confirmáveis. Isto aliado a um campo identificador de mensagens, para detecção de mensagens repetidas, em uma *header* de apenas 4 *bytes* tornam o CoAP compacto e confiável.

É um protocolo baseado no modelo *request/response*. Solicitações e respostas são transmitidas de forma assíncrona. O *header*, métodos e códigos de status são codificados de forma binária, o que reduz o *overhead* do protocolo. CoAP atende as necessidades de um protocolo pequeno, que permita uma conexão permanente, e interações diretas M2M e *multicast* (MICRIUM, 2015).

2.4 MODELO DE CONEXÃO

A Figura 2.3 apresenta o modelo de conexão dos dispositivos em um sistema IoT. Usuários se comunicam com os dispositivos diretamente pela Internet, ou via um *gateway*. Assim os dispositivos devem ser capazes de se comunicar com a Internet diretamente, ou devem se comunicar com um dispositivo que faz a interface entre a rede local e a Internet, o chamado *edge node*.

Figura 2.3 – Modelo de conexão dos dispositivos



Fonte: (Adaptado de VASSEUR, 2010).

3 AUTOMAÇÃO RESIDENCIAL

A principal função de sistemas de automação residencial é de integrar sistemas tecnológicos com o intuito de suprir as necessidades, da residência e usuários, de conforto, segurança e gerenciamento de energia. É uma área com diversas aplicações, como controle de iluminação, de acesso, controle remoto da residência, gerenciamento do consumo energético, e conforto, que apresenta um grande potencial de mercado. Segundo Vasseur (2010), cálculos de analistas da indústria configuram um número potencial, em longo prazo, de 50 a 100 dispositivos por residência, em cerca de 200 a 300 milhões de residências pelo mundo.

Quando se fala em segurança de residências, a automação residencial vai além de sistemas de alarme contra roubo. Assim foca em situações como acender remotamente as luzes da garagem antes de abrir o portão, ou dispor de um botão de pânico para aceder todas as luzes da casa. Dispositivos de segurança incluem ainda sensores de movimento, detectores de fumaça, sensores de gás e de vazamento de água.

Com a crescente atenção a sustentabilidade, o gerenciamento de energia tem papel importante em sistemas de automação residencial. Neste contexto o sistema realiza tarefas como o desligamento de cargas como lâmpadas e condicionadores de ar quando a casa está vazia.

Com o acesso à Internet tão ubíquo, um sistema de automação residencial que se utilize da Internet das Coisas pode providenciar o controle e supervisão remotos da casa a um baixo custo. Aplicações incluem o monitoramento da casa a distância, recebimento de alarmes de sensores de movimento e detectores de fumaça, e controle de lâmpadas e condicionadores de ar (VASSEUR, 2010).

3.1 CARACTERÍSTICAS E DESAFIOS

O ambiente residencial se apresenta muito menos nocivo aos dispositivos nele inseridos se comparado ao ambiente de uma planta industrial por exemplo. Dispositivos utilizados em automação industrial estão muitas vezes sujeitos a altas temperaturas, expostos a poeira e materiais químicos, a vibração e ruídos eletromagnéticos, características estas ausentes na maioria das vezes em ambientes residenciais. Desta forma sistemas de automação residencial permitem um menor grau de proteção e robustez dos dispositivos.

No entanto, este ambiente pode apresentar um desafio à dispositivos com tecnologias de comunicação do tipo rádio frequência (RF) como ZigBee e Wi-Fi. Em áreas populosas o uso de bandas RF licenciadas cresce rapidamente, especialmente da banda de 2.4GHz. Nesta situação é comum a necessidade de retransmissões para garantir a comunicação entre os dispositivos, o que pode afetar especialmente dispositivos alimentados a bateria nos quais a comunicação deve ser a mais eficiente possível com o objetivo de aumentar o tempo de vida da bateria (VASSEUR, 2010).

Na maioria das aplicações de automação residencial não existem restrições temporais rígidas, pacotes da comunicação podem ser perdidos e retransmitidos sem grandes efeitos. Ainda, a maioria das aplicações não exigem mobilidade, ou seja, os dispositivos são estacionários. Estas características favorecem o uso de dispositivos sem bateria, alimentados por conversores baseados em eletrônica de potência conectados a rede elétrica.

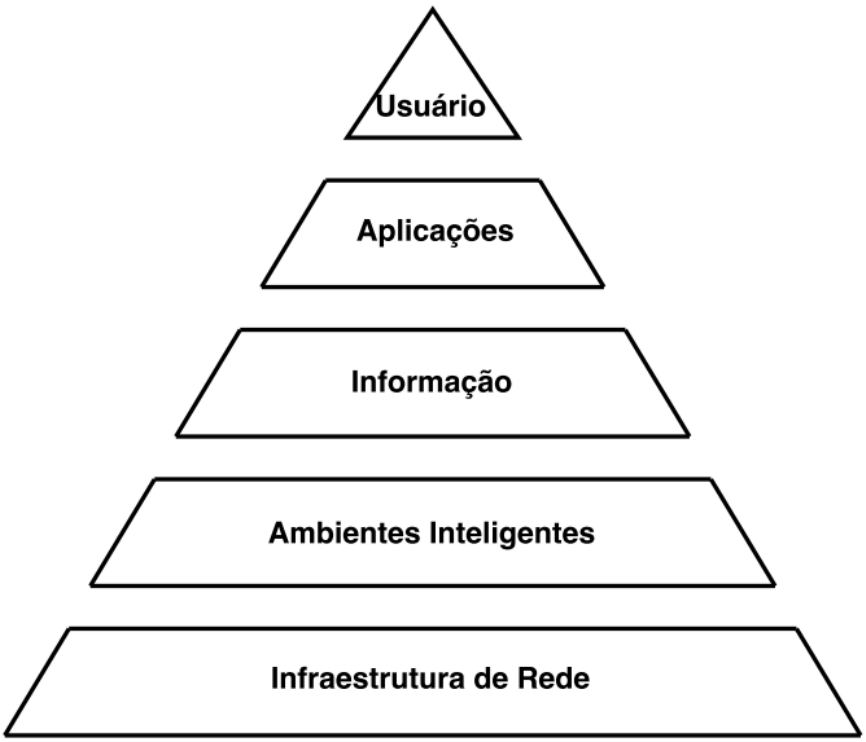
3.2 REDES DOMÉSTICAS

A Figura 3.1 apresenta o modelo de mercado das redes domésticas de Gearard O'Driscoll (2001 *apud* BOLZANI, 2004). A primeira camada representa a estrutura de comunicação do sistema, apresenta a rede de acesso, que interliga a residência ao provedor de Internet, e a rede doméstica, que interliga os dispositivos da residência. Juntas estas redes permitem a entrega de conteúdo digital ao usuário, e o acesso remoto aos dispositivos inteligentes permitindo a automação da residência (BOLZANI, 2004).

A segunda camada representa os diversos ambientes inteligentes do sistema. Estes contêm diversos sensores e atuadores que permitem o monitoramento e controle da residência. Os dispositivos trocam informações entre si e entre outros grupos de dispositivos inseridos em diferentes ambientes, diretamente ou através de um dispositivo central que gerencia a comunicação.

A camada de informação está ligada às informações digitais consumidas pelos usuários, ou seja, produtos e conteúdos virtuais como filmes, textos, fotografias e chamadas telefônicas. Os serviços e aplicações englobam serviços de apoio ao funcionamento da rede, e aplicações de Internet, telefonia, automação residencial, segurança e entretenimento. Juntas estas são as seções do mercado que devem gerar mais lucro. (BOLZANI, 2004).

Figura 3.1 – Modelo de mercado das redes domésticas.



Fonte: (Adaptado de BOLZANI, 2004).

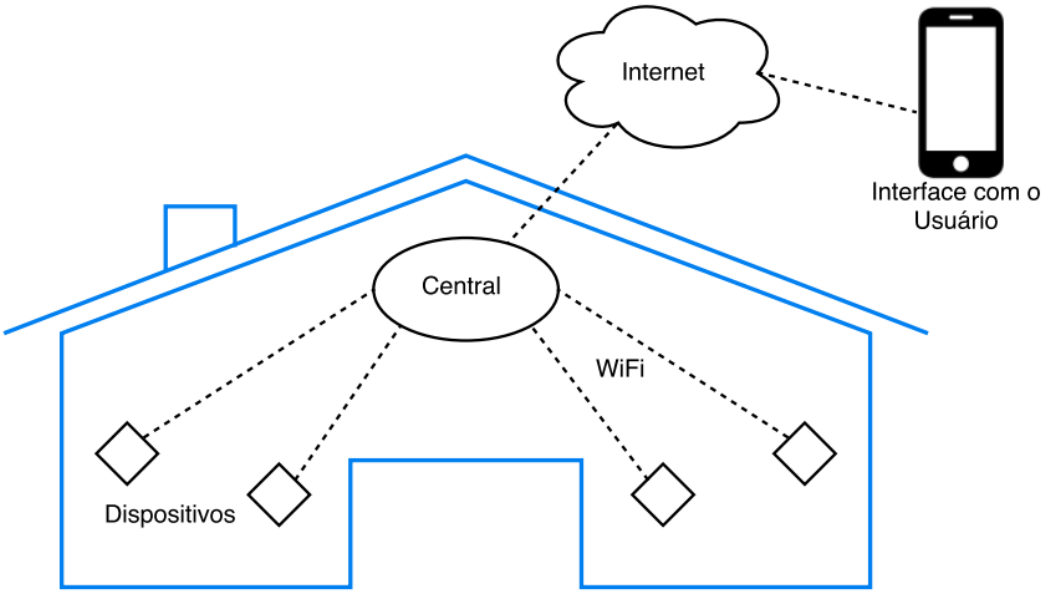
Por fim, na ultima camada está o usuário do sistema de rede. As camadas inferiores devem permitir ao usuário o consumo das mais variadas informações digitais, o monitoramento dos ambientes inteligentes e controle dos dispositivos inteligentes.

4 DESENVOLVIMENTO E IMPLEMENTAÇÃO

4.1 SISTEMA PROPOSTO

A proposta deste projeto é implementar um sistema de automação residencial baseado em Internet das Coisas. O modelo proposto pode ser visualizado na Figura 4.1 e consiste de um dispositivo central atuando como *edge node*, fazendo assim uma interface entre os dispositivos da rede domésica e a Internet. Usuários devem ser capazes de controlar e supervisionar o sistema pela Internet com um *smartphone*.

Figura 4.1 – Modelo proposto



Fonte: Autor.

Como roteadores Wi-Fi são tão comuns nas casas atualmente, Wi-Fi foi escolhida como tecnologia de comunicação, dispensando assim a necessidade de infraestrutura adicional.

É proposto ainda o desenvolvimento do protocolo de comunicação dos dispositivos com a central. O protocolo deve ser leve, de forma a não exigir grandes quantidades de memória dos dispositivos, deve implementar mensagens curtas para que tarefas de transmissão sejam rápidas, e proporcionar à aplicação a fácil identificação e adesão de novos dispositivos.

Baseado na descrição do sistema é possível especificar o *hardware* dos dispositivos e da central do sistema, e desenvolver o protocolo de comunicação, que atendam os objetivos propostos. Objetivou-se encontrar soluções de *hardware* de baixo custo, que apresentassem interface de comunicação Wi-Fi integrada.

4.2 CENTRAL

O dispositivo central deve ser capaz de gerenciar toda a comunicação do sistema, centralizar e distribuir informações. Para tanto foi utilizada a plataforma de desenvolvimento embarcado Raspberry Pi 3, mostrada na Figura 4.2. Trata-se de um computador de baixo custo, e tamanho reduzido. A versão 3 apresenta interface de conexão Wi-Fi embarcada.

Figura 4.2 – Plataforma utilizada para implementar a central.

Fonte: (raspberrypi.org).

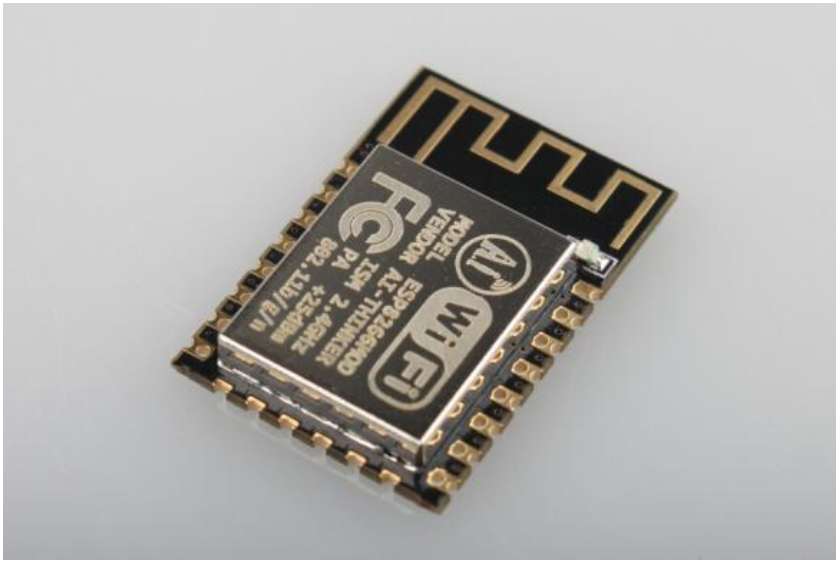
4.3 DISPOSITIVOS

Os dispositivos da rede de automação foram implementados a partir do chip ESP8266. Este chip integra interface de comunicação Wi-Fi, e um microcontrolador Tensilica L106 de

32 *bits*, dispensando a necessidade de circuitos com processador e interface de rede separados, conta ainda com tamanho bem reduzido, e consumo baixo de energia. Possui 17 GPIOs (*General Purpose Input/Output*), suporta interface i2c e apresenta um conversor ADC de 10 *bits*. (ESPRESSIF SYSTEMS, 2017)

Para os protótipos desenvolvidos foi utilizado o módulo ESP-12F mostrado na Figura 4.3. Este módulo já contem a circuitaria básica necessária ao funcionamento do chip ESP8266, como antena, cristal oscilador de 26MHz, e memória flash externa. Disponibiliza 11 GPIOs, incluindo o pino da entrada analógica (AI-THINKER, 2015). Este módulo pode ser programado utilizando linguagem de programação Lua, ou utilizando o *core* para o ambiente de desenvolvimento Arduino IDE, que foi utilizado neste projeto por apresentar uma linguagem de programação mais simples e familiar.

