UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO

ALAN GROBÉRIO BRAGA

GERENCIAMENTO E AUTOMAÇÃO SEM FIO DOS SISTEMAS DE UMA RESIDÊNCIA

VITÓRIA – ES
dez/2018

ALAN GROBÉRIO BRAGA

GERENCIAMENTO E AUTOMAÇÃO SEM FIO DOS SISTEMAS DE UMA RESIDÊNCIA

Parte manuscrita do Projeto de Graduação do aluno **Alan Grobério Braga**, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Orientador: Prof. Dr. Antônio Manoel Ferreira Frasson

VITÓRIA – ES
dez/2018

ALAN GROBÉRIO BRAGA

GERENCIAMENTO E AUTOMAÇÃO DOS SISTEMAS DE UMA RESIDÊNCIA

Parte manuscrita do Projeto de Graduação do aluno **Alan Grobério Braga**, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Aprovada em (dia), de (mês) de (ano).

COMISSÃO EXAMINADORA:

_____

Dr. Antônio Manoel Ferreira Frasson
Universidade Federal do Espírito Santo
Orientador

_____

Eng.º José Luiz Borba
Universidade Federal do Espírito Santo
Examinador

_____

Eng.º Francis Araujo dos Santos
Vale S.A.
Examinador

Dedicatória especial a meus pais e avós, que me deram
a vida e os meios para segui-la na caminhada.

"A única grandeza do homem é reconhecer sua pequenez"
O. de Carvalho

Agradecimentos

Agradeço a Deus pelo dom da vida, e a São José por iluminar minha inteligência, muitas vezes em que tudo parecia obscurecido. A Maria Santíssima que não cessa de derramar as graças de Deus sobre mim, à medida que me é necessário.

Agradecimentos especiais a Jésus, com sugestões assaz importantes para a concretização desta monografia, e a Binho pela concessão de alguns materiais.

Aos Mestres e Doutores da UFES pela paciência e boa vontade em ensinar, em especial à Eliete, mais que uma professora, uma mãe; a Raquel, um dom para ensinar sem igual; a Klaus, pela amizade e experiência transmitida; ao Coordenador de curso Alessandro, pelo empenho em tentar propiciar-nos o melhor da estrutura e ensino. Enfim, ao meu orientador Frasson, pela companhia nessa empreitada. Meu muito obrigado.

RESUMO

O conceito de internet das coisas vem sendo cada vez mais utilizado nos dias atuais, em que dispositivos são conectados e disponíveis para acesso via internet. Este trabalho propõem um sistema de gerenciamento dos processos de uma residência, tais quais iluminação, segurança, aparelhos eletrônicos. Para tanto, faz-se o uso do microcontrolador ESP8266 para comandar e monitorar diversas tarefas em uma casa. O controle é feito através de um servidor na nuvem ou servidor local, este último instalado no próprio PC, e um aplicativo *android* faz a interface para com o usuário. A espinha dorsal do trabalho é o MQTT (*Message Queuing Telemetry Transport*), um protocolo leve e simples que se adequa perfeitamente aos dispositivos embarcados. Esse protocolo possibilita a utilização de hardware que oferecem pouca capacidade de processamento, memória e limitação na largura de banda. Um protótipo com alguns periféricos foi construído para validar o sistema de automação e monitoramento idealizado, obtendo resultados promissores no que tange o desempenho tanto para acesso local quanto para acesso remoto via internet.

ABSTRACT

The concept of internet of things has been increasingly used in the present day, in which devices are connected and available for access via the internet. This work proposes a system for managing the processes of a residence, such as lighting, security, electronic devices. To do so, the ESP8266 microcontroller is used to control and monitor various tasks in a house. The control is done through a server in the cloud or local server, the latter installed on the PC itself, and an android application interfaces to the user. The backbone of the work is Message Queuing Telemetry Transport (MQTT), a lightweight and simple protocol that fits seamlessly with embedded devices. This protocol enables the use of hardware that offers little processing capacity, memory, and bandwidth limitation. A prototype with some peripherals was built to validate the idealized automation and monitoring system, obtaining promising results in terms of performance for both local access and remote access via internet.

LISTA DE FIGURAS

Figura 1 - Classificação de processadores interconectados por escala. 16
Figura 2 - Modelos de referência OSI e TCP/IP e a correspondência entre suas camadas .. 17
Figura 3 - Características de diferentes padrões de enlaces sem fio..... 18
Figura 4 - Estrutura de um pacote de controle MQTT 20
Figura 5 - Níveis de qualidade oferecidos pelo MQTT 22
Figura 6 – Placa ESP8266-DevKitC..... 26
Figura 7 – Configuração dos pinos de saída da ESP12 28
Figura 8 – Consumo de energia por modos de operação..... 29
Figura 9 – Diagrama do dispositivo reed switch utilizado para acionar o alarme..... 30
Figura 10 - Dispositivo sensor DS18B20 da Maxim..... 31
Figura 11 - Diagrama de blocos do sensor DS18B20..... 33
Figura 12 – Registrador de configuração, que determina a resolução do ADC 33
Figura 13 - Formato do Registrador de temperatura 35
Figura 14 - Modo alimentação parasita durante conversão AD (esquerda) e sensor sendo alimentado por fonte externa (direita). 36
Figura 15 - ROM 64-bit codificada a laser..... 37
Figura 16 - Memória mapeada do DS18B20..... 38
Figura 17 – Circuito digital do cálculo de CRC de 8 bits..... 39
Figura 18 - Topologia da rede para acesso remoto..... 41
Figura 19 – Topologia da rede para acesso local..... 42
Figura 20 – Pacotes mais simples oferecido pela PaaS CloudMQTT 43
Figura 21 – Detalhes da conta no CloudMQTT 44
Figura 22 – Monitoramento das mensagens trafegadas na rede 45
Figura 23 – Instalação do broker Mosquitto no servidor local 46
Figura 24 – Inicialização do serviço do Broker Mosquitto 47
Figura 25 - Diagrama de fluxo do código do ESP8266. 49
Figura 26 – *Servers* local e remoto configurados para operar o sistema 51
Figura 27 – Configuração do servidor no *app MQTT Dash* 52
Figura 28 – Configuração da *metric* para comando da iluminação. 53
Figura 29 – Configuração do *payload* e QoS para a *metric* Iluminação 54
Figura 30 – Circuito elaborado para realização dos testes. 55
Figura 31 – Protótipo de testes com as cargas de iluminação e motor ligadas..... 56

Figura 32 – Diagrama elétrico do transistor para acoplar circuito de potência 57
Figura 33 – Tela do app MQTT Dash de controle dos periféricos 58
Figura 34 – Escolha do servidor com rede local ou via Internet 59
Figura 35 – Cargas acionadas e em funcionamento 60
Figura 36 – Instalação do pacote do ESP8266 na IDE Arduino..... 65
Figura 37 – Instalação da placa ESP8266..... 66
Figura 38 – Seleção da placa para programação 67

LISTA DE QUADROS

Quadro 1 - Tipos de pacote de controle 21
Quadro 2 - Diagrama de fluxo do protocolo com QoS zero 23
Quadro 3 - Exemplo de diagrama de fluxo do protocolo com QoS 1 23
Quadro 4 - Diagrama de fluxo protocolar com QoS dois 24

LISTA DE ABREVIATURAS E SIGLAS

AP	*Access Point*
CI	Circuito Integrado
IEEE	*Institute of Electrical and Electronic Engineers*
IDE	*Integrated Development Environment*
IoT	*Internet of Things*
IP	*Internet Protocol*
ISO	*International Standards Organization*
ITU	*International Telecommunication Union*
LAN	*Local Area Network*
MCU	*Microcontroller Unit*
MQTT	*Message Queuing Telemetry Transport*
OSI	*Open Systems Interconnection*
PC	*Personal Computer*
QoS	*Quality of Service*
SDK	*Software Development Kit*
SSL	*Secure Socket Layer*
TCP	*Transport Control Protocol*
TI	*Texas Instrument*
TLS	*Transport Layer Security*
TRM	*Technical Reference Manual*
Wi-Fi	*Wireless Fidelity*
UFES	Universidade Federal do Espírito Santo
URL	*Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 Motivação 14
1.2 Objetivo 14
2 EMBASAMENTO TEÓRICO 15
2.1 Noções de Redes de Computadores 15
2.1.1 Redes Locais 15
2.1.2 Os modelos OSI e TCP/IP 16
2.1.3 O padrão 802.11 17
2.2 Protocolo MQTT 19
2.2.1 Formato do Pacote de Controle do MQTT 20
2.2.2 Qualidade de Serviço (QoS) 22
2.2.3 Nome e filtro de tópicos e símbolos 25
3 DESCRIÇÃO DO HARDWARE 26
3.1 ESP8266 26
3.2 Sensor magnético 30
3.3 Sensor de temperatura 31
3.3.1 Visão geral 33
3.3.2 Operação – Medição de temperatura 35
3.3.3 Alimentando o DS18B20 36
3.3.4 Memória 37
3.3.5 Geração código CRC 38
4 DESENVOLVIMENTO 40
4.1 Arquitetura da rede 41
4.2 Algoritmo embarcado 48
4.3 Automação IoT 51
5 RESULTADOS E DISCUSSÕES 58
6 CONCLUSÕES 61
REFERÊNCIAS BIBLIOGRÁFICAS 62
APENDICE A – INSTALAÇÃO DO ESP8266 NA IDE ARDUINO 65
APÊNDICE B – CÓDIGO IOT 68

1 INTRODUÇÃO

A conexão de sensores e dispositivos à Internet cresce de maneira vertiginosa, de tal sorte a ser cunhado o termo Internet das Coisas, a partir do momento, estimado entre 2008 e 2009, em que aqueles se tornaram em maior número que o de pessoas conectadas à rede mundial de computadores. Segundo Cisco IBSG (2011, p.4), até 2020 serão 50 bilhões de dispositivos conectados, uma média de 6,8 para cada habitante do planeta.