Figura 4.3 – Módulo ESP-12F utilizado no desenvolvimento dos dispositivos.



Fonte: (AI-THINKER, 2015).

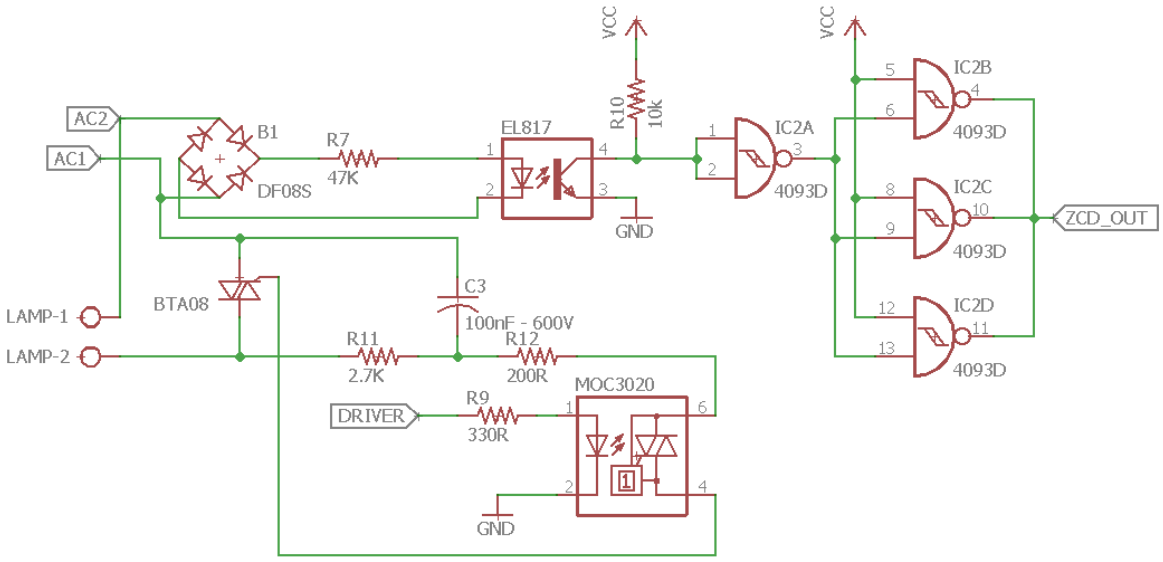
Foram desenvolvidos dois dispositivos, um para controle de potência AC que deve permitir a dimerização de lâmpadas para controle de luminosidade de ambientes, e outro para monitorar variáveis do ambiente como temperatura, umidade e luminosidade. Como os dispositivos são estacionários optou-se pela utilização de um conversor de potência como

fonte de alimentação para ambos dispositivos, o conversor utilizado permite uma tensão AC de entrada de até 265V, e tem saída CC de 5V. Como os módulos ESP-12F trabalham em 3.3V, foi utilizado o regulador de tensão LM1117 para adequação da tensão de alimentação dos módulos. Os protótipos dos dispositivos foram desenvolvidos utilizando o *software* CAD Eagle.

4.3.1 Dispositivo Para Controle de Potência AC

O dispositivo controla a potência entregue a carga a partir do controle do disparo de um triac. Este controle é feito em malha aberta pelo cálculo do tempo para o disparo dada uma potência desejada. Para tanto é necessário um circuito que gere uma interrupção no microcontrolador quando da passagem por zero da tensão em AC que alimentará a carga. A Figura 4.4 mostra o esquemático do circuito de potência utilizado, onde pode ser visto o circuito detector de passagem por zero utilizando um optoacoplador EL817, e um circuito Schmit-trigger utilizando o circuito integrado CD4093 para adequação do sinal de interrupção do microcontrolador.

Figura 4.4 – Esquemático do circuito de potência do dispositivo.

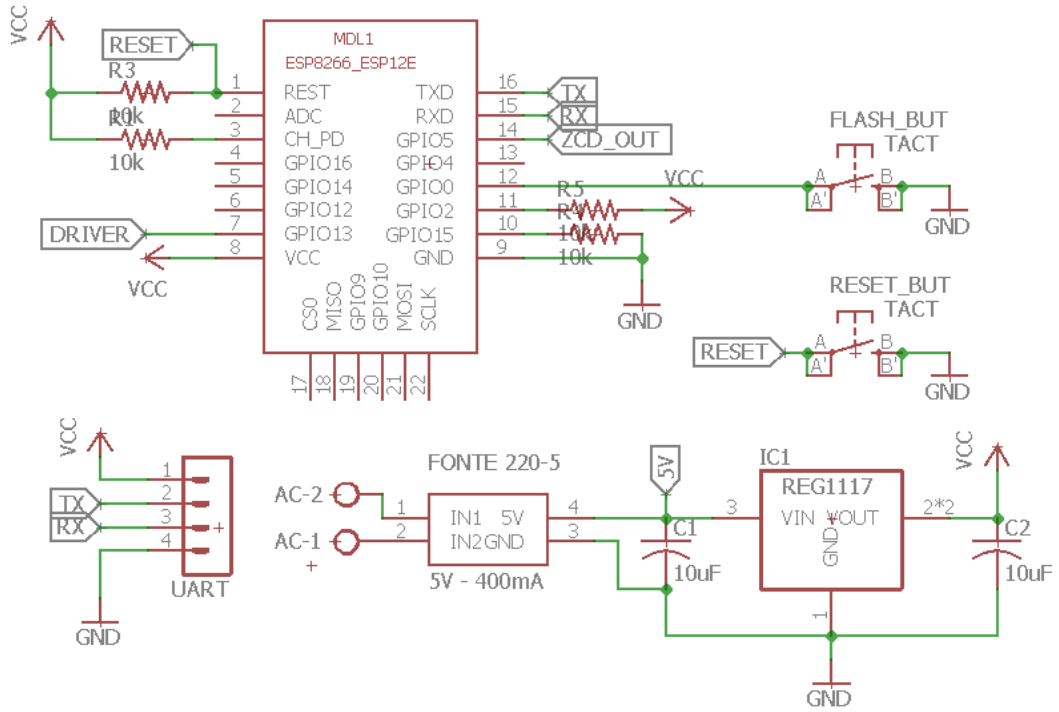


Fonte: Autor.

A Figura 4.4 apresenta ainda o circuito *driver* para o triac. Foi utilizado o triac BTA08 que permite o acionamento de cargas com corrente de até 8A, e o circuito integrado *driver* MOC3020.

A Figura 4.5 apresenta o esquemático do circuito regulador de tensão com LM1117, bem como a interface de programação do módulo ESP-12F utilizando os pinos 15 (RX) e 16 (TX) do módulo e botões ligados aos pinos 1 (*RESET*) e 12 (GPIO0) para reset do módulo e *boot* em modo de programação. Foi utilizado o pino 14 (GPIO5) do microcontrolador para a interrupção gerada pelo circuito detector de passagem por zero, e o pino 7 (GPIO13) para a saída de controle do circuito *driver* do triac.

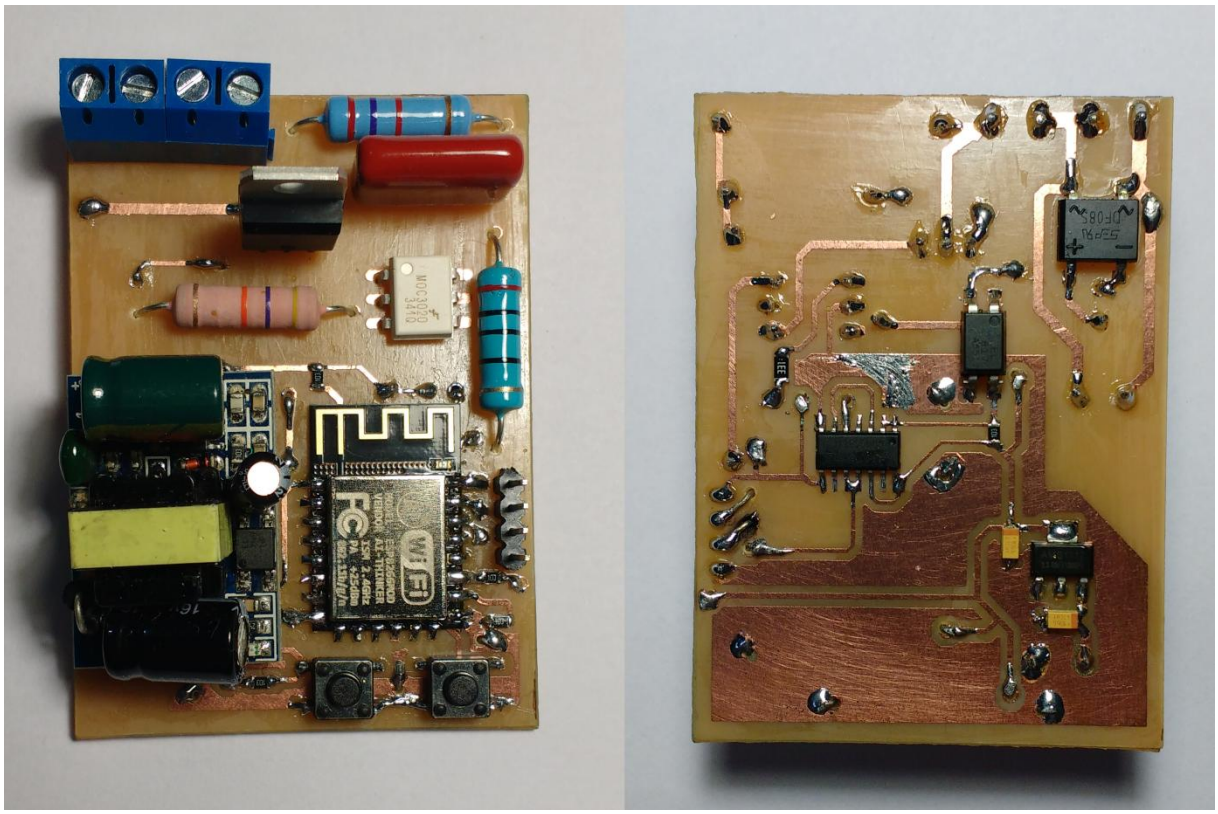
Figura 4.5 – Esquemático do dispositivo desenvolvido.



Fonte: Autor.

O protótipo do dispositivo é mostrado na Figura 4.6. Apresenta dimensões de 6,8cm por 4,9cm, o que possibilita seu uso em caixas de passagem.

Figura 4.6 – Dispositivo para controle de potência AC.



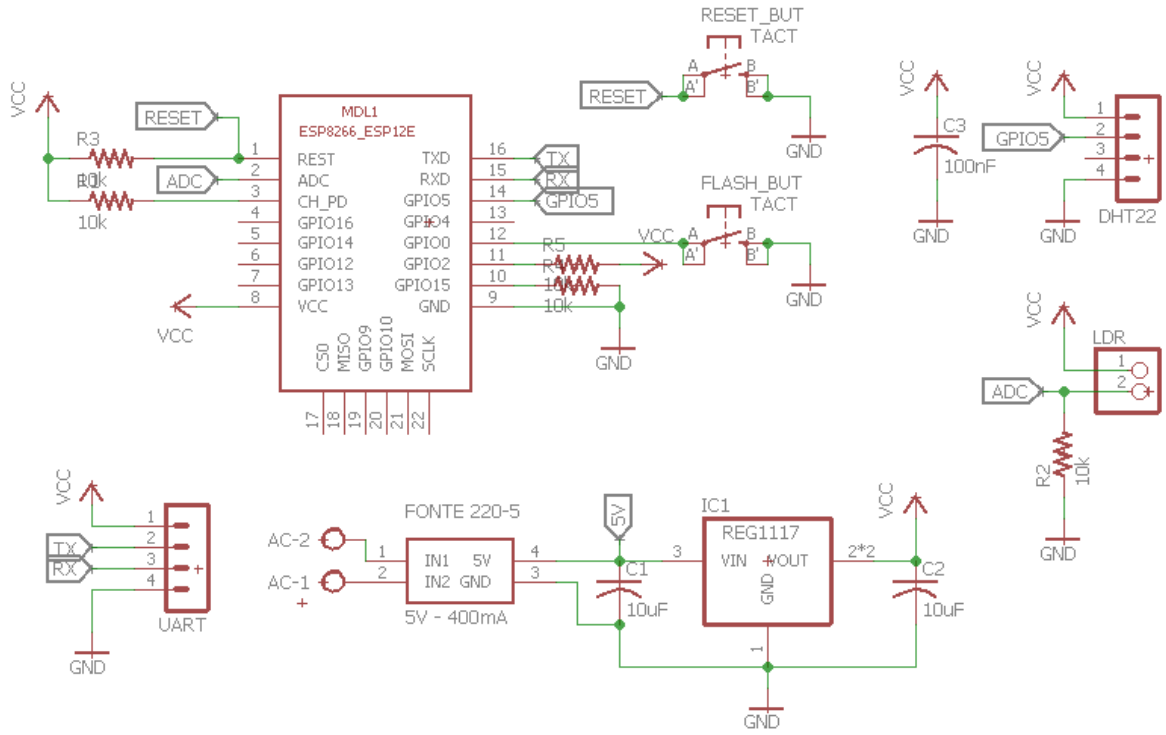
Fonte: Autor.