Nesse ínterim, as milhares de redes e sistemas de controle que fazem parte do nosso dia-dia, até então isoladas, caminham em direção a serem interligadas, gerando assim sistemas cada vez mais inteligentes para analisar informações e agregar conhecimento para aumento de eficiência e produtividade. Outra aplicação de destaque é a segurança patrimonial e a praticidade para uma época em que o tempo é um recurso não muito disponível, e a violência é preocupação constante para muitos cidadãos.

Não obstante ainda haja alguma resistência por parte das pessoas, a automação residencial já é uma realidade tangível para muitos, devido ao reduzido custo que ela vem demonstrando e facilidade de manuseio oferecida aos leigos usuários.

Este trabalho está dividido em 6 capítulos, perpassando a teoria e os materiais e métodos envolvidos em sua concepção, apresentando ao final o produto obtido.

No capítulo 2, abrange-se o conhecimento teórico sobre redes de computadores e um debruçar apropriado do protocolo usado. No capítulo 3, os elementos físicos são comentados e suas especificações são fornecidas. Em sequência, o capítulo 4 contém o desenvolvimento propriamente dito, abordando a arquitetura adotada, o algoritmo de programação e a interface com o usuário final aplicada a uma plataforma de testes. O capítulo 5, por sua vez, apresenta o resultado final do trabalho e sugestões e melhorias para projetos futuros. Por fim, o capítulo 6 traz as conclusões pertinentes.

1.1 Motivação

Do interesse do autor sobre as áreas de Eletrônica e Telecomunicações, e tomando por base a conjetura da modernidade em que a tecnologia sobressai em diversas atividades do cotidiano, surgiu a proposta de conceber um sistema de automação residencial via Internet ou rede local, utilizando-se do conhecimento de Redes de Computadores e Sistemas Embarcados, adquiridos ao longo da formação acadêmica em Engenharia Elétrica, concebendo assim um produto que poderia agregar algum valor mercadológico e expandir os horizontes do autor acerca da temática escolhida.

1.2 Objetivo

Este trabalho propõe a implantação de uma solução para automação residencial completa, abarcando as camadas de enlace, rede e aplicação. Diante disso, utilizou-se o microcontrolador ESP8266, e do programa *Mosquitto*, que implementa o protocolo MQTT, na nuvem e localmente. O trabalho pode ser dividido em três fases, a saber, definição da arquitetura da rede e protocolo utilizado, implementação do algoritmo contemplando algumas funcionalidades requeridas por um sistema de automação e monitoramento e, por fim, a integração destes com o servidor através de uma interface gráfica para controle do usuário, feita por um aplicativo *android*, disponibilizado gratuitamente na Internet.

2 EMBASAMENTO TEÓRICO

Este capítulo destina-se a fornecer o ferramental teórico necessário, para se compreender o escopo no qual se encontra aplicado o corrente projeto de graduação.

2.1 Noções de Redes de Computadores

De acordo com TANENBAUM (2011, p.1), a expressão ‘rede de computadores’ indica:

> [...] um conjunto de computadores autônomos interconectados por uma única tecnologia. Dois computadores estão interconectados quando podem trocar informações [...]. Existem redes de muitos tamanhos, modelos e formas, como veremos mais adiante. Elas normalmente estão conectadas para criar redes maiores, com a Internet sendo o exemplo mais conhecido de uma rede de redes.

No entanto, cada vez mais todo tipo de equipamento ou aparelho, como eletrodomésticos e dispositivos de sensoriamento remoto estão sendo conectados à Internet, a chamada IoT, tema explorado nesta monografia, e, como constata KUROSE (2010, p.2), “o termo rede de computadores está começando a soar um tanto desatualizado, dados os muitos equipamentos não tradicionais que estão sendo ligados à internet”.

2.1.1 Redes Locais

Rede Local, ou Local Area Network (LAN), é uma rede que pode compreender uma distância de aproximadamente 10 m - como um cômodo, a 1 quilômetro, como um campus de uma universidade, e podem utilizar uma série de tecnologias de transmissão diferentes, podendo ser com fio ou sem fio, como ondas de rádio, fio de cobre ou fibra óptica.

A Figura 1 mostra vários sistemas de processadores por seu tamanho físico, um dos dois critérios pelos quais as redes são classificadas, a escalabilidade. O outro é a tecnologia de transmissão, isto é, enlaces ponto a ponto, o qual conecta pares de máquinas individuais, ou broadcast, onde apenas um canal de comunicação é compartilhado por todos os sistemas finais da rede.

Figura 1 - Classificação de processadores interconectados por escala.

Distância do interprocessador	Processadores localizados no mesmo	Exemplo
1 m	Metro quadrado	Área pessoal
10 m	Cômodo	Rede local
100 m	Prédio	
1 km	Campus	
10 km	Cidade	Rede metropolitana
100 km	País	Rede a longas distâncias
1.000 km	Continente	
10.000 km	Planeta	A Internet

Fonte: TANEMBAUM, (2011).

2.1.2 Os modelos OSI e TCP/IP

O modelo de referência OSI (Open Systems Interconnection) foi desenvolvido na década de 80 e revisado em 1995, como um primeiro passo em direção à padronização internacional dos protocolos usados nas várias camadas de uma arquitetura de rede.

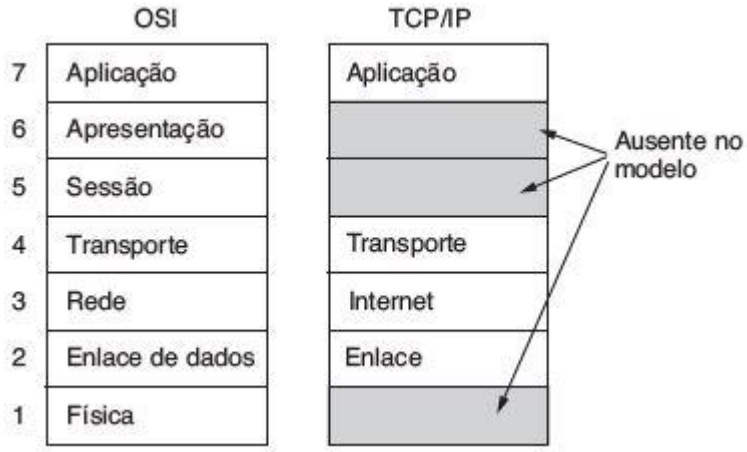
Como asserta Tanenbaum (2011, p.25 e 26), embora não seja propriamente uma arquitetura de rede, pois não especifica os serviços e protocolos exatos que devem ser usados em cada camada, o modelo em si é bastante geral e ainda válido, e as características descritas em cada camada ainda são muito importantes.

O modelo de referência TCP/IP, nome dado devido a seus dois principais protocolos, surgiu ainda na década de 70 nos Estados Unidos, e teve como um dos principais objetivos de projeto a capacidade para conectar diferentes redes de maneira uniforme, sendo assim, depois de melhorado, definido como um padrão na comunidade da Internet (Braden, 1989). “Além disso, como eram visadas aplicações com requisitos divergentes, desde a transferência de arquivos e

a transmissão de dados de voz em tempo real, era necessária uma arquitetura flexível" (TANEMBAUM, 2011, p.28).

A próxima figura mostra as diferentes camadas que compõe ambos os modelos, e as respectivas correspondências entre essas camadas.

Figura 2 - Modelos de referência OSI e TCP/IP e a correspondência entre suas camadas



Fonte: TANENBAUM, (2011).

2.1.3 O padrão 802.11

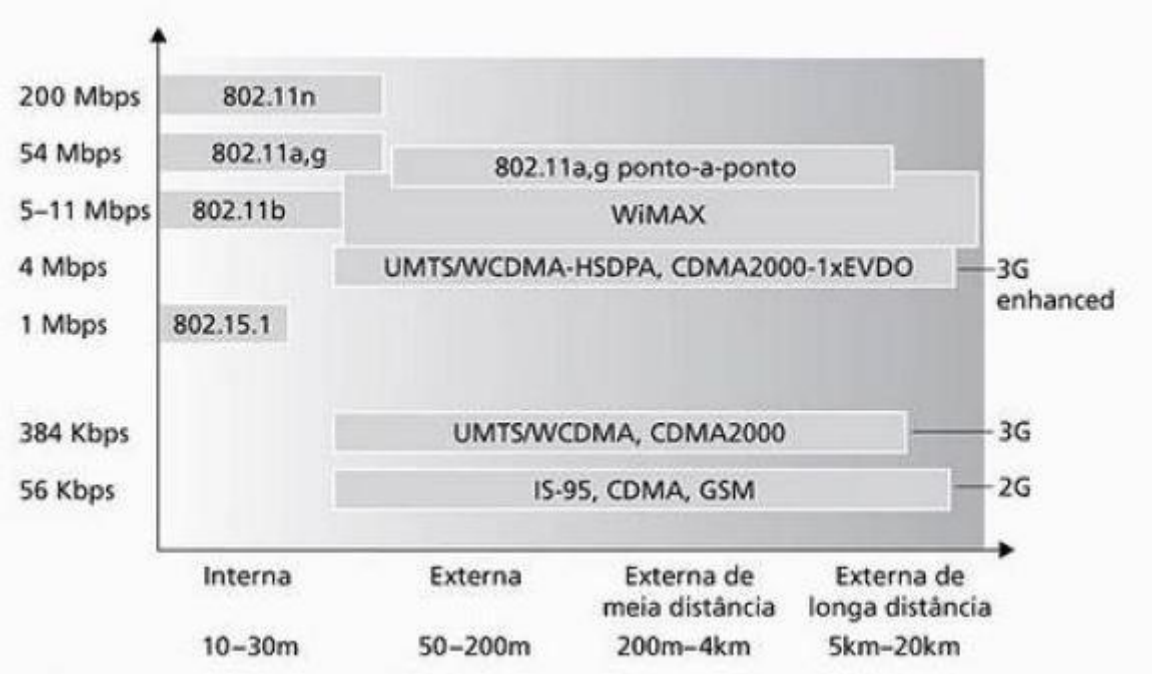
O protocolo IEEE 802.11, conhecido como Wi-Fi, é o principal padrão de LAN sem fio, onde clientes, como laptops e smartphones, são conectados a outra rede, como uma intranet da empresa ou à Internet, através de estações-base, como AP's (Access Points). "Os pontos de acesso se conectam à rede com fios, e toda comunicação entre os clientes passa por um ponto de acesso [AP]." (TANEMBAUM, 2011, p. 43)

Os sistemas 802.11 operam nas bandas não licenciadas, tal qual a ISM (*Industrial, Scientific, and Medicals*), definidas pela *ITU Radiocommunication Sector*, como por exemplo, 902-928 MHz, 2,4-2,5 GHz, 5,725-5,825 GHz).

Transmissores e receptores de rádio perfazem o trabalho de comunicação entre duas estações, chamados de enlaces sem fio, e "tecnologias de enlace sem fio diferentes têm taxas de transmissão diferentes e podem transmitir a distâncias diferentes". (KUROSE, 2010, p. 379)

A Figura 3 mostra duas características fundamentais (área de cobertura e taxa de enlace) dos padrões de enlace sem fio mais populares, sendo os valores aproximações razoáveis, dado que essas medidas podem variar dependendo da distância, condições do canal e do número de usuários na rede sem fio. (KUROSE, 2010).

Figura 3 - Características de diferentes padrões de enlaces sem fio



Fonte: KUROSE, (2010).

2.2 Protocolo MQTT

Criado pela IBM no final da década de noventa, sua motivação original era vincular sensores em *pipelines* de petróleo a satélites. Em 2014, ele foi ligeiramente modificado e mais documentado, se tornando oficialmente um padrão aberto OASIS, com suporte nas linguagens de programação populares, usando diversas implementações de *software* livre. (YUAN, 2017)

O protocolo MQTT (*Message Queuing Telemetry Transport*) é baseado no conceito Cliente/Servidor e segue o paradigma Publicar/Inscrever-se para troca de informações. É um protocolo aberto e simples, e projetado para ser de fácil implementação, características que o fazem ideal para aplicações em que há limitação de espaço e de processamento.

O MQTT está na camada de aplicação e é construído sobre o protocolo de rede TCP/IP, este que é utilizado pela rede das redes, a Internet, ou sobre outros protocolos que agregam conexões bidirecionais, ordenadas e sem perda, podendo também ser os protocolos TLS e *WebSocket*, não abordados neste texto. (OASIS STANDARD, 2014, p.52)

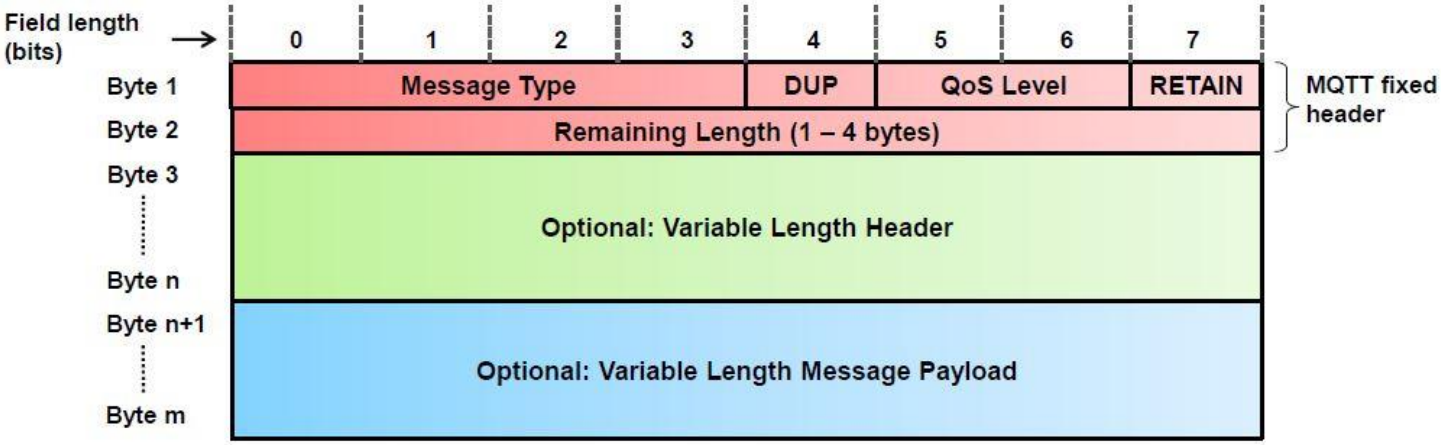
Seus atributos principais são o uso do padrão *Publish/Subscribe*, que efetua distribuição *one-to-many* (um para muitos) e, de acordo com YUAN (2017, parágrafo 2), proporciona desacoplamento entre "o emissor e o receptor da mensagem tanto no tempo como no espaço e, portanto, é escalável em ambientes de rede que não são confiáveis". Este desacoplamento é conseguido devido ao servidor, designado Broker, fazer o gerenciamento das mensagens e interface entre os clientes que o acessam.

Deste modo, os clientes podem publicar informações, de telemetria ou comando por exemplo, ou se subscreverem para receberem os dados que lhe interessam, utilizando um tablet ou celular android. Isto é feito através da criação de categorias específicas, chamadas de tópicos, que são acessados pelos clientes a fim de publicar ou colher informações. Assim, um cliente Subscriber (assinante) se inscreve em um tópico para receber as informações do cliente Publisher que foram enviadas para aquele tópico.

2.2.1 Formato do Pacote de Controle do MQTT

O protocolo MQTT trabalha intercambiando uma série de pacotes de controle definidos. Cada pacote de controle é constituído de três partes, e são enviados sempre na ordem como ilustrado na Figura 4.

Figura 4 - Estrutura de um pacote de controle MQTT



Fonte: EGLI, (2016).

A primeira parte é obrigatória, o cabeçalho fixo, que informa o tipo de pacote de controle que está sendo trocado na rede, e também o número de bytes restantes que contém o presente pacote de controle, conforme ainda a figura precedente.

Os tipos de pacote de controle informam qual a finalidade do pacote de controle que está sendo trocado entre cliente e servidor, como um pacote CONNECT, o qual o cliente solicita uma conexão ao servidor, ou um PUBLISH, este que por sua vez realiza uma publicação. O Quadro 1 designa os tipos de pacote de controle e o que desempenham.

Quadro 1 - Tipos de pacote de controle

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment
PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

Fonte: OASIS STANDARD, (2014).

A segunda parte, o cabeçalho variável, está presente na maioria dos pacotes, e varia conforme o tipo de pacote. Esta contém o campo com o número de identificação do pacote, o nome do protocolo e seu nível de QoS e uma indicação se a mensagem publicada deverá ser retida no servidor ou não, dentre outros, e se destina a organizar o fluxo de informação.