4.3.2 Dispositivo Sensor

Este dispositivo permite o monitoramento de temperatura, umidade e luminosidade do ambiente. Para o monitoramento de temperatura e umidade foi utilizado o sensor digital DHT22, que permite a medição de umidade relativa de 0 a 100% com precisão de +-2%, e a medição de temperatura de -40°C a 80°C com precisão de +- 0,5°C. Para monitoramento da luminosidade foi utilizado um sensor LDR (*light dependente resistor*), trata-se de um sensor resistivo que tem sua resistência variada conforme a intensidade da luz incidente sobre ele.

Na Figura 4.7 é apresentado o esquemático do dispositivo, foi utilizado o pino 14 (GPIO5) do módulo ESP-12F para comunicação com o sensor DHT22, e o pino 2 associado ao conversor AD do microcontrolador para leitura da tensão do divisor resistivo formado pelo sensor LDR e um resistor de 10k ohms.

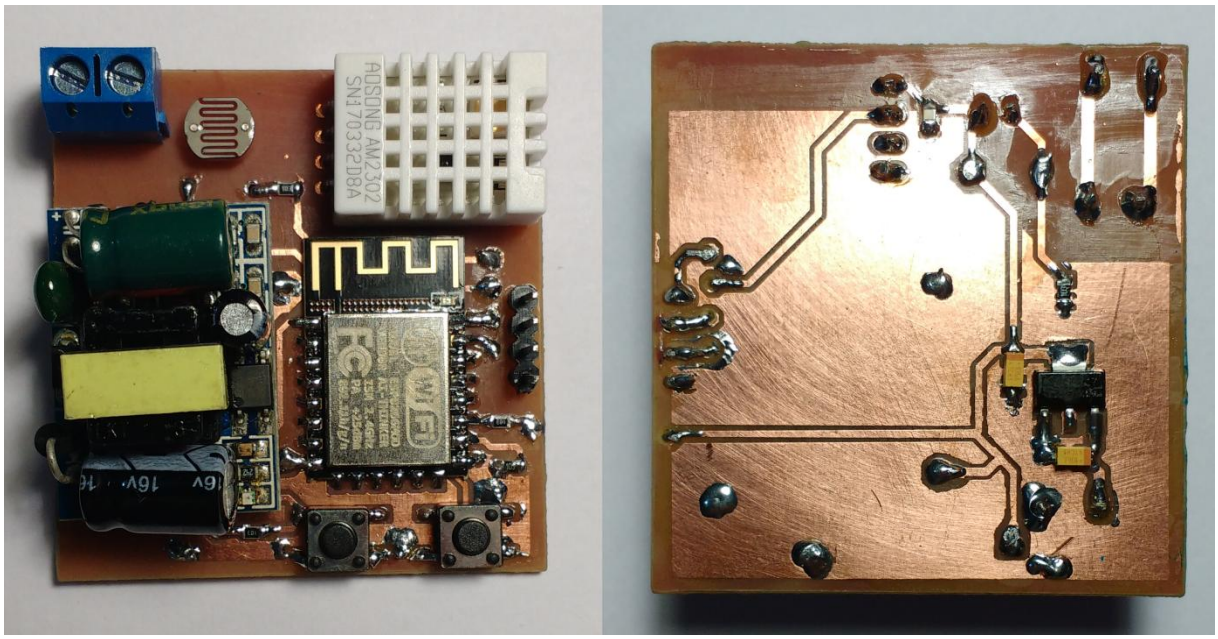
Figura 4.7 – Esquemático do dispositivo sensor.



Fonte: Autor.

O dispositivo apresenta a mesma circuitaria de regulagem de tensão, e de interface de programação, bem como os botões para *boot* em modo de programação, contida no dispositivo para controle de potência AC. Na Figura 4.8 é mostrado o protótipo do dispositivo sensor. Este tem dimensões de 5cm por 4,9cm.

Figura 4.8 – Dispositivo sensor.



Fonte: Autor.

4.4 PROTOCOLO DE COMUNICAÇÃO

O protocolo de comunicação desenvolvido visa uma implementação que exigirá pouco espaço em memória dos dispositivos. As mensagens possuem um *header* pequeno de apenas 3 *bytes*, o que implica em transmissões mais rápidas permitindo economia do consumo energético em dispositivos que possam ser alimentados a bateria.

O protocolo de transporte sobre o qual o protocolo desenvolvido opera é o UDP. Este foi escolhido por ser mais simples e mais leve que o TCP. Ainda, o sistema deve tirar proveito da possibilidade de se enviar mensagens em *broadcast* com o UDP para funções como a identificação de novos dispositivos introduzidos à rede. Características importantes do TCP são replicadas no protocolo desenvolvido, como a confirmação de recebimento das mensagens, e rejeição de mensagens repetidas.

O protocolo desenvolvido permite a subdivisão dos nodos em classes. No sistema desenvolvido neste projeto, por exemplo, as classes representam os vários ambientes da residência, e os nodos de cada classe são os dispositivos presentes em cada um destes ambientes. Este protocolo implementa um modelo do tipo *Publish/Subscribe* onde os nodos não se comunicam diretamente entre si mas sim através de um nodo central que atua como

broker, este deve ser capaz de concentrar informações, gerenciar as classes e tópicos publicados e distribuir as informações aos nodos *subscribers*.

4.4.1 Estrutura da Mensagem

A Figura 4.9 apresenta a estrutura das mensagens, bem como as informações contidas em cada campo da mensagem.

Figura 4.9 – Estrutura das mensagens.

Informação	Message ID	Node ID	Tipo	Instrução	Payload
Estrutura	Byte 1	Byte 2	Byte 3		Tamanho variável

Fonte: Autor.

Como no protocolo de transporte UDP as mensagens podem se repetir, o primeiro *byte* da mensagem apresenta um número identificador da mensagem, dessa forma mensagens repetidas podem ser ignoradas. Este número ajuda ainda a identificar a ordem em que as mensagens foram enviadas, já que no UDP as mensagens podem chegar fora de ordem.

O segundo *byte* apresenta um número de identificação dos dispositivos, a descrição deste campo é resumida na Tabela 1. Inicialmente todos os dispositivos contêm o ID padrão inicial igual a zero, até que a central os identifique e os atribua um novo ID. O ID número 255 é reservado para o envio de mensagens em *broadcast* aos dispositivos já identificados, dessa forma, até 254 dispositivos podem ser adicionados.

Tabela 1 – Descrição dos IDs (segundo *byte* da mensagem).

ID	Descrição
0	ID padrão inicial de dispositivos não identificados
1 a 254	ID atribuível
255	Reservado para envio de mensagens em *broadcast*

Fonte: Autor.

Como UDP não garante a entrega das mensagens, a confirmação do recebimento é feita através de uma mensagem resposta de *acknowledgment*, a necessidade desta mensagem de confirmação é indicada no *bit* mais significativo do terceiro *byte* da mensagem da seguinte forma:

- Campo “Tipo” com o *bit* 1: Confirmação necessária;
- Campo “Tipo” com o *bit* 0: Nenhuma confirmação é necessária.

Os *bits* restantes apresentam as instruções especificadas pelo protocolo. Estas serão apresentadas na próxima seção. A Figura 4.10 apresenta um exemplo onde uma mensagem com instrução de valor 5 está associada a uma mensagem com confirmação necessária, ou seja, o *bit* mais significativo do *byte* 3 da mensagem apresenta o *bit* 1.

Figura 4.10 – Exemplo de mensagem do tipo confirmável.

Message ID	Node ID	Tipo	Instrução							Payload
		1	0	0	0	0	1	0	1	
Byte 1	Byte 2	Byte 3								Variável

Fonte: Autor.

Por fim, o restante do datagrama pode ser utilizado para alocar um *payload*, o tamanho do *payload* varia conforme o conteúdo e conforme a instrução da mensagem, como é mostrado na próxima seção. Neste campo são especificados as classes e tópicos pertinentes à mensagem, estes são *strings* com codificação *ascii*.

4.4.2 Instruções

A Tabela 1 apresenta as instruções especificadas para o protocolo desenvolvido. Estas têm o proposito de instruir os nodos a cerca do propósito da mensagem sendo enviada.

Nas seções a seguir são descritas as instruções do protocolo de comunicação desenvolvido, bem como o campo de *payload* associado a cada uma das instruções.

Tabela 2 – Instruções do protocolo.

Instrução	**Valor**	***Payload***
Hello	0	Não
Request ID	1	Sim
Request IP	2	Não
Acknowledge	3	Não
Read	4	Sim
Write	5	Sim
Publish	6	Sim
Subscribe	7	Sim
Unsubscribe	8	Sim

Fonte: Autor.

4.4.2.1 Hello

Esta instrução é utilizada sempre em *broadcast* e tem o valor 0. A mensagem com a instrução *Hello* possui apenas o *header* fixo (*payload* de tamanho 0). É utilizada com o objetivo de encontrar novos dispositivos, e portanto deve ser enviada ao ID padrão inicial dos dispositivos (zero). Pode também ser utilizada para testar a comunicação com dispositivos já listados especificando o ID do mesmo.

4.4.2.2 *Request ID*

É a resposta de novos dispositivos à central uma vez recebido uma mensagem com a instrução 0 (*Hello*). Possui o valor 1. O dispositivo solicita à central um novo número ID, que deverá ser salvo na memória.

A Figura 4.11 apresenta as informações contidas no *payload* associado às mensagens com a instrução *Request ID*. O primeiro *byte* apresenta o tipo de dispositivo conforme a Tabela 3. No restante do *payload* o dispositivo pode passar, já na identificação, os tópicos aos quais publicará ou inscrever-se-á, assim passa um *byte* contendo o número de tópicos, e em seguida os tópicos propriamente ditos em campos de 15 *bytes* para cada tópico.

Figura 4.11 – *Payload* de mensagens com a instrução *Request ID.*

	Header			Payload				
Infromação	mID	nID	Confirmação/Instrução 1	Tipo	n° de tópicos	Tópico 1	Tópico 2	...
Tamanho	1 byte	1 byte	1 byte	1 byte	1 byte	15 bytes	15 bytes	...

Fonte: Autor.

Tabela 3 – Primeiro *byte* do *payload* associado à instrução *Request ID* referente a descrição do tipo de dispositivo.

Tipo	Descrição
0	Dispositivo *Publisher*
1	Dispositivo *Subcriber*
2	Ambos *Publisher* e *Subscriber*

Fonte: Autor.

4.4.2.3 *Request IP*

Na maioria dos sistemas de rede doméstica o endereçamento dos dispositivos se da de forma dinâmica, ou seja, dispositivos podem receber um novo endereço IP quando religados, ou quando o roteador é religado. Assim é importante que a central atualize esta informação para garantir a comunicação.

Uma vez que a central não recebe uma resposta de um dispositivo, esta envia uma mensagem em *broadcast*, especificando o ID do dispositivo, com a instrução *Request IP*, de valor 2. Assim como na instrução *Hello*, mensagens com esta instrução também não possuem *payload.*

4.4.2.4 Acknowledge

Como citado anteriormente, no protocolo de transporte UDP não há confirmação de recebimento das mensagens. Com o objetivo de aumentar a confiabilidade do protocolo, a acusação de recebimento é feita através de mensagens com a instrução *Acknowledge*, enviada como resposta a mensagens com o campo de confirmação setado com *bit* 1, como mostrado anteriormente. Esta instrução também possui apenas o header fixo e tem valor 3.

4.4.2.5 Read

A instrução *Read* é utilizada para mensagens de solicitação de informações, tal como a solicitação de uma leitura única do valor de algum tópico quando não há necessidade de inscrição para recebimento contínuo de atualizações do tópico.

A informação desejada é alocada nos primeiros 15 *bytes* do *payload* como pode ser visto na Figura 4.12. Caso a solicitação seja para leitura de algum tópico, a classe a qual este pertence deve ser especificada. A instrução *Read* tem o valor 4.

Figura 4.12 – *Payload* de mensagens com a instrução *Read*.

	Header			Payload	
Infromação	mID	nID	Confirmação/Instrução 4	Informação	Classe
Tamanho	1 byte	1 byte	1 byte	15 bytes	15 bytes

Fonte: Autor.

4.4.2.6 Write

A instrução *Write* possui o valor 5 e é utilizada para o envio de informações, sejam estas respostas a solicitações (*Read*), envio do novo ID aos nodos (resposta a *Request* ID), ou distribuição da atualização de algum tópico aos nodos *subscribers*. A informação é explicitada nos primeiros 15 *bytes* do *payload* seguido do conteúdo pertinente àquela informação como pode ser visto na Figura 4.13.

Figura 4.13 – *Payload* de mensagens com a instrução *Write*.

	Header			Payload	
Infromação	mID	nID	Confirmação/Instrução 5	Informação	Conteúdo
Tamanho	1 byte	1 byte	1 byte	15 bytes	variável

Fonte: Autor.

4.4.2.7 Publish

A instrução *Publish* é utilizada pelos dispositivos para atualizar o valor dos tópicos associados a este nodo, ou ainda para publicar novos tópicos. Caso o tópico publicado não pertença à classe a qual o nodo está inserido, esta deve ser especificada nos 15 primeiros *bytes* do *payload*. Em seguida é especificado o tópico publicado, seguido do conteúdo associado a este tópico como pode ser visto na Figura 4.14. A instrução *Publish* possui o valor 6.

Figura 4.14 – *Payload* de mensagens com a instrução *Publish*.