A terceira parte, o *payload*, é o próprio conteúdo da mensagem que está sendo transmitida, ou a carga útil, como por exemplo, o nome de um tópico a ser subscrevido ou um valor de temperatura a ser publicado.

2.2.2 Qualidade de Serviço (QoS)

O MQTT oferece três níveis de qualidade de serviço, os quais fornecem confiabilidade distintas na entrega dos pacotes trafegados, e pode ser visto na Figura 5.

Figura 5 - Níveis de qualidade oferecidos pelo MQTT

Fonte: EGLI, (2016).

- **QoS 0**: At most once delivery (no máximo uma entrega)

  Nenhuma resposta é mandada pelo receptor, também não é efetuado reenvio por parte do transmissor. É o serviço mais simples de entrega.

Quadro 2 - Diagrama de fluxo do protocolo com QoS zero

Sender Action	Control Packet	Receiver Action
PUBLISH QoS 0, DUP=0		
	---------->	
		Deliver Application Message to appropriate onward recipient(s)

Fonte: OASIS STANDARD, (2014).

- **QoS 1**: At least once delivery (pelo menos uma entrega)

  Este nível de qualidade garante que a mensagem chegue ao receber pelo menos uma vez. Um pacote publicado contém o número de identificação, e o receptor retorna com confirmação de que o pacote foi entregue.

Quadro 3 - Exemplo de diagrama de fluxo do protocolo com QoS 1

Sender Action	Control Packet	Receiver action
Store message		
Send PUBLISH QoS 1, DUP 0, <Packet Identifier>	---------->	
		Initiate onward delivery of the Application Message[1]
	<----------	Send PUBACK <Packet Identifier>
Discard message		

Fonte: OASIS STANDARD, (2014).

- **QoS 2**: Exactly once delivery (precisamente uma entrega)

É a qualidade máxima para transmissão confiável, na qual tanto perda quanto repetição de mensagens não são admitidas, às expensas de maior tráfego de pacotes no canal. A confirmação de entrega, então, é mais elaborada, sendo realizada em dois passos.

Quadro 4 - Diagrama de fluxo protocolar com QoS dois

Sender Action	Control Packet	Receiver Action
Store message		
PUBLISH QoS 2, DUP 0 <Packet Identifier>		
	---------->	
		Method A, Store message or Method B, Store <Packet Identifier> then Initiate onward delivery of the Application Message[1]
		PUBREC <Packet Identifier>
	<----------	
Discard message, Store PUBREC received <Packet Identifier>		
PUBREL <Packet Identifier>		
	---------->	
		Method A, Initiate onward delivery of the Application Message[1] then discard message or Method B, Discard <Packet Identifier>
		Send PUBCOMP <Packet Identifier>
	<----------	
Discard stored state		

Fonte: OASIS STANDARD, (2014).

2.2.3 Nome e filtro de tópicos e símbolos

Nome de tópico se refere ao rótulo ao qual a mensagem será publicada, enquanto filtro de tópico é a expressão contida na subscrição que indica os tópicos de interesse do cliente *subscriber*. O broker (o servidor), manda uma cópia das mensagens que contém filtros de interesse iguais aos nomes de tópico.

Uma barra normal ('/' U+002F) é usada para separar os tópicos em níveis, provendo uma estrutura hierárquica para estes. O separador de níveis é interessante para realizar o uso dos caracteres coringas, que incluem múltiplos tópicos a serem subscritos ao mesmo tempo, e são explicados a seguir.

O símbolo de número ('#' U+0023), coringa de múltiplos níveis, corresponde a se inscrever a todos os tópicos seguintes ao último nível, este último incluso. Ele necessariamente deve ser o último caractere de um filtro de tópico, e pode ser escrito após a barra ou após o nome do último nível, sem a barra.

O símbolo mais ('+' U+002B) é para quando se deseja obter os vários rótulos contidos num mesmo nível, sendo ele então coringa de nível único.

Segue, então, uma lista sumarizada dos três principais símbolos utilizados pelos clientes:

- '/', separa um tópico em vários níveis hierárquicos
- '#', abrange todos os níveis inferiores
- '+', abrange todos os rótulos do nível atual

Por exemplo, se um cliente subscreve em 'esporte/tênis/jogador1/#', ele receberia as mensagens publicadas nos seguintes tópicos: 'esporte/tênis/jogador1', 'esporte/tênis/jogador1/ranking' e 'esporte/tênis/jogador1/pontos/Wimbledon'.

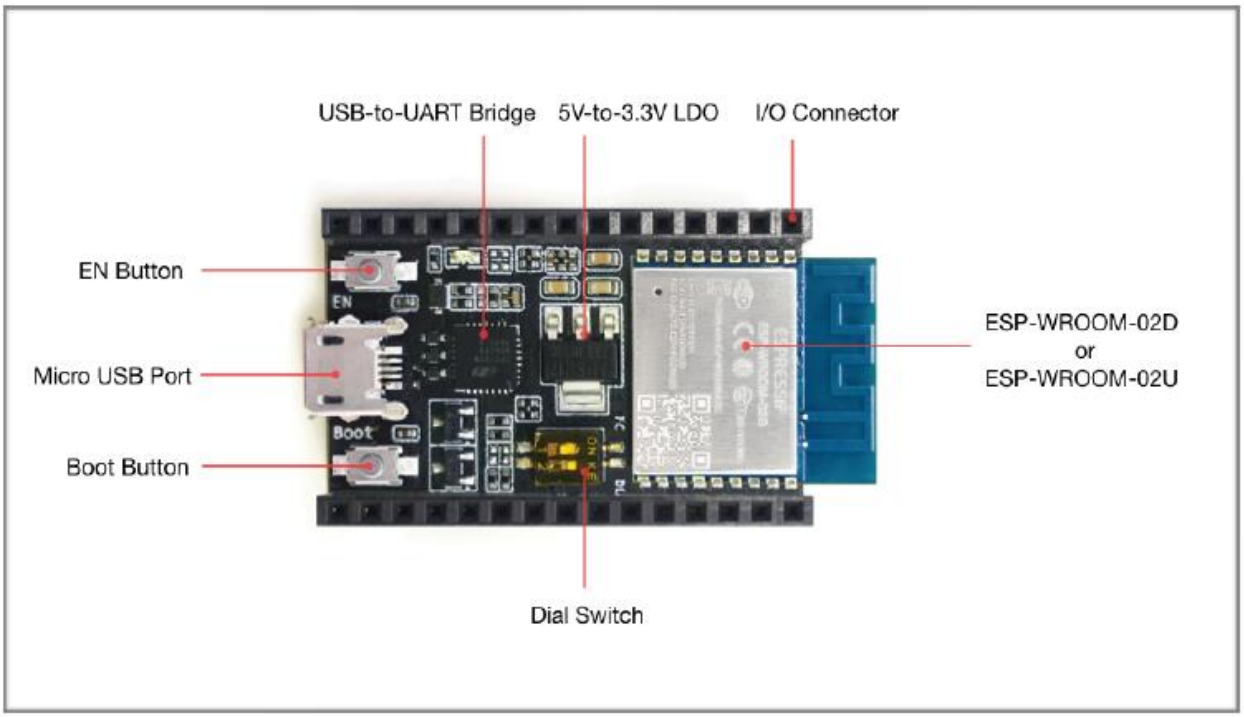
Se este mesmo cliente subscrevesse no tópico 'esporte/tênis/+', ele receberia mensagens publicadas apenas em 'esporte/tênis/jogador1' e 'esporte/tênis/jogador2', e não mais dos níveis sucessivos.

3 DESCRIÇÃO DO HARDWARE

3.1 ESP8266

Foi utilizado no presente projeto como núcleo de processamento a plataforma de desenvolvimento ESP8266-DevKitC, ou ESP12, produzida especificamente para aplicações IoT pela Espressif Systems, que contém o ESP8266, um microcontrolador SoC (*System-on-Chip*) com elevada integração que oferece uma solução completa para conexões de rede WiFi, podendo ser usado tanto como dispositivo mestre para uma aplicação ou escravo como *bridge* WiFi para outro microcontrolador, i.e., como um *shield*.

Figura 6 – Placa ESP8266-DevKitC



Fonte: ESPRESSIF, (2018).