	Header			Payload		
Infromação	mID	nID	Confirmação/Instrução 6	Classe	Tópico	Conteúdo
Tamanho	1 byte	1 byte	1 byte	15 bytes	15 bytes	variável

Fonte: Autor.

4.4.2.8 Subscribe

A instrução *Subscribe* é utilizada pelos dispositivos para solicitar sua inscrição em um tópico com o objetivo de receber atualizações deste tópico continuamente. O tópico ao qual se deseja inscrição é especificado nos primeiros 15 *bytes* do *payload* da mensagem, caso o tópico não pertença a classe a qual o nodo solicitante está inserido, a classe também deve ser especificada como mostrado na Figura 4.15. Esta instrução possui o valor 7.

Figura 4.15 – *Payload* de mensagens com a instrução *Subscribe*.

	Header			Payload	
Infromação	mID	nID	Confirmação/Instrução 7	Tópico	Classe
Tamanho	1 byte	1 byte	1 byte	15 bytes	15 bytes

Fonte: Autor.

4.4.2.9 Unsubscribe

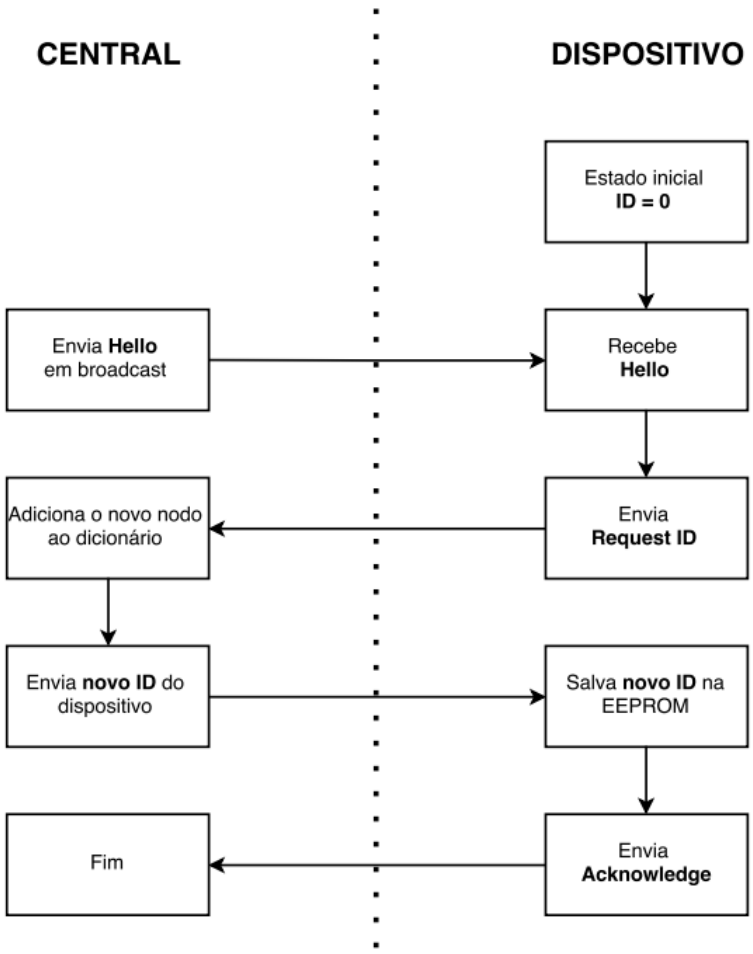
A instrução *Unsubscribe* é utilizada pelos dispositivos para solicitar sua retirada da lista de inscritos de um tópico de forma a não mais receber atualizações deste. Esta instrução tem o valor 8 e o *payload* das mensagens associadas a esta instrução é idêntico ao apresentado

em mensagens com a instrução *Subscribe*. Em ambas as instruções *Subscribe* e *Unsubscribe* o nodo solicitante pode especificar apenas a classe, de forma a inscrever-se ou anular sua inscrição em todos os tópicos daquela classe.

4.4.3 Identificação de Dispositivos

Como citado anteriormente, o protocolo tira proveito da possibilidade de envio de mensagens em *broadcast*, com o protocolo de transporte UDP, na identificação de novos dispositivos. A Figura 4.16 apresenta um fluxograma com a rotina utilizada para este fim.

Figura 4.16 – Rotina de identificação de novos dispositivos.



Fonte: Autor.

Uma vez inserido à rede, o dispositivo possui o ID inicial padrão igual a zero, quando a central envia a instrução *Hello* em *broadcast* com *node* ID igual a zero, o novo dispositivo

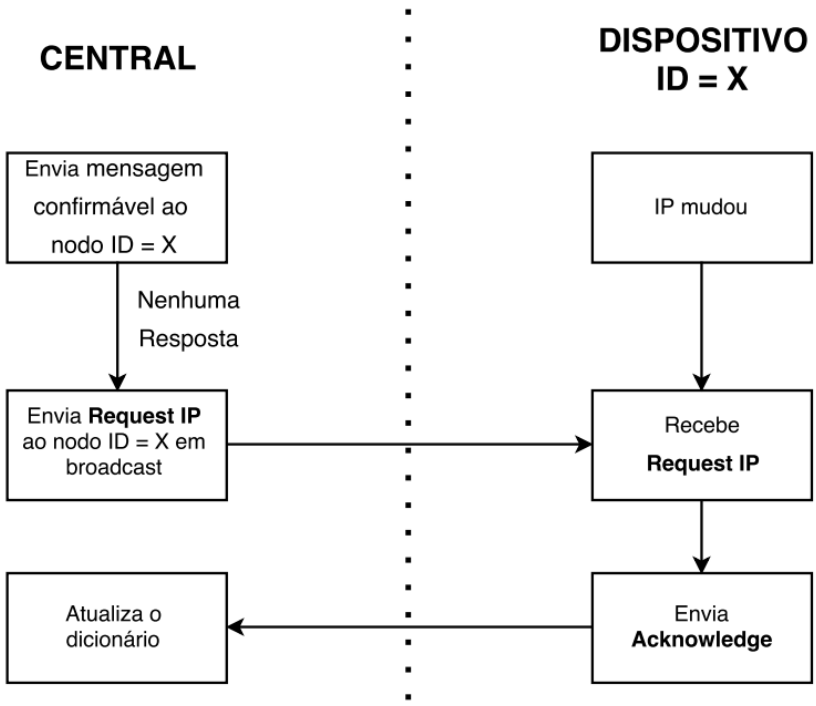
responde com a instrução *Request* ID, então a central adiciona o IP deste dispositivo ao dicionário de dispositivos associado a um novo ID, este novo ID é enviado, numa mensagem associada à instrução *Write*, ao dispositivo que deverá salvá-lo na memória EEPROM.

4.4.4 Atualização do Endereço IP

O endereçamento dos dispositivos feito pelo roteador se dá de forma dinâmica, isto significa que se o roteador reiniciar o dispositivo poderá receber um novo endereço IP, assim a central deve atualizar esta informação em seu dicionário de dispositivos. A Figura 4.17 apresenta um fluxograma com a rotina implementada para tratar deste problema.

Uma vez enviada uma mensagem pela central a um dispositivo, a qual se aguarda uma resposta, caso este dispositivo não responda a mensagem, a central trata esta ausência de resposta como uma alteração no endereço IP do dispositivo e envia em *broadcast* a instrução *Request* IP especificando o *node* ID para que somente o dispositivo de interesse responda. Recebida esta solicitação, o dispositivo responde com uma mensagem de confirmação (instrução *Acknowledge*), e a central pode então atualizar o endereço deste dispositivo.

Figura 4.17 – Rotina de atualização do endereço IP de dispositivos.

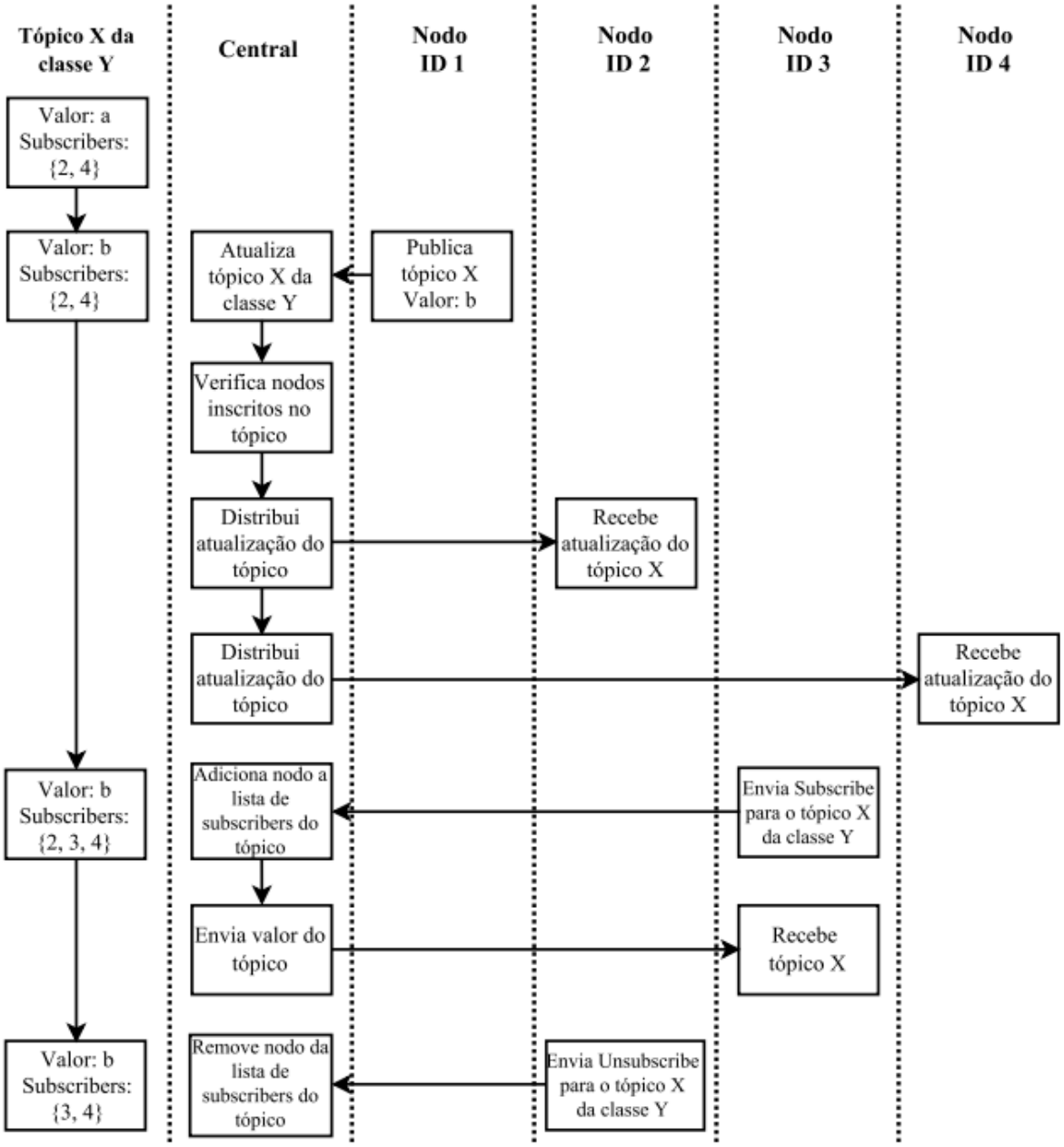


Fonte: Autor.

4.4.5 *Publishing, Subscribing* e *Unsubscribing*

O fluxograma da Figura 4.18 apresenta um exemplo de troca de mensagens que envolve a atualização e inscrições de tópicos e gerenciamento de listas de inscritos. Na primeira coluna é possível ver o conteúdo do tópico X bem como sua lista de nodos inscritos a cada interação que altera alguma destas variáveis, este tópico é publicado pelo nodo de ID 1 que pertence a classe Y.

Figura 4.18 – *Publishing*, *Subscribing* e *Unsubscribing*.



Fonte: Autor.

No início o tópico X contem o valor A, até que o nodo 1, que é *publisher* do tópico X, envia um novo valor (instrução *Publish*). A central atualiza o valor do tópico, verifica quais nodos estão inscritos no mesmo, e em seguida distribui a nova atualização do tópico a estes nodos, ou seja, mensagens com a instrução *Write* contendo o tópico e seu valor são enviadas aos nodos *subscribers* 2 e 4.

Em seguida o nodo 3 envia uma solicitação para inscrição no tópico X da classe Y (instrução *Subsribe*), a central adiciona o ID 3 à lista de nodos inscritos no tópico e logo após envia o valor da ultima atualização do tópico ao novo nodo *subscriber*. A ultima interação consiste da solicitação, enviada pelo nodo 2, para saída da lista de inscritos do tópico X da classe Y (instrução *Unsubscribe*), e atualização da lista de inscritos pela central.