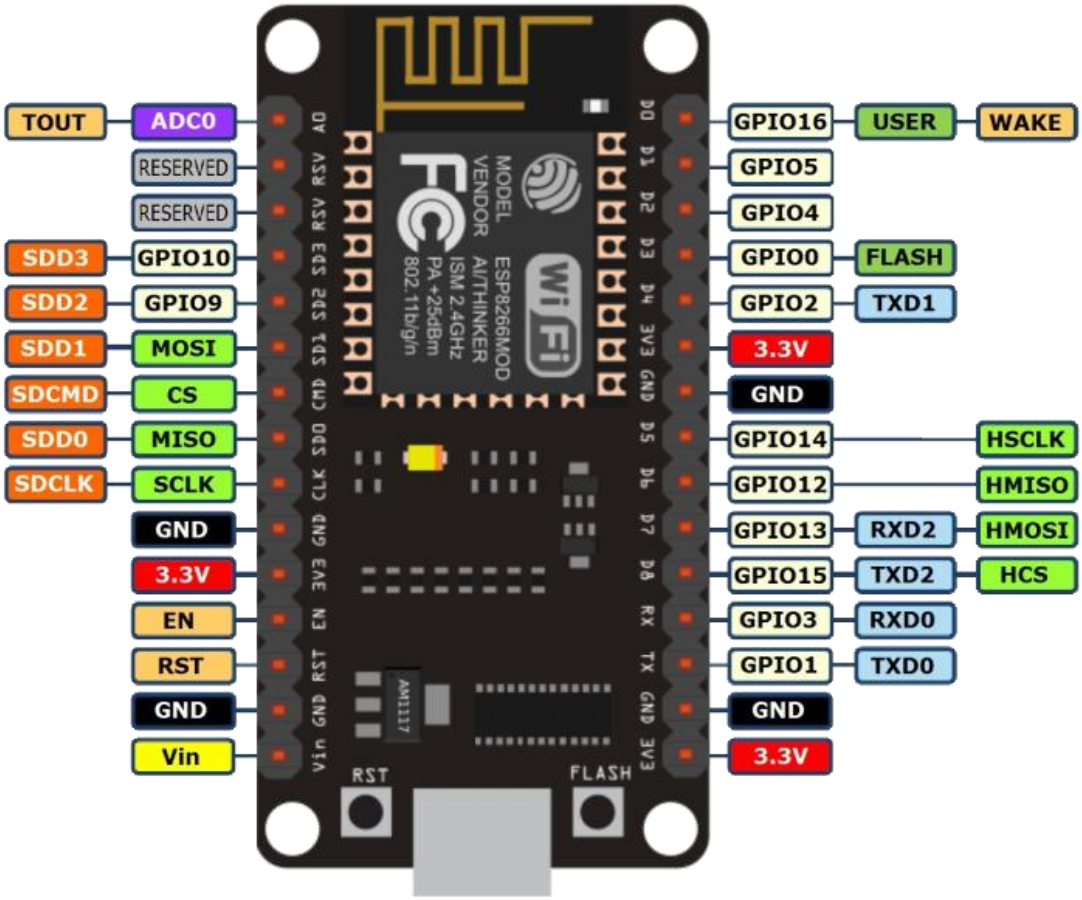
Como descrito a seguir, ele detém um elevado poder de processamento com seu Tensilica de 32 bits e 80MHz, capacidade de memória satisfatória, uma quantidade de GPIO's expressiva, e, o grande diferencial para seu alto benefício-custo, integra um chip Wifi, dispensando o uso de *shields* (componentes extras com determinada função acoplados ao microcontrolador).

O ESP8266 possui as seguintes especificações:

- ✓ Tensão de operação: 3,0~3,6 V (3,3V típico)
- ✓ Corrente de operação: 80mA em média
- ✓ Máxima corrente GPIO's: 12 mA
- ✓ Processador Tensilica L106 32 bit RISC
- ✓ Velocidade do processador: 80~160MHz
- ✓ 17 GPIO's (multiplexado com outras funções)
- ✓ Memória RAM de 50 KB exclusiva para aplicações
- ✓ Memória externa SPI Flash 16MB max (512KB standard)
- ✓ 1 ADC com 1024 níveis de resolução (10 bits)
- ✓ WiFi 802.11 b/g/n 2.4 GHz com suporte a WPA/WPA2
- ✓ Modos de operação transceptor: STA/AP/AP+STA
- ✓ Pilha do protocolo TCP/IP
- ✓ Protocolos de rede: IPv4, TCP/UDP/HTTP/FTP
- ✓ Barramento periféricos: UART/SDIO/SPI/I2C/I2S/IR Remote Control/GPIO/PWM

A disposição dos pinos e as funções dos mesmos podem ser conferidas na figura que se segue.

Figura 7 – Configuração dos pinos de saída do ESP12



Fonte: MURTA, (2018).

O ESP8266 é essencialmente um WiFi/Serial *Transceiver*, e foi projetado com um avançado gerenciamento de energia específico para uso em dispositivos móveis e aplicações de IoT, e sua arquitetura *low-power* opera nos seguintes modos conforme a Figura 8, e posterior explanação dos mesmos.

Figura 8 – Consumo de energia por modos de operação

Power Mode	Description	Power Consumption
Active (RF working)	TX 802.11b, CCK 11Mbps, P_{OUT}=+17 dBm	170 mA
	Rx 802.11b, 1024 bytes packet length , –80 dBm	50 mA
Moderm-sleep①	CPU is working	15 mA
Light-sleep②	-	0.9 mA
Deep-sleep③	Only RTC is working	20 uA
Shut down	-	0.5 uA

Fonte: ESP8266EX Datasheet, (2018). Adaptada.

No modo ativo, o chip Wi-Fi está operante, podendo receber, transmitir, ou ouvir dados. O CPU e outros periféricos são pausados no modo *Light-sleep*, e a qualquer evento (MAC, *host*, temporizador RTC, ou interrupções externas) irá ativar o processador. Já no modo *Deep-sleep*, todas as partes do MCU estão desligadas, exceto o Real Time Clock (RTC), ideal para aplicações com intervalos acima de 100s de transmissão de dados.

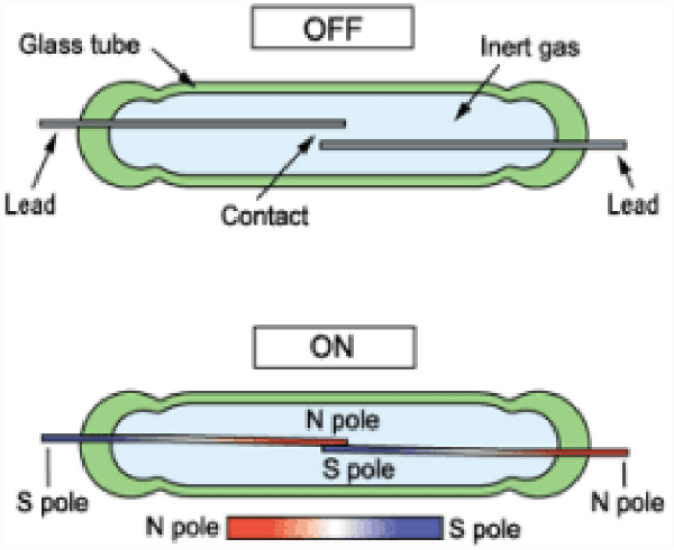
Uma grande praticidade para sua utilização, o ESP8266 pode ser programado pela IDE do Arduino, havendo apenas que adicionar o respectivo módulo para aquele, e consta na seção de apêndices.

3.2 Sensor magnético

A chave magnética, ou *reed switch*, constitui-se em duas lâminas flexíveis feitas de material magnético encapsuladas em um bulbo de vidro preenchido de um gás inerte para evitar a ação oxidativa do ar sobre as lâminas, o que poderia afetar a qualidade dos contatos.

Sua ação consiste na atuação de um campo magnético externo que, ao ser aproximado do bulbo de vidro, atrai as lâminas que, por sua vez, fecham contato e estabelecem a passagem de corrente, que será então detectada num circuito externo para dada finalidade. No presente caso, simulará o acionamento do alarme em uma residência. Na Figura 9 vemos o diagrama de constituição do dispositivo e seu princípio de funcionamento. Importante salientar a que o campo magnético externo precisa ser devidamente orientado para ocorrer a atração das lâminas e o consequente fechamento do contato.

Figura 9 – Diagrama do dispositivo reed switch utilizado para acionar o alarme



Fonte: KUMAR, (201-?)

O dispositivo pode ainda ser utilizado em inúmeras outras aplicações, e não apenas na finalidade de sensor para alarme, como um relé de alta sensibilidade enrolando-se um fio esmaltado em torno do bulbo de vidro de modo a formar uma bobina, ou com dois ou mais reed-switches se pode sensoriar um objeto que gira.

3.3 Sensor de temperatura

Um estudo mais aprofundado será apresentado sobre o sensor de temperatura empregado no trabalho, compreendendo mais detalhadamente o funcionamento destes, no intento de deixar documentado para futuros trabalhos em áreas afins.

O sensor utilizado para captar a temperatura ambiente foi o DS18B20, um sensor que pode ser imerso em líquido e tem amplitude de aferição de -55°C a +125 °C.

O termômetro digital DS18B20 confere resolução ajustável de 9 até 12 bits e possui uma função alarme com limiares de disparo superior e inferior, a qual pode ser programada pelo usuário. Sua comunicação é feita através de um fio apenas (*1-Wire bus*, do inglês barramento de 1 fio), possuindo então apenas uma linha bidirecional de dados e um fio para a referência (GND). Além do que, pode ser alimentado pela própria linha de dados, no chamado modo *parasite-power*, eliminando a necessidade de alimentação externa.

Figura 10 - Dispositivo sensor DS18B20 da Maxim

Fonte: AUTOR.

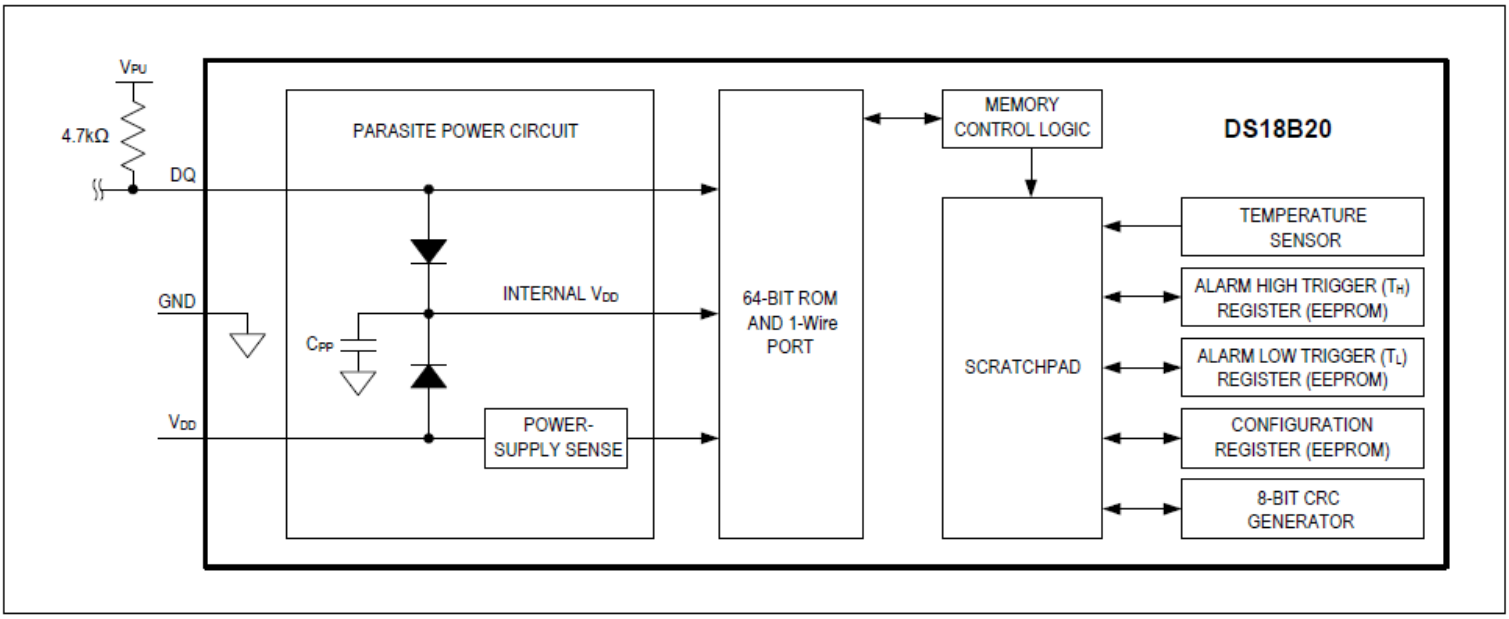
A seguir uma lista contemplando todos os seus benefícios e características:

- Interface *1-Wire* requerendo somente um pino de porta para comunicação
- Reduz o número de componentes com o sensor de temperatura e EEPROM integrados
  - Medição de temperatura de -55°C a +125 °C
  - 0,5°C de precisão entre -10°C a +85°C
  - Resolução programável de 9 a 12 bits
  - Dispensa a necessidade de componentes externos
- Modo Parasite-Power requer apenas 2 pinos para operar (DQ e GND)
- Simplifica a medição de temperatura distribuída (vários sensores) com sua capacidade de *multidrop line*, utilizando apenas uma linha de sinal, pois cada dispositivo tem um código serial gravado em sua ROM de 64-bit.
- Configuração de alarme (não-volátil) definida pelo usuário, com comando de busca para identificar os sensores com temperatura fora dos limites programados.

3.3.1 Visão geral

A Figura 11 mostra o diagrama de blocos constituintes do dispositivo, e a seguir uma descrição geral do dispositivo será exposta ao leitor.

Figura 11 - Diagrama de blocos do sensor DS18B20



Fonte: MAXIM INTEGRATED, (2015).

A ROM de 64-bit guarda o código identificador do dispositivo, juntamente com o byte de CRC que é calculado pelos 56 bits do código da própria ROM, e o byte com o código da família 1-Wire do DS18B20, como pode-se ver na Figura 12.

Figura 12 – Registrador de configuração, que determina a resolução do ADC

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0	R1	R0	1	1	1	1	1

Fonte: MAXIM INTEGRATED, (2015).

A *scratchpad memory* (do inglês, bloco de rascunho) contém o registrador de temperatura que guarda o valor digital de *output* do sensor em 2-bytes (MSB e LSB) tipo sinal estendido e com complemento a dois.

Além que, fornece acesso aos 2 bytes do valor de disparo superior e inferior do alarme do dispositivo, T_H *e* T_L, e ao registrador de configuração de 1-byte, conforme figura 15, o qual serve a fornecer a resolução do conversor AD que digitalizará a informação da temperatura aferida. Estes três últimos bytes citados são não voláteis (EEPROM), portanto reterão o valor quando o sistema desligar.

O sensor usa o protocolo 1-Wire bus, exclusivo da companhia, que realiza a comunicação fazendo uso de um sinal de controle. A linha de controle demanda um resistor *pullup* de baixo valor, já que os dispositivos são interligados ao barramento via porta 3-state ou porta *open-drain,* este último sendo o caso do sensor em estudo.

3.3.2 Operação – Medição de temperatura

Como dito, a resolução para a aferição da temperatura é configurável pelo usuário em 9,10,11 e 12 bits, correspondendo a incrementos de 0.5, 0.25, 0.125 e 0.0625 °C, respectivamente.

O valor da temperatura é, então, armazenada no registrador de temperatura como número binário de 16-bit com sinal estendido e complemento a dois, mostrado na Figura 13. Os bits de sinal (S) indicam se o valor é positivo ou negativo. Se o sensor estiver configurado para resolução de 12-bit, todos os bits do registrador de temperatura irão conter dados válidos; com resolução de 11-bit, o bit 0 ficará inutilizado, e assim por diante.

Figura 13 - Formato do Registrador de temperatura

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
LS BYTE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
MS BYTE	S	S	S	S	S	2^6	2^5	2^4

S = SIGN

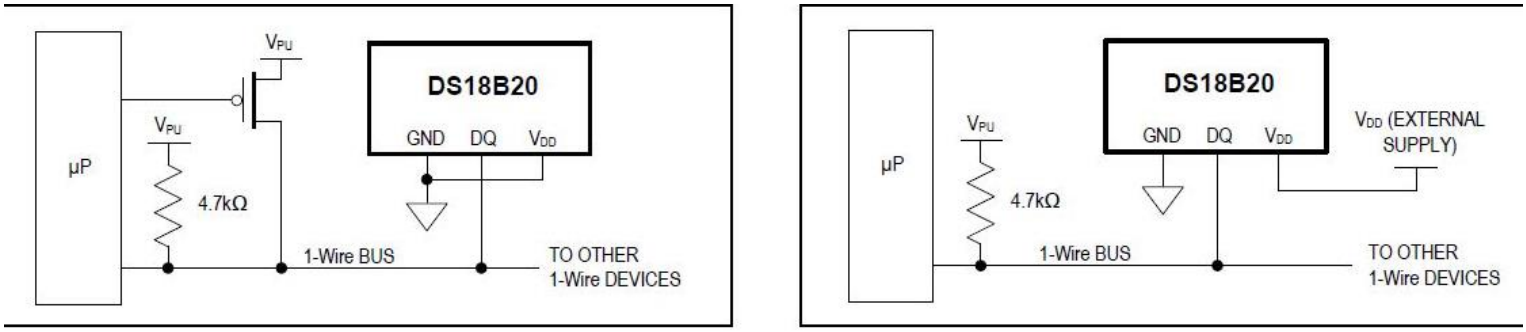
Fonte: MAXIM INTEGRATED, (2015).

3.3.3 Alimentando o DS18B20

O DS18B20 pode ser alimentado por fonte externa pelo pino Vdd, ou pode operar em modo “parasite power”, que permite operar sem alimentação externa local. Este modo é muito útil em aplicações que requerem sensoriamento remoto de temperatura ou são limitadas em termos de espaço.

A Figura 14 mostra o circuito de controle do modo *parasite power*, o qual drena potência do barramento pelo pino DQ quando o barramento está em nível alto, e a corrente drenada é também usada para carregar o capacitor parasite power (Cpp) para prover potência quando o barramento está em baixa. Importante frisar que nesse modo o pino Vdd deve ser conectado à referência (GND).

Figura 14 - Modo alimentação parasita durante conversão AD (esquerda) e sensor sendo alimentado por fonte externa (direita).