4.5 INTERFACE COM O USUÁRIO

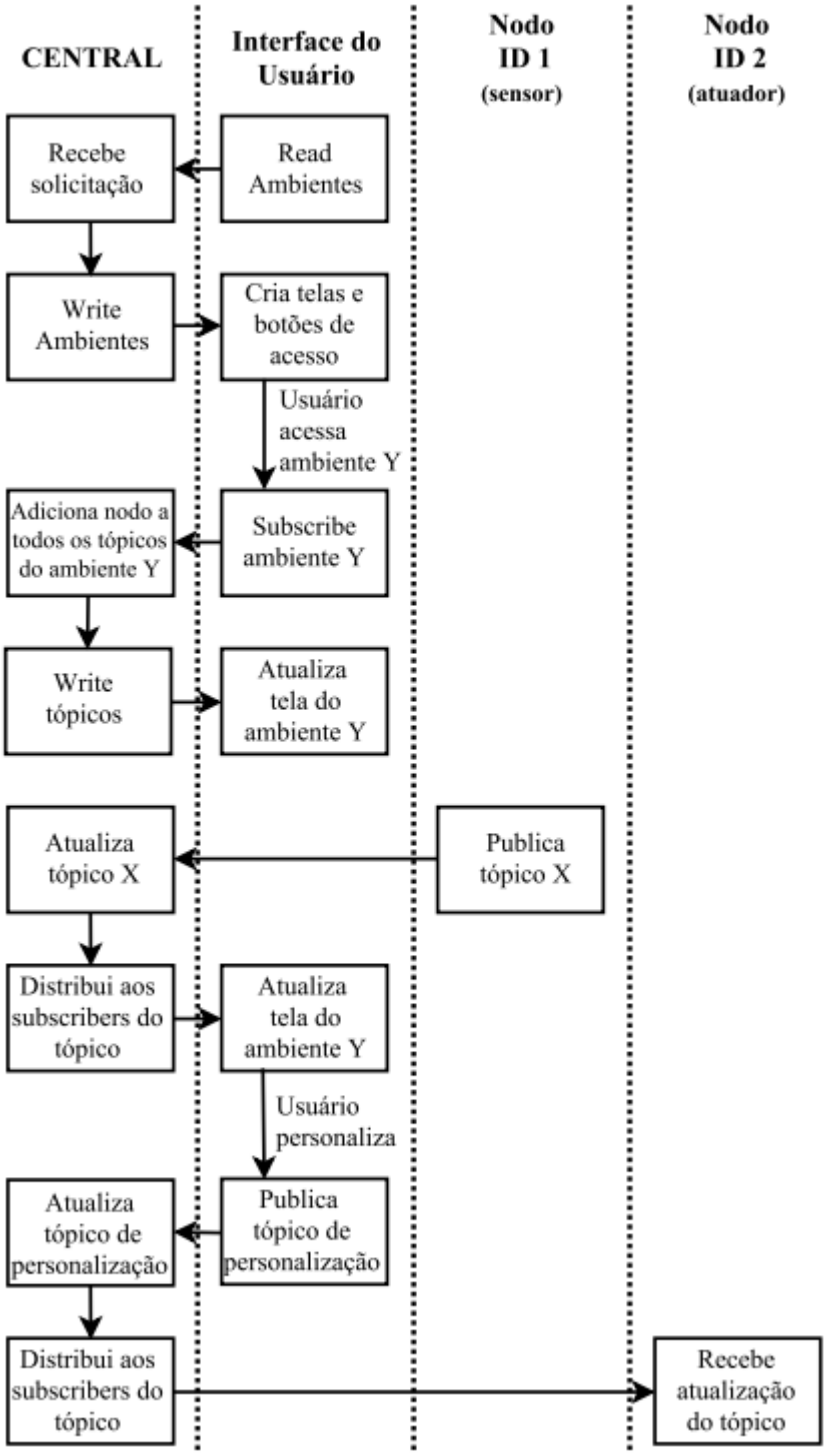
Para que o usuário possa interagir com o sistema optou-se pelo desenvolvimento de uma aplicação para *smartphone*, de forma a permitir ao usuário acesso às informações de monitoramento e controle do sistema a qualquer momento. Para este fim foi utilizada a linguagem de programação Python e a biblioteca Kivy (KIVY, 2017) para desenvolvimento de *graphical user interfaces* (GUI). A biblioteca Kivy permite o desenvolvimento *cross plataform* de aplicações, ou seja, o mesmo código pode ser utilizado para diferentes sistemas operacionais, tais como Android e iOS.

Este aplicativo deve permitir que o usuário monitore variáveis dos ambientes da residência, tais como a temperatura, umidade e luminosidade obtidas pelo dispositivo sensor desenvolvido. Deve permitir também que o usuário controle dispositivos atuadores, como o dispositivo para controle de potência AC desenvolvido neste projeto possibilitando a dimerização de lâmpadas.

O aplicativo desenvolvido cria as telas e *widgets* em *runtime*, ou seja, as telas são criadas durante o uso do aplicativo pelo usuário conforme a comunicação com a central baseando-se nas classes e tópicos do protocolo de comunicação. Isto permite que, quando da expansão do sistema, para outros ambientes da residência por exemplo, com adesão de novos dispositivos, o aplicativo se adapte e se atualize automaticamente sem a necessidade de reprogramação do mesmo.

O fluxograma da Figura 4.19 apresenta um exemplo básico do funcionamento e interação do aplicativo desenvolvido com o sistema.

Figura 4.19 – Exemplo de interação da interface do usuário com o sistema.

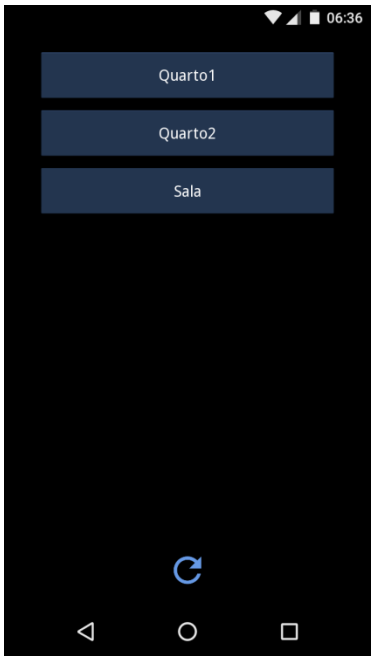


Fonte: Autor.

Como pode ser visto na Figura 4.19 primeiramente o aplicativo envia à central uma solicitação (instrução *Read*) de leitura dos ambientes, ou seja, é solicitado o envio das classes do protocolo de comunicação, já que neste sistema as classes do protocolo representam os diversos ambientes da residência. A central então envia (instrução *Write*) as informações solicitadas e o aplicativo as utiliza para a criação dos objetos de telas para cada ambiente, e dos botões da tela inicial para que o usuário possa escolher o ambiente que deseja acessar.

A Figura 4.20 apresenta a tela inicial do aplicativo, onde podem ser vistos os botões criados baseados nas classes do protocolo de comunicação que dão acesso às telas dos ambientes da residência. O botão de *refresh* na parte inferior da tela executa um teste da comunicação com a central (instrução *Hello*) e reenvia a solicitação de leitura dos ambientes de forma a verificar se há novos ambientes.

Figura 4.20 – Tela inicial do aplicativo desenvolvido.



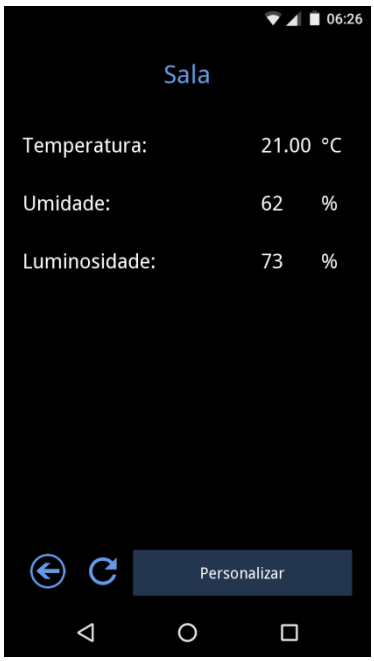
Fonte: Autor.

Os ambientes são armazenados em um arquivo .json de forma que não se faz necessária a solicitação de leitura dos ambientes a cada inicialização do aplicativo. As únicas informações armazenadas são os ambientes e o número ID recebido na identificação.

As atualizações das variáveis de monitoramento mostradas nas telas dos ambientes se dão pela inscrição do aplicativo nos tópicos publicados nestes ambientes. Como pode ser visto na sequencia do fluxograma da Figura 4.19, uma vez que o usuário acessa a tela de um dos ambientes, o aplicativo envia uma solicitação de inscrição (instrução *Subscribe*) para aquele ambiente e a central o adiciona às listas de *subscribers* de todos os tópicos do mesmo. Em seguida a central envia os tópicos e o aplicativo atualiza a tela do ambiente, agora a cada publicação em algum destes tópicos, o aplicativo a receberá e poderá atualizar a tela do ambiente como mostrado na Figura 4.19 quando da publicação do tópico X pelo Nodo 1 pertencente ao ambiente Y.

A Figura 4.21 apresenta a tela de um ambiente onde podem ser vistos os tópicos publicados e seus valores atualizados. O botão de retorno na parte inferior da tela abre a tela inicial e envia a central uma solicitação de saída das listas de *subscribers* (instrução *Unsubscribe*) dos tópicos do ambiente pertinente a esta tela. O botão de *refresh* reenvia a solicitação de inscrição para caso a tela não tenha se atualizado devido alguma falha de comunicação, e o botão Personalizar abre a tela de personalização do ambiente.

Figura 4.21 – Tela do aplicativo para um ambiente.

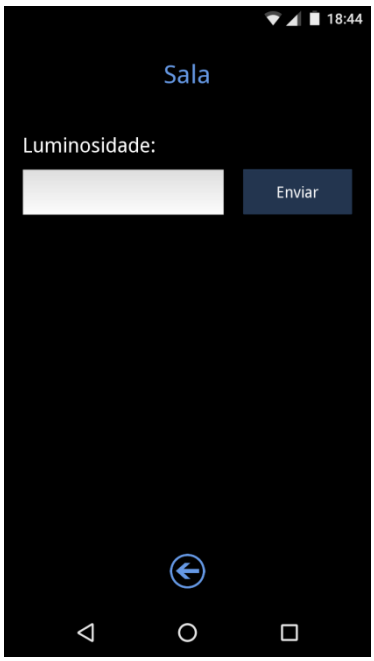


Fonte: Autor.

Para criar as telas de personalização é necessário que dispositivos *subscribers* que queiram receber os dados de personalização do usuário enviem na identificação (como mostrado na seção 4.4.2.2 sobre a instrução *RequestID*) os tópicos a serem publicados pelo usuário. Tais tópicos são indicados pelo prefixo “u_” seguido do nome do tópico para indicar que este tópico deve ser utilizado na criação da tela de personalização para o ambiente (classe) ao qual este dispositivo pertence. Para exemplificar: para que o dispositivo para controle de potência AC desenvolvido receba dados da luminosidade desejada pelo usuário, este deve passar na identificação o tópico “u_luminosidade”, assim quando o usuário acessar a tela do ambiente, e o aplicativo receber os tópicos do mesmo, o tópico “u_luminosidade” não aparecerá na tela de monitoramento do ambiente, mas sim indicará que o usuário pode personalizar esta variável, a qual aparecerá na tela de personalização.

A Figura 4.22 apresenta a tela para personalização de um ambiente. O usuário pode digitar o valor desejado na caixa de texto, o botão Enviar publica o tópico com prefixo “u_” com o valor digitado pelo usuário. Este tópico será atualizado na central e enviado ao dispositivo atuador *subscriber*, como mostrado na sequência do fluxograma da Figura 4.19. O botão de retorno na parte inferior da tela abre novamente a tela de monitoramento do ambiente.

Figura 4.22 – Tela do aplicativo para personalização de um ambiente.



Fonte: Autor.

Geralmente provedores de Internet não oferecem IP estático a usuários domésticos, este serviço se reserva a empresas e custa caro, isto implica num IP dinâmico do roteador na Internet. Para que o usuário possa monitorar e controlar o sistema remotamente, ou seja, para que possa utilizar o aplicativo pela Internet, optou-se pela utilização de um serviço de Dynamic *Domain Name System* (DDNS), este serviço permite a utilização de um *hostname* fixo para o endereçamento, um *software* de atualização instalado no dispositivo *host* monitora mudanças no IP externo e o atualiza no serviço DDNS quando há uma mudança, assim dados enviados ao *hostname* fixo criado são encaminhados ao dispositivo *host* mesmo quando seu IP é alterado.

O serviço DDNS utilizado foi o No-IP (NO-IP, 2017). Este serviço disponibiliza *hosts* gratuitos e *software* de atualização automática compatível com sistemas operacionais Linux, possibilitando assim seu uso com a plataforma Raspberry Pi utilizada como central do sistema. A utilização deste serviço implica na configuração, na Raspberry Pi, de um IP estático na rede doméstica e na abertura, na configuração do roteador, de uma porta associada a este IP para que mensagens encaminhadas ao *hostname* DDNS (que serão redirecionadas ao IP externo atualizado automaticamente) possam ser direcionadas a central do sistema, ou seja, à plataforma Raspberry PI.

4.6 INTEGRAÇÃO DO SISTEMA

Até aqui foram expostos os diferentes componentes do sistema, nesta seção é apresentada a solução desenvolvida para a integração destas partes em um sistema funcional.

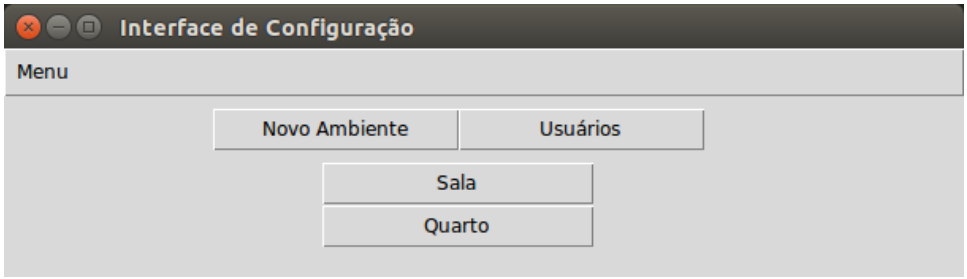
Foi desenvolvida uma interface de configuração do sistema que permite declarar os ambientes da residência, ou seja, criar as classes do protocolo de comunicação, bem como fazer a divisão dos dispositivos entre estes. Isto possibilita que os dispositivos não necessariamente precisem saber em que ambiente estão inseridos, assim dispositivos com o mesmo proposito podem receber a mesma programação de forma que publiquem tópicos iguais, porém poderão atuar em ambientes diferentes, já que são separados em classes.