Fonte: MAXIM INTEGRATED, (2015).

Para a maioria das aplicações, o barramento 1-wire e Cpp podem fornecer corrente suficiente desde que as especificações de tempo e demanda de ddp sejam satisfeitas. No entanto, quando o sensor está realizando conversão de temperatura ou copiando dados da memória *scratchpad* para a EEPROM, a corrente exigida pode chegar a 1,5 mA. Essa corrente pode causar inaceitável queda de tensão no resistor pullup e é maior que a corrente suportada pelo capacitor.

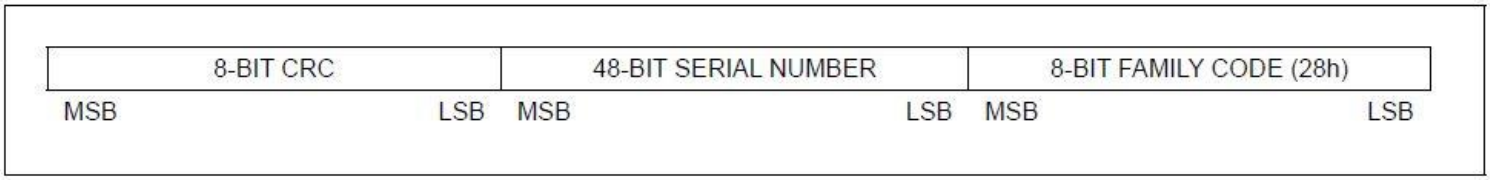
Assim, para assegurar que o sensor pode suprir a corrente necessária, providencia-se uma ligação direta à alimentação positiva sempre que as duas situações críticas supracitadas tomam lugar. Para tanto, faz-se uso de um MOSFET para conectar o barramento diretamente ao polo positivo da alimentação do microcontrolador, e nenhuma outra atividade pode ocorrer no barramento enquanto isto, que deve permanecer em nível alto durante a conversão ou transferência dos dados.

3.3.4 Memória

A memória do DS18B20 consiste em uma ROM, uma SRAM scratchpad e uma EEPROM para armazenar os bytes superior e inferior (T_H e T_L) do valor de disparo da função alarme e para guardar o registrador de configuração. Caso o alarme não esteja sendo usado, os registradores T_H e T_L podem servir de memória de propósito-geral.

A ROM de 64-bit guarda o código identificador do dispositivo, juntamente com o byte de CRC que é calculado pelos 56 bits do código da própria ROM, e o byte com o código da família 1-Wire do DS18B20, como pode-se ver na Figura 15.

Figura 15 - ROM 64-bit codificada a laser



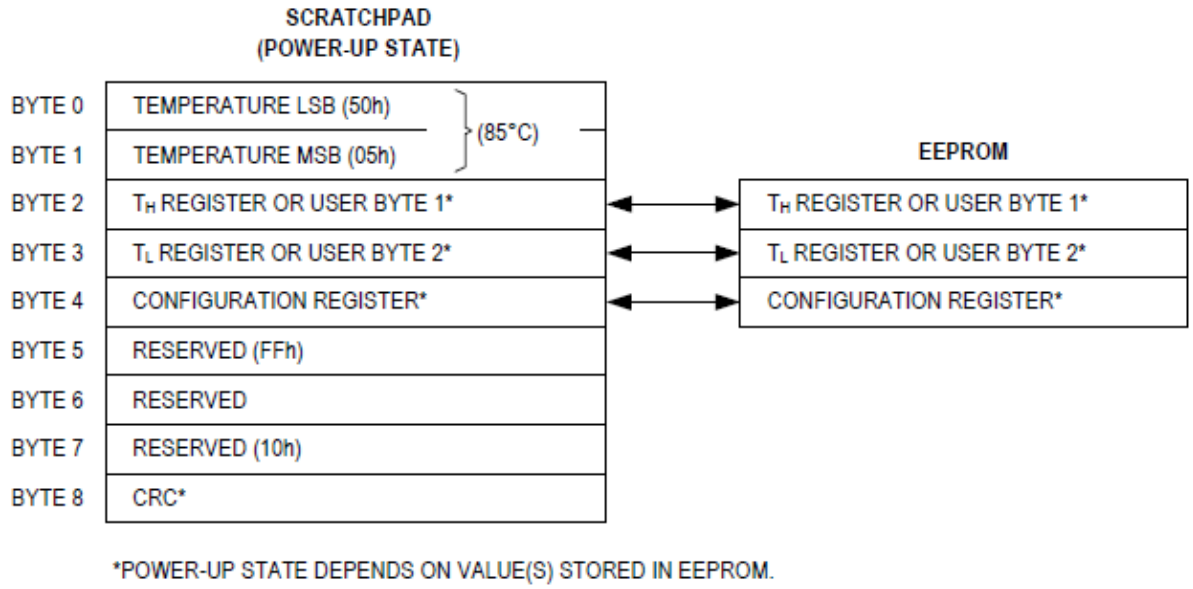
Fonte: MAXIM INTEGRATED, (2015).

Byte 0 e byte 1 da memória scratchpad contém o LSB e MSB do registrador de temperatura, respectivamente. Estes são de apenas leitura (read-only). Bytes 2 e 3 provê acesso aos registradores T_H e T_L. Byte 4 contém os dados do registrador de configuração. Por fim, os bytes 5, 6 e 7 são reservados para uso interno do dispositivo e não podem ser sobrescrevidos.

O byte 8 é também apenas leitura e mantém o código CRC dos bytes 0 a 7 desta memória. O método de geração do CRC será descrito no próximo item.

Dados são escritos nesta memória usando o comando Write Scratchpad (4Eh); e devem ser transmitidos iniciando do bit menos significativo do byte 2. Quando da leitura da memória scratchpad, dados são transferido via barramento 1-Wire começando do bit menos significativo do byte 0. Para transferir T_H e T_L e a configuração, da memória scratchpad para a EEPROM, o mestre (o esp-32) deve efetuar o comando Copy Scratchpad (48h).

Figura 16 - Memória mapeada do DS18B20



Fonte: MAXIM INTEGRATED, (2015).

3.3.5 Geração código CRC

Dados seriais podem ser averiguados de erros de várias maneiras. Uma das mais simples é acrescentar um nono bit ao byte transmitido, denominado bit de paridade, de forma a deixar os 9 bits sempre com número ímpar de bits 1, por exemplo, ao byte 11010001, com quatro bits ‘1’, seria adicionado um 9º bit com valor ‘1’, e assim o conjunto de 9 bits ficaria com um número ímpar de bits ‘1’; caso o byte já contivesse um número ímpar de bits ‘1’, o bit de controle acrescido seria o ‘0’. O lado receptor da transmissão, por sua vez, confere se a sequência de 9 bits contém um número ímpar de bits ‘1’, e se for confirmado, os dados são validados. Esse modo, embora útil, no entanto, apenas detecta um número ímpar de erros, pois se, por exemplo, dois bits ‘0’ forem transmitidos como ‘1’, a validação dos dados ainda assim

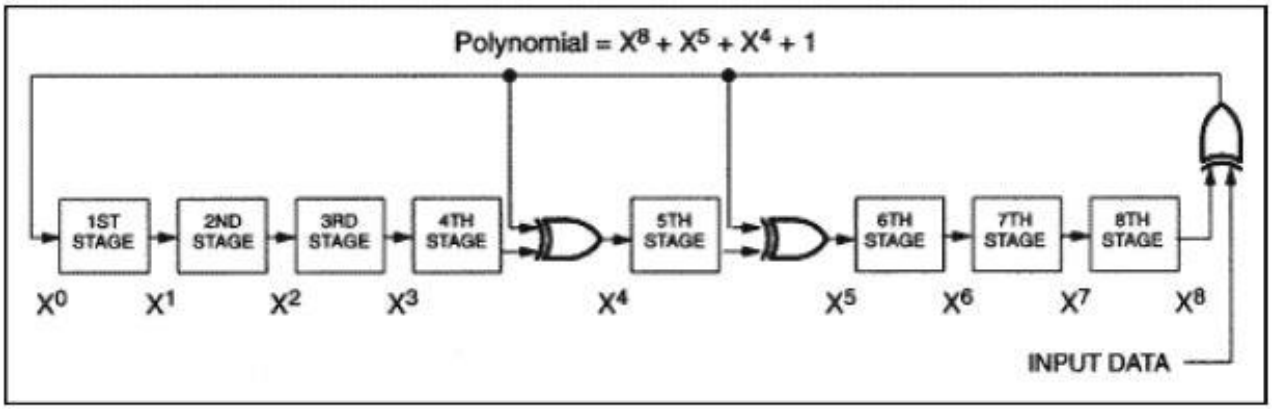
será confirmada, mesmo havendo tido a corrupção dos dois bits ‘0’, pois continuará tendo número ímpar de 1’s. Esse tipo de esquema é chamado paridade ímpar.

O acrônimo CRC significa Cyclic Redundancy Check, ou Verificação de Redundância Cíclica, e é o método de detecção de erro mais eficiente para validação de uma sequência de dados seriais, com um mínimo montante de hardware.