O programa da interface de configuração e o programa que atua como *broker* (responsável pela comunicação do sistema) possuem as mesmas classes que descrevem os ambientes e dispositivos. Isto permite que as instâncias criadas na configuração possam ser serializadas e utilizadas pelo *broker*, assim é necessário fazer a configuração apenas uma vez,

quando da instalação do sistema, ou a cada vez que o sistema for expandido adicionando-se novos nodos e ambientes. Estes programas foram implementados com a linguagem de programação Python, para a serialização dos objetos foi utilizado o módulo Pickle e a interface foi implementada utilizando o módulo Tkinter de desenvolvimento de GUI (PYTHON, 2017).

A Figura 4.23 apresenta a tela inicial da interface de configuração. O botão Novo Ambiente dá acesso à tela de entrada dos ambientes, a cada classe do protocolo criada, um novo botão é adicionado à tela inicial que dá acesso a tela de configuração do ambiente como mostrado na figura os botões para os ambientes Sala e Quarto.

Figura 4.23 – Tela inicial da interface de configuração do sistema.

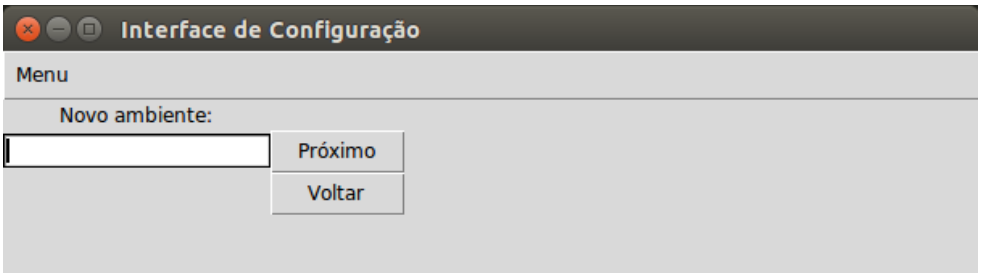


Fonte: Autor.

O botão Usuários é referente a uma classe do protocolo intrínseca do sistema que não contem dispositivos dos ambientes, mas sim os dispositivos *smartphone* com interfaces de usuários (esta classe também é passada ao aplicativo quando este solicita a leitura das classes, porém é ignorada no momento da criação das telas de ambientes). Este botão abre a tela que permite a identificação destes dispositivos. O menu superior apresenta a opção Salvar que chama a função de serialização dos objetos, este menu pode ser acessado em todas as telas.

A Figura 4.24 apresenta a tela de entrada dos nomes dos ambientes. O nome pode ser digitado na caixa de entrada e o botão Próximo cria uma nova instância de classe do protocolo e a tela para identificação dos dispositivos daquele ambiente.

Figura 4.24 – Tela para entrada dos nomes dos ambientes.



Fonte: Autor.

A Figura 4.25 apresenta a tela utilizada na identificação dos dispositivos do ambiente. O botão Encontra Dispositivos inicia a rotina apresentada no fluxograma da Figura 4.15 sobre a identificação de dispositivos, ou seja, envia em *broadcast* uma mensagem com a instrução *Hello* e *node* ID igual a zero, trata as respostas (intruções *RequestID*) criando novas instâncias de nodos e envia os números de identificação a cada dispositivo. Depois de criadas as instâncias, os dados destes dispositivos são mostrados na tela para que haja *feedback* visual do sucesso ou fracasso da correta identificação de todos os dispositivos.

Figura 4.25 – Tela utilizada na identificação dos dispositivos de um ambiente.

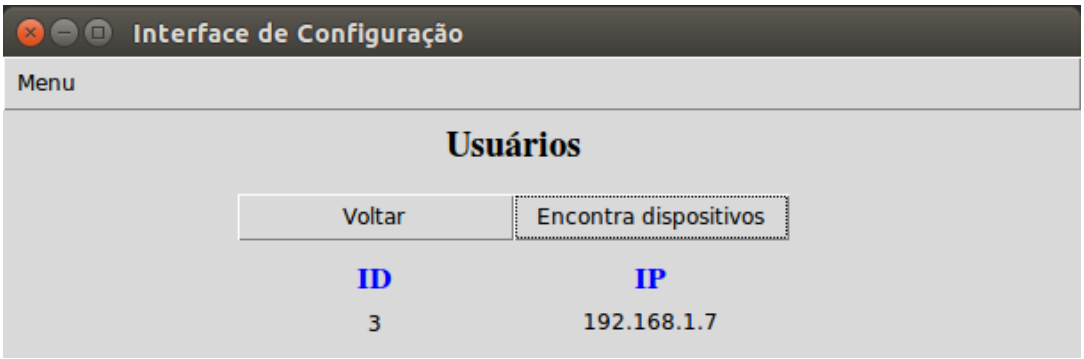
Fonte: Autor.

É importante notar que este método de identificação dos dispositivos implica em uma particularidade do sistema. Os dispositivos de cada ambiente devem ser alimentados apenas quando a identificação daquele ambiente for feita, caso contrário todos os dispositivos do

sistema responderiam a instrução *Hello* enviada e seriam adicionados ao mesmo ambiente. Ainda, dispositivos devem estar previamente conectados à rede doméstica utilizada pelo sistema, isto se dá pela programação do nome da rede (SSID) e senha nos módulos ESP-12F.

A Figura 4.26 apresenta a tela utilizada na identificação de dispositivos de interface com o usuário. O botão Encontra Dispositivo tem a mesma função apresentada na tela dos ambientes, porém aqui os dispositivos são adicionados à classe Usuário do protocolo. Os aplicativos não enviam tópicos, pois estes variam durante a execução como mostrado na seção 4.5 sobre interface com o usuário. Esta tela também apresenta *feedback* visual.

Figura 4.26 – Tela utilizada na identificação de dispositivos de interface com o usuário.



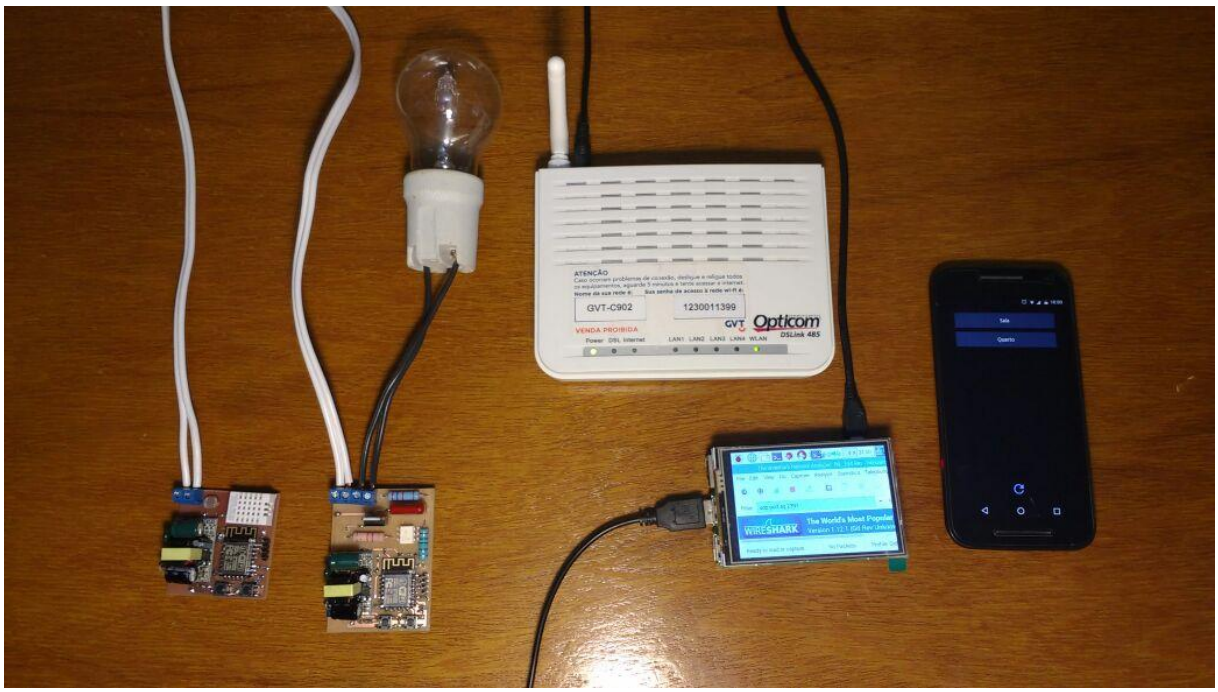
Fonte: Autor.

5 RESULTADOS

5.1 TESTES REALIZADOS

Esta seção apresenta testes realizados no sistema desenvolvido. A Figura 5.1 apresenta o setup de testes utilizado, que consiste dos dois dispositivos desenvolvidos atuando como nodos *publisher* e *subscriber* com a lâmpada a ser dimerizada, a plataforma Raspberry Pi 3 atuando como central do sistema, um *smatphone* com o aplicativo desenvolvido para interface com o usuário, e um roteador Wi-Fi para implementação da rede. Serão apresentadas as capturas dos pacotes da comunicação, feitas utilizando o *software* para análise de rede Wireshark com o objetivo de visualizar a troca de mensagens entre os dispositivos.

Figura 5.1 – Setup de teste.



Fonte: Autor.

Nos testes realizados o dispositivo sensor atua como nodo *publisher*, este publica três tópicos referentes a leitura de temperatura, umidade e luminosidade dos sensores. Já o dispositivo para controle de potência AC atua como nodo *subscriber* dos tópicos “luminosidade” e “u_luminosidade”, ou seja, este recebe atualizações dos tópicos publicados tanto pelo dispositivo sensor, quanto pelo usuário.

O primeiro teste realizado foi a cerca da identificação de dispositivos. Utilizando a interface de configuração do sistema desenvolvida, foi criado um ambiente e então com ambos dispositivos alimentados foi executada a rotina de identificação. A Figura 5.2 apresenta a captura das mensagens trocadas. Quando do envio pela cental (IP 192.168.1.201) da instrução *Hello* em *broadcast* ambos dispositivos respondem com a solicitação de identificação, primeiramente o dispositivo para controle de potência AC (IP 192.168.1.6) e em seguida o dispositivo sensor (IP 192.168.1.5). A central então os identifica, cria as instâncias de nodo de cada um, e envia os novos números ID recebendo a confirmação de recebimento dos dois dispositivos.

Figura 5.2 – Captura de pacotes na identificação dos dispositivos.

Time	Source	Destination	Protocol	
27.916930000	192.168.1.201	255.255.255.255	UDP	⇦ **Hello** enviado em broadcast
28.143302000	192.168.1.6	192.168.1.201	UDP	⇦ **Request ID** (Subscriber)
28.148314000	192.168.1.5	192.168.1.201	UDP	⇦ **Request ID** (Publisher)
28.649917000	192.168.1.201	192.168.1.6	UDP	⇦ Central envia novo ID
28.663070000	192.168.1.6	192.168.1.201	UDP	⇦ **Acknowledge**
28.663381000	192.168.1.201	192.168.1.5	UDP	⇦ Central envia novo ID
28.673141000	192.168.1.5	192.168.1.201	UDP	⇦ **Acknowledge**

Fonte: Autor.

Feita a identificação dos dispositivos foi possível testar a comunicação, para tanto foi feita a serialização das instancias criadas através do menu Salvar da interface de configuração e em seguida deu-se inicio à execução do *broker*. A Figura 5.3 apresenta a captura das mensagens trocadas quando da publicação do tópico “luminosidade” pelo dispositivo p*ublisher*.

Figura 5.3 – Captura de pacotes na publicação de tópicos.

Time	Source	Destination	Protocol	
6.565897000	192.168.1.5	192.168.1.201	UDP	⇦ **Publish**
6.666628000	192.168.1.201	192.168.1.5	UDP	⇦ **Acknowledge**
6.666825000	192.168.1.201	192.168.1.6	UDP	⇦ **Write**
6.672144000	192.168.1.6	192.168.1.201	UDP	⇦ **Acknowledge**

Fonte: Autor.

Depois de recebida a publicação pelo dispositivo p*ublisher*, a central confirma o recebimento, e após verificar a lista de nodos inscritos no tópico, envia a atualização ao dispositivo *subscriber* que responde com a confirmação de recebimento.

A Figura 5.4 apresenta a captura dos pacotes trocados entre o aplicativo de interface com o usuário e o sistema.

Figura 5.4 – Captura de pacotes na interação sistema/usuário.

Time	Source	Destination	Protocol	
7.905478000	192.168.1.7	192.168.1.201	UDP	⇦ **Read** Ambientes
8.006144000	192.168.1.201	192.168.1.7	UDP	⇦ **Write** Ambientes
11.939313000	192.168.1.7	192.168.1.201	UDP	⇦ **Subscribe**
12.040239000	192.168.1.201	192.168.1.7	UDP	⇦ **Write** tópicos
12.040374000	192.168.1.201	192.168.1.7	UDP	⇦ } **Write** ultima
12.040473000	192.168.1.201	192.168.1.7	UDP	⇦ } atualização dos
12.040558000	192.168.1.201	192.168.1.7	UDP	⇦ } tópicos
20.525897000	192.168.1.5	192.168.1.201	UDP	⇦ **Publish** 1° tópico
20.626761000	192.168.1.201	192.168.1.5	UDP	⇦ **Acknowledge**
20.627010000	192.168.1.201	192.168.1.7	UDP	⇦ **Write** 1° tópico
20.784695000	192.168.1.7	192.168.1.201	UDP	⇦ **Acknowledge**
21.522290000	192.168.1.5	192.168.1.201	UDP	⇦ **Publish** 2° tópico
21.622884000	192.168.1.201	192.168.1.5	UDP	⇦ **Acknowledge**
21.623026000	192.168.1.201	192.168.1.7	UDP	⇦ **Write** 2° tópico
21.805027000	192.168.1.7	192.168.1.201	UDP	⇦ **Acknowledge**
22.520658000	192.168.1.5	192.168.1.201	UDP	⇦ **Publish** 3° tópico
22.621405000	192.168.1.201	192.168.1.5	UDP	⇦ **Acknowledge**
22.621571000	192.168.1.201	192.168.1.6	UDP	⇦ **Write** 3° tópico
22.626436000	192.168.1.6	192.168.1.201	UDP	⇦ **Acknowledge**
22.626841000	192.168.1.201	192.168.1.7	UDP	⇦ **Write** 3° tópico
22.830548000	192.168.1.7	192.168.1.201	UDP	⇦ **Acknowledge**
24.505208000	192.168.1.7	192.168.1.201	UDP	⇦ **Unsubscribe**
24.606004000	192.168.1.201	192.168.1.7	UDP	⇦ **Acknowledge**
31.872288000	192.168.1.7	192.168.1.201	UDP	⇦ **Publish** personalização
31.973095000	192.168.1.201	192.168.1.7	UDP	⇦ **Acknowledge**
31.973332000	192.168.1.201	192.168.1.6	UDP	⇦ **Write** personalização
31.979695000	192.168.1.6	192.168.1.201	UDP	⇦ **Acknowledge**

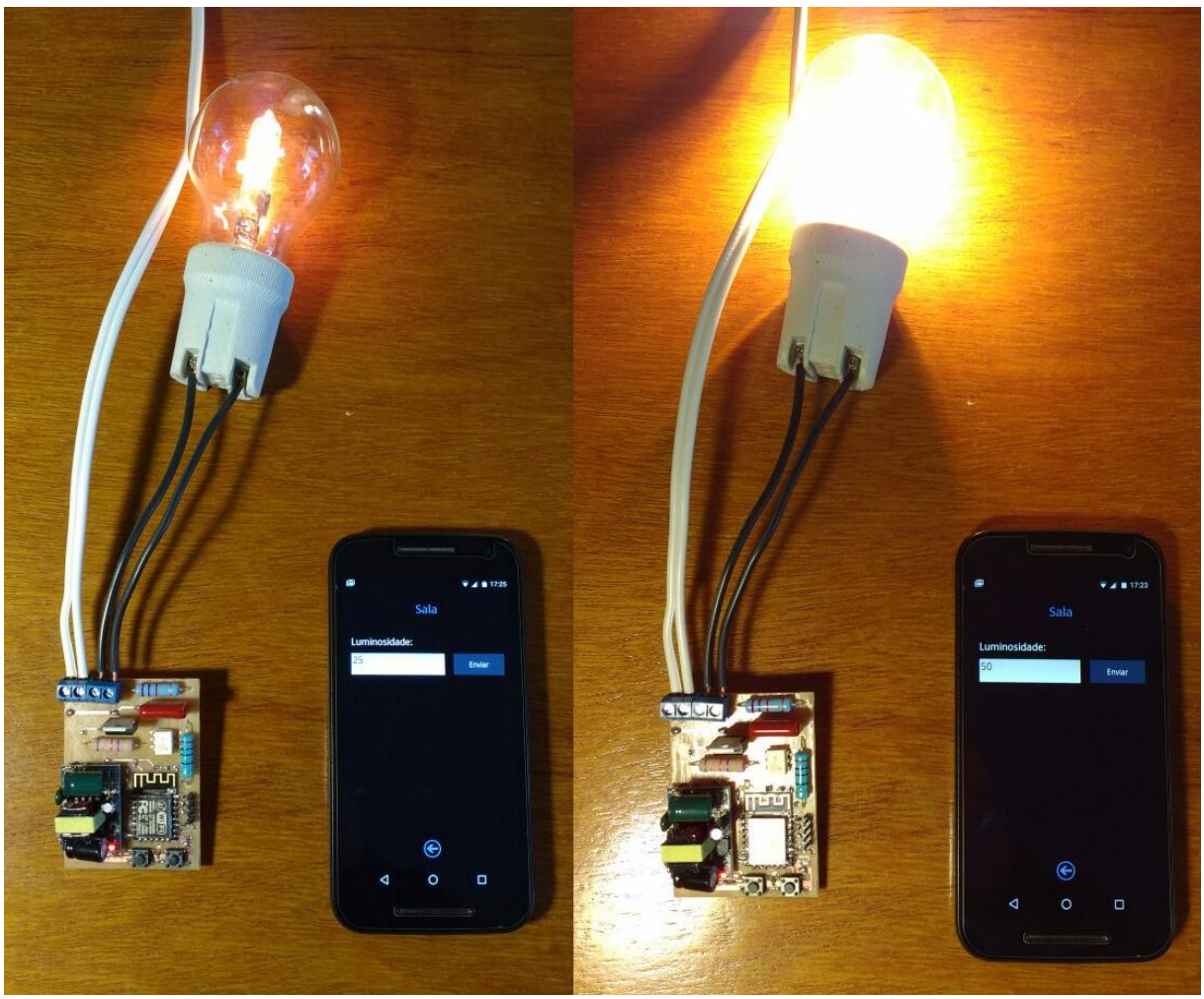
Fonte: Autor.

No inicio o aplicativo (IP 192.168.1.7) solicita a leitura dos ambientes, e quando recebe a resposta cria a tela do ambiente declarado na interface de configuração. Em seguida o usuário acessa a tela do ambiente, o que resulta no envio da solicitação de inscrição à central que responde com os tópicos bem como os valores da ultima atualização de cada um. Quando o dispositivo sensor publica os tópicos de temperatura e umidade, estes são enviados

apenas ao aplicativo, já o tópico de luminosidade (3° tópico na figura) é enviado também ao dispositivo de controle de potência AC. Em seguida o usuário vai para a tela de personalização, a saída da tela de monitoramento resulta no envio da solicitação *Unsubscribe*. A personalização feita pelo usuário resulta na publicação do tópico "u_luminosidade" que é enviado ao dispositivo de controle de potência AC para atuação.

A Figura 5.5 apresenta o resultado da atuação do dispositivo *subscriber* para controle de potência AC, quando do envio pelo usuário das informações de personalização do ambiente, ou seja, quando da publicação do tópico "u_luminosidade". O usuário deve entrar com um valor entre 0% e 100%, na figura é possível ver a atuação quando da entrada pelo usuário do valor 25% e 50% respectivamente a esquerda e a direita da figura.

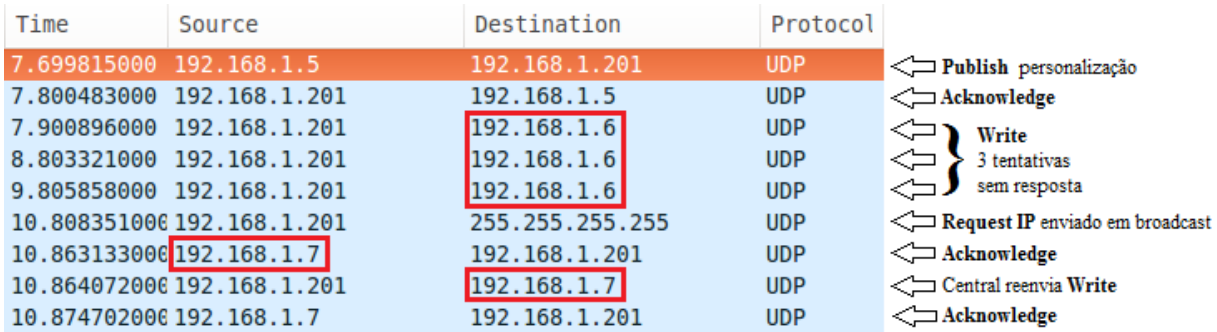
Figura 5.5 – Atuação do dispositivo *subscriber* baseada na personalização do ambiente pelo usuário.



Fonte: Autor.

Para testar a atualização do endereço IP dos dispositivos o roteador foi reiniciado, isto resultou na troca do endereço IP dos dispositivos como mostrado na Figura 5.6. Quando o aplicativo envia a atualização do tópico “u_luminosidade” a central atualiza seu endereço IP, porém quando tenta enviar a atualização ao dispositivo *subscriber* não recebe a confirmação já que seu IP foi alterado, a central então envia em *broadcast*, especificando o numero ID do dispositivo *subscriber*, a instrução *Request ID*. Em seguida recebe a confirmação de recebimento e pode atualizar o IP do dispositivo e reenviar a mensagem.

Figura 5.6 – Captura de pacotes na atualização do endereço IP de dispositivos.



Fonte: Autor.

Por fim foi testada a comunicação com o usuário pela Internet. Como pode ser visto na Figura 5.7 a comunicação funciona normalmente quando da utilização pelo aplicativo do *hostname* criado no serviço DDNS para endereçar mensagens à central. A central foi capaz de atualizar o endereço do dispositivo do usuário e distribuir corretamente todas as publicações.

A Figura 5.8 apresenta detalhes de uma mensagem, neste caso quando do envio de uma atualização do tópico “umidade” a um nodo *subscriber*. Como pode ser visto o pacote apresenta, incluindo os *frames* das camadas de transporte, rede e dados, um total de 64 bytes. O campo de dados, ou seja, a mensagem padronizada pelo protocolo de comunicação apresenta 22 bytes e seu conteúdo pode ser visto na parte inferior da figura em sua representação hexadecimal.

Figura 5.7 – Captura de pacotes na comunicação pela internet com o aplicativo.

Time	Source	Destination	Protocol	
4.266507000	179.204.231.234	192.168.1.201	UDP	⇦ **Read** Ambientes
4.367160000	192.168.1.201	179.204.231.234	UDP	⇦ **Write** Ambientes
10.595083000	179.204.231.234	192.168.1.201	UDP	⇦ **Subscribe**
10.696007000	192.168.1.201	179.204.231.234	UDP	⇦ **Write** tópicos
10.696147000	192.168.1.201	179.204.231.234	UDP	⇦ } **Write** ultima
10.696241000	192.168.1.201	179.204.231.234	UDP	⇦ } atualização dos
10.696328000	192.168.1.201	179.204.231.234	UDP	⇦ } tópicos
13.677487000	192.168.1.2	192.168.1.201	UDP	⇦ **Publish** tópico
13.778195000	192.168.1.201	192.168.1.2	UDP	⇦ **Acknowledge**
13.778394000	192.168.1.201	192.168.1.7	UDP	⇦ **Write** tópico
13.786427000	192.168.1.7	192.168.1.201	UDP	⇦ **Acknowledge**
13.786800000	192.168.1.201	179.204.231.234	UDP	⇦ **Write** tópico
14.139018000	179.204.231.234	192.168.1.201	UDP	⇦ **Acknowledge**
17.563068000	179.204.231.234	192.168.1.201	UDP	⇦ **Unsubscribe**
17.663833000	192.168.1.201	179.204.231.234	UDP	⇦ **Acknowledge**
22.563653000	179.204.231.234	192.168.1.201	UDP	⇦ **Publish** personalização
22.664144000	192.168.1.201	179.204.231.234	UDP	⇦ **Acknowledge**
22.664229000	192.168.1.201	192.168.1.7	UDP	⇦ **Write** personalização
22.667361000	192.168.1.7	192.168.1.201	UDP	⇦ **Acknowledge**

Fonte: Autor.

Figura 5.8 – Conteúdo de uma mensagem.

```
▶Frame 616: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface 0
▶Ethernet II, Src: IntelCor_3e:70:74 (ac:72:89:3e:70:74), Dst: 88:79:7e:9f:a3:98 (88:79:7e:9f:a3:98)
▶Internet Protocol Version 4, Src: 192.168.1.201 (192.168.1.201), Dst: 192.168.1.6 (192.168.1.6)
▶User Datagram Protocol, Src Port: 2391 (2391), Dst Port: 2391 (2391)
▼Data (22 bytes)
   Data: 390185756d6964616465202020202020202037302e33
   [Length: 22]

0000  88 79 7e 9f a3 98 ac 72  89 3e 70 74 08 00 45 00   .y~....r .>pt..E.
0010  00 32 20 39 40 00 40 11  96 62 c0 a8 01 c9 c0 a8   .2 9@.@. .b......
0020  01 06 09 57 09 57 00 1e  8e 59 39 01 85 75 6d 69   ...W.W.. .Y9..umi
0030  64 61 64 65 20 20 20 20  20 20 20 20 37 30 2e 33   dade         70.3
```

Fonte: Autor.

5.2 CUSTOS

A Tabela 4 apresenta os custos dos componentes do sistema, considerando taxas de importação de 60% e ICMS de 12%, para a central e componentes utilizados nos dispositivos. Como pode ser visto o sistema apresentou baixo custo.

Tabela 4 – Custos.

Componente	**R$**
Raspberry Pi 3	220,79
Dispositivo para controle de potência AC	35,00
Dispositivo sensor	40,00

Fonte: Autor.

6 CONCLUSÃO

O sistema desenvolvido se apresentou como uma boa solução por não utilizar infraestrutura cabeada, mas sim utilizar uma interface de comunicação Wi-Fi que se encontra disponível na maioria das residências com acesso a Internet. Isto aliado ao *hardware* de baixo custo utilizado possibilitou um sistema final de custo reduzido.

Conseguiu-se desenvolver um protocolo de comunicação com uma boa confiabilidade ao reproduzir características do protocolo TCP. Capacidades implementadas como identificação de nodos e mensagens em *broadcast* possibilitam uma fácil adesão de novos dispositivos, tornando assim o sistema facilmente expansível. O protocolo possibilita ainda que o sistema seja implantado na residência utilizando a infraestrutura de rede sem fio já existente sem a necessidade de alterações da configuração da mesma, uma vez que não se faz necessária a utilização de IP estático.

Outra característica importante que torna o sistema facilmente expansível é a capacidade do aplicativo desenvolvido de se adaptar as mudanças do sistema, dispensando a necessidade de reprogramação do mesmo quando houver alterações no sistema.

A interface de configuração desenvolvida torna simples a integração do sistema, facilitando sua instalação. Possibilita ainda, aliada às características do protocolo de comunicação, uma maneira simples de classificação dos dispositivos de forma que dispositivos com mesmo proposito possam operar em ambientes diferentes sem necessidade de receber uma programação diferente e sem necessidade de conhecimento prévio do *layout* da residência.

REFERÊNCIAS

AI-THINKER. **ESP-12F WiFi Module**. Versão 1.0. 2015. Disponível em: <https://www.elecrow.com/download/ESP-12F.pdf>. Acesso em 15/04/2017.

BANKS, Andrew; GUPTA, Rahul. **MQTT Version 3.1. 1. OASIS standard**, 2014. Disponível em: < http://mqtt.org/documentation>. Acesso em 14/10/2016.

BOLZANI, Caio Augustus Morais. **Residências inteligentes**. Editora Livraria da Física, 2004.

ESPRESSIF SYSTEMS. **ESP8266EX Datasheet.** Versão 5.4. 2017. Disponível em: <https://espressif.com/en/products/hardware/esp8266ex/resources>. Acesso em 15/04/2017.

GREHS, Daniel Henrique. **Sistema de irrigação doméstico baseado em Internet das Coisas.** 2016. Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/147673/000999748.pdf?sequence=1>. Acesso em 10/09/2016.

KIVY Documentation. Release 1.10.1.dev0. Disponível em: <https://media.readthedocs.org/pdf/kivy/latest/kivy.pdf>. Acesso em: 10/05/2017.

MARTINS, Ismael Rodrigues; ZEM, José Luís. **Estudo dos Protocolos de Comunicação MQTT e CoAP para Aplicações Machine-to-Machine e Internet das Coisas.** Revista Tecnológica da Fatec Americana, Volume 1, Número 1, 2015.

MICRIUM EMBEDDED SOFTWARE. **Designing the Internet of Things**. 2015. Disponível em: < https://www.micrium.com/iot/overview/ >. Acesso em 02/09/2016.

NO-IP Website. Disponível em: <www.noip.com>. Acesso em: 13 abr. 2017.

PYTHON 3.4 Documentation. Diponível em: <https://docs.python.org/3.4/>. Acesso em: 13/01/2017.

RIBEIRO, Tiago José de Araújo; OLIVEIRA, Sérgio Campello. **Projeto de um Gateway para Automação Residencial.** Revista de Engenharia e Pesquisa Aplicada, Volume 1, Número 1, 2016.

SANTOS, Bruno P. **Internet das Coisas: da Teoria à Prática.** 2016. Disponível em: <http://homepages.dcc.ufmg.br/~mmvieira/cc/papers/>. Acesso em 07/09/2016.

SHELBY, Zach; HARTKE, Klaus; BORMANN, Carsten. **The constrained application protocol (CoAP)**. 2014. Disponível em: < http://coap.technology/spec.html>. Acesso em 14/10/2016.

VASSEUR, Jean-Philippe; DUNKELS, Adam**. Interconnecting smart objects with ip: The next internet.** Morgan Kaufmann Publishers, 2010.

ANEXO A – CÓDIGO DO BROKER

```
import socket
import _pickle as pickle

class Nodos(object):
	def __init__(self, ID, IP, tipo, topicos, ambiente):
		self.ID = ID
		self.IP = IP
		self.tipo = tipo
		self.topicos = topicos
		self.ambiente = ambiente
		self.mID = 0

	def send_to_node(self, mensagem):
		s.sendto(mensagem, (self.IP, PORTA))

	novas_mensagens = []
	novos_enderecos = []

	def aguarda_resp(self, instr, tpc=None):
		cont = 0
		while True:
			try:
				msg, addr = s.recvfrom(128)
				if addr[0] != LOCAL:
					if self.check_mID(msg[0]) is True:
						if addr[0] != self.IP:
							if instr == 2 and msg[1] == self.ID:
								self.IP = addr[0]
								return True
								break
							else:
								self.novas_mensagens.append(msg)
								self.novos_enderecos.append(addr)
						elif msg[2] != instr:
							self.novas_mensagens.append(msg)
							self.novos_enderecos.append(addr)
						elif instr == 3:
							return True
							break
						elif instr == 5 and trimmer(msg[3:18].decode(), ' ') == tpc:
							return float(trimmer(msg[18:25].decode(), '\x00'))
							break
			except socket.timeout:
				cont += 1
			if cont == 10:
				return False
				break

	def request_ip(self): # Instrução 2. [mID | nID | conf/Instr]
		mensagem = bytes((self.new_mID(), self.ID, 0x02))
		b.sendto(mensagem, ('<broadcast>', PORTA))
		if self.aguarda_resp(2) is True:
			serializa('nodos.txt', nodos)
			return True
		else: return False
```

```
        def Acknowledge(self):
                mensagem = bytes((self.new_mID(), self.ID, 0x03))
                self.send_to_node(mensagem)

        def check_mID(self, new):
                if self.mID == 255 and new in range(1,10):
                        self.mID = new
                        return True
                elif new <= self.mID:
                        return False
                else:
                        self.mID = new
                        return True

        def new_mID(self):
                if self.mID == 255:
                        self.mID = 0
                        return self.mID
                else:
                        self.mID += 1
                        return self.mID


class Ambientes(object):
        def __init__(self, nome, nodos, topicos):
                self.nome = nome
                self.nodos = nodos
                self.topicos = topicos


def serializa(file, dicion):
        with open(file,'wb') as f:
                for i in range(1,len(dicion)+1):
                        pickle.dump(dicion[i], f, -1)

def deserializa(file,dicion):
        with open(file,'rb') as f:
                i = 1
                while True:
                        try:
                                dicion[i] = pickle.load(f)
                                i = i+1
                        except EOFError:
                                break

def send_broadcast(mensagem):
        b.sendto(mensagem, ('<broadcast>', PORTA))

def trimmer(str, comp):
        output = []
        for i in str:
                if i != comp:
                        output.append(i)
        return ''.join(output)

def grower(str):
        output = []
        output.append(str)
        for i in range(15-len(str)):
                output.append(' ')
```

```
        return ''.join(output)

def hello(nID, msg):
        nodos[nID].mID = 0
        if msg[2] > 100:
                nodos[nID].Acknowledge()

def request_id(nID, msg):
        pass

def request_ip(nID, msg):
        pass

def ackn(nID, msg):
        pass

def read(nID, msg): # Instrução 4. [mID | nID | conf/Instr | informacao | classe ]
        info = trimmer(msg[3:18].decode(), ' ')
        if info == 'ambientes':
                ambs = bytes()
                for i in ambientes:
                        ambs += bytes(grower(ambientes[i].nome), 'ascii')

                header = bytes((nodos[nID].new_mID(), nID, 0x05))
                lenth = bytes([len(ambientes)])
                pl = lenth + ambs
                mensagem = header + pl
                nodos[nID].send_to_node(mensagem)

def write(nID, msg): # Instrução 5. [mID | nID | conf/Instr | informacao | conteudo]
        pass

def publish(nID, msg): # Instrução 6. [mID | nID | conf/Instr | Ambiente | Topico | conteudo]
        amb = trimmer(msg[3:18].decode(), ' ')
        topic = trimmer(msg[18:33].decode(), ' ')
        value = float(trimmer(msg[33:len(msg)].decode(), '\x00'))

        if amb == '':
                amb = nodos[nID].ambiente

        globals()[amb][topic] = value

        for i in getattr(ambientes[amb], topic):
                header = bytes((nodos[i].new_mID(), nodos[i].ID, 133)) # Write
                topico = bytes(grower(topic),'ascii')
                valor = bytes(str(value),'ascii')
                mensagem = header + topico + valor
                for j in range(3):
                        nodos[i].send_to_node(mensagem)
                        resp = nodos[i].aguarda_resp(3)
                        if resp is True:
                                break
                else:
                        reqIP = nodos[i].request_ip()
                        if reqIP is True:
                                header = bytes((nodos[i].new_mID(), nodos[i].ID, 133))
                                mensagem = header + topico + valor
                                nodos[i].send_to_node(mensagem)
```

```
def subscribe(nID, msg): # Instrução 7. [mID | nID | conf/Instr | Tópico | Ambiente]
	topic = trimmer(msg[3:18].decode(), ' ')
	amb = trimmer(msg[18:33].decode(), ' ')

	if topic == '':
		topicos = bytes()
		for i in range(len(ambientes[amb].topicos)):
			topic = ambientes[amb].topicos[i]
			topicos += bytes(grower(topic), 'ascii')
			if nID not in getattr(ambientes[amb], topic):
				getattr(ambientes[amb], topic).append(nID)

		header = bytes((nodos[nID].new_mID(), nID, 0x05))
		mensagem = header + topicos
		nodos[nID].send_to_node(mensagem)

		for topic in globals()[amb]:
			if topic[0:2] != 'u_':
				header = bytes((nodos[nID].new_mID(), nID, 0x05))
				topico = bytes(grower(topic),'ascii')
				valor = bytes(str(globals()[amb][topic]),'ascii')
				mensagem = header + topico + valor
				nodos[nID].send_to_node(mensagem)


	else:
		if amb == '':
			amb = nodos[nID].ambeinte

		if nID not in getattr(ambientes[amb], topic):
				getattr(ambientes[amb], topic).append(nID)

		header = bytes((nodos[nID].new_mID(), nID, 0x05))
		topico = msg[3:18]
		valor = (globals()[ambiente][topic])
		mensagem = header + topico
		if valor != None:
			vlr = bytes(str(valor),'ascii')
			mensagem += vlr
		nodos[nID].send_to_node(mensagem)

def unsubscribe(nID, msg): # Instrução 8. [mID | nID | conf/Instr | Tópico | Ambiente]
	topic = trimmer(msg[3:18].decode(), ' ')
	amb = trimmer(msg[18:33].decode(), ' ')
	if topic == '':
		for i in range(len(ambientes[amb].topicos)):
			topic = ambientes[amb].topicos[i]
			try:
				getattr(ambientes[amb], topic).remove(nID)
			except:
				pass

	else:
		if amb == '':
			amb = nodos[nID].ambeinte

		try:
			getattr(ambientes[amb], topic).remove(nID)
		except:
```

```
                        pass


def listen():
        lista_mensagens = Nodos.novas_mensagens
        lista_enderecos = Nodos.novos_enderecos
        Nodos.novas_mensagens = []
        Nodos.novos_enderecos = []
        while True:
                try:
                        msg, addr = s.recvfrom(128)
                        if addr[0] != LOCAL:
                                lista_mensagens.append(msg)
                                lista_enderecos.append(addr[0])
                except socket.timeout:
                        break
        for i in range(len(lista_mensagens)):
                try:
                        msg = lista_mensagens[i]
                        addr = lista_enderecos[i]
                        nID = msg[1]

                        if(nodos[nID].IP != addr):
                                nodos[nID].IP = addr
                                serializa('nodos.txt', nodos)

                        if nodos[nID].check_mID(msg[0]) is True:
                                if msg[2] > 100: # confirmacao necessaria
                                        nodos[nID].Acknowledge()
                                        instrucao = msg[2]-128
                                else:  # confirmacao desnecessaria
                                        instrucao = msg[2]

                                instr[instrucao](nID, msg)

                except Exception as e:
                        print(e)

#### INICIALIZAÇÃO ####
PORTA = 2391

nodos = {}
amb = {}
ambientes = {}

brkID = 255
LOCAL = [(l.connect(('8.8.8.8', 53)), l.getsockname()[0], l.close()) for l in [socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)]][0][1]

instr = {0 : hello,
         1 : request_id,
         2 : request_ip,
         3 : ackn,
         4 : read,
         5 : write,
         6 : publish,
         7 : subscribe,
         8 : unsubscribe}

try:
```

```
        b = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        b.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.bind(('', PORTA))
        s.settimeout(0.1)

except Exception as e:
        print(e)

deserializa('nodos.txt', nodos)
deserializa('ambientes.txt', amb)

for i in amb:
        ambientes[amb[i].nome] = amb[i]

for i in nodos:
        nodos[i].mID = 0

for i in ambientes:
        novo_dicionario = {}
        for j in range(len(ambientes[i].topicos)):
                novo_dicionario[ambientes[i].topicos[j]] = None
        globals()[ambientes[i].nome] = novo_dicionario

        for j in range(len(ambientes[i].nodos)):
                if nodos[ambientes[i].nodos[j]].tipo == 'publisher':
                        for k in nodos[ambientes[i].nodos[j]].topicos:
                                setattr(ambientes[i], k, [])
                                for l in range(len(ambientes[i].nodos)):
                                        if nodos[ambientes[i].nodos[l]].tipo == 'subscriber':
                                                for m in nodos[ambientes[i].nodos[l]].topicos:
                                                        if m == k:
                                                                getattr(ambientes[i],
k).append(ambientes[i].nodos[l])
                                                        if m[0:2] == 'u_':
                                                                try:
                                                                        if ambientes[i].nodos[l] not in
getattr(ambientes[i], m):
                                                                                getattr(ambientes[i],
m).append(ambientes[i].nodos[l])
                                                                except AttributeError:
                                                                        setattr(ambientes[i], m, [])
                                                                        getattr(ambientes[i],
m).append(ambientes[i].nodos[l])


mensagem = bytes((0, 255, 0))
send_broadcast(mensagem)

print('Executando Broker')

while True:
        try:
                listen()
        except KeyboardInterrupt:
                break

b.close()
s.close()
```