Seu funcionamento é simples, para verificar a integridade dos dados o dispositivo mestre recalcula o código CRC para determinada sequência de bits e o compara com o valor CRC calculado e enviado pelo transmissor juntamente com os dados. Se ambos os valores CRC são iguais, os dados são validados livres de erro.

A Figura 17 apresenta a configuração de hardware para o cálculo do valor CRC, que consiste em 8 registradores de deslocamento com retroalimentação, e usualmente é vista como uma função polinomial de uma variável x qualquer.

Figura 17 – Circuito digital do cálculo de CRC de 8 bits



Fonte: Application Note 27, (2001).

Uma importante propriedade do circuito é que se o valor final do CRC de uma cadeia de dados calculado for inserido novamente no circuito, o valor resultante de todos os bits será zero. Logo, para os dados serem validados, o transmissor envia os dados junto com o byte CRC, este sendo o último byte transmitido, e todos os registradores devem conter bit ‘0’ ao final da passagem dos dados pelo circuito CRC.

4 DESENVOLVIMENTO

A partir do exposto até o momento, segue-se a explanação de como foi desenvolvida a integração das partes constituintes do projeto, mostrando as dificuldades transpostas em sua concretização e as soluções adotadas a fim de contorná-las.

Realizando as pesquisas iniciais para determinar as diretivas de projeto (*basis of Project*), percebeu-se a viabilidade em se usar o protocolo MQTT, devido às suas vantagens já descritas, notadamente seu pouco uso de hardware e largura de banda e facilidade de implementação, e ainda sua vasta disseminação no campo da IoT, o que chancela sua eficácia.

O primeiro passo foi a propositura da arquitetura da rede. Pensada inicialmente com uma central de controle constituída também de um microcontrolador, com o avanço dos estudos sobre o tema notou-se que essa função de gerenciamento seria lograda pelo *broker* referente ao protocolo MQTT, e desde que este requer aproximadamente 10MB de memória, foi dispensado o uso de um microcontrolador de baixo custo, devido à sua pequena capacidade de memória, deixando a atribuição de hospedar a unidade central de controle, a saber, o *broker*, ao próprio PC do autor e ao servidor na nuvem.

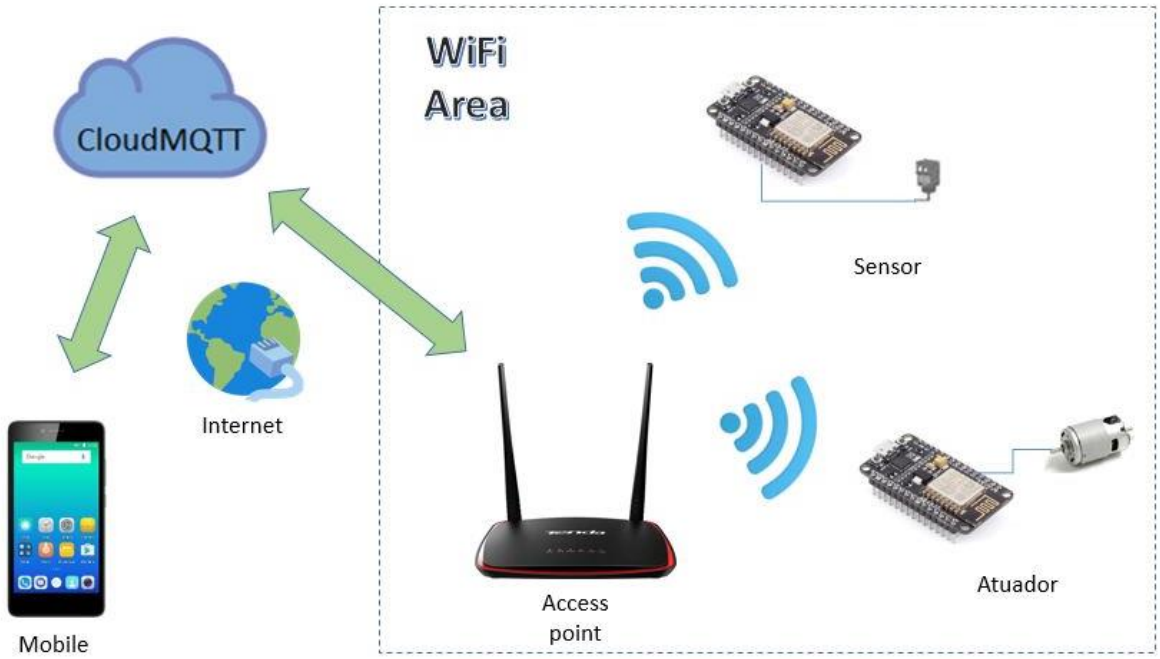
A última seção deste capítulo apresenta o aplicativo utilizado para controle e monitoramento dos sistemas da residência, utilizado como interface com o usuário final, e a plataforma de testes elaborada para demonstração da operação do sistema e validação do projeto.

4.1 Arquitetura da rede

A arquitetura da rede foi subdividida em dois modelos, uma para acesso remoto, via internet, e outra para acesso local, dispensando o uso da internet

A primeira optou-se por utilizar o PaaS CloudMQTT, servidor MQTT que executa o software Mosquitto na nuvem, por oferecer uma opção de pacote grátis e um serviço eficaz e fácil de implementar. Seu modo de funcionamento é simples e intuitivo: o aplicativo se conecta ao CloudMQTT e este se comunica com o roteador, que, por sua vez, já pela rede WiFi local, se comunica com os sensores e atuadores, como pode ser visto na Figura 18.

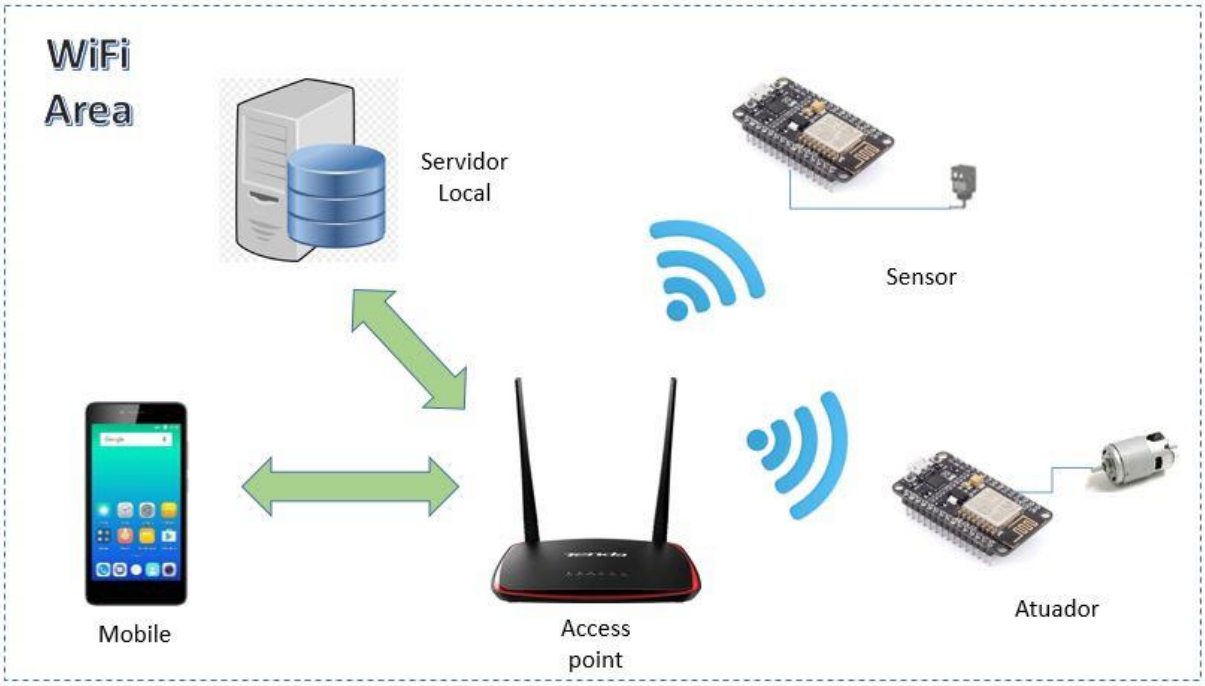
Figura 18 - Topologia da rede para acesso remoto



Fonte: AUTOR.

Já para o acesso local, que está esboçado na Figura 19, foi instalado o *Mosquitto* no próprio PC em funcionamento na residência, *software* que implementa a função do *broker* pertencente ao protocolo MQTT e gerencia o tráfego de mensagens pela rede. Operando todo via *Wifi*, sem uso de outras tecnologias, o usuário do sistema *android* acessa o servidor local através do *switch* (terminologia correta para designar o roteador de residências), e este realiza a comunicação com os clientes nós da rede, ou seja, os sensores e atuadores.

Figura 19 – Topologia da rede para acesso local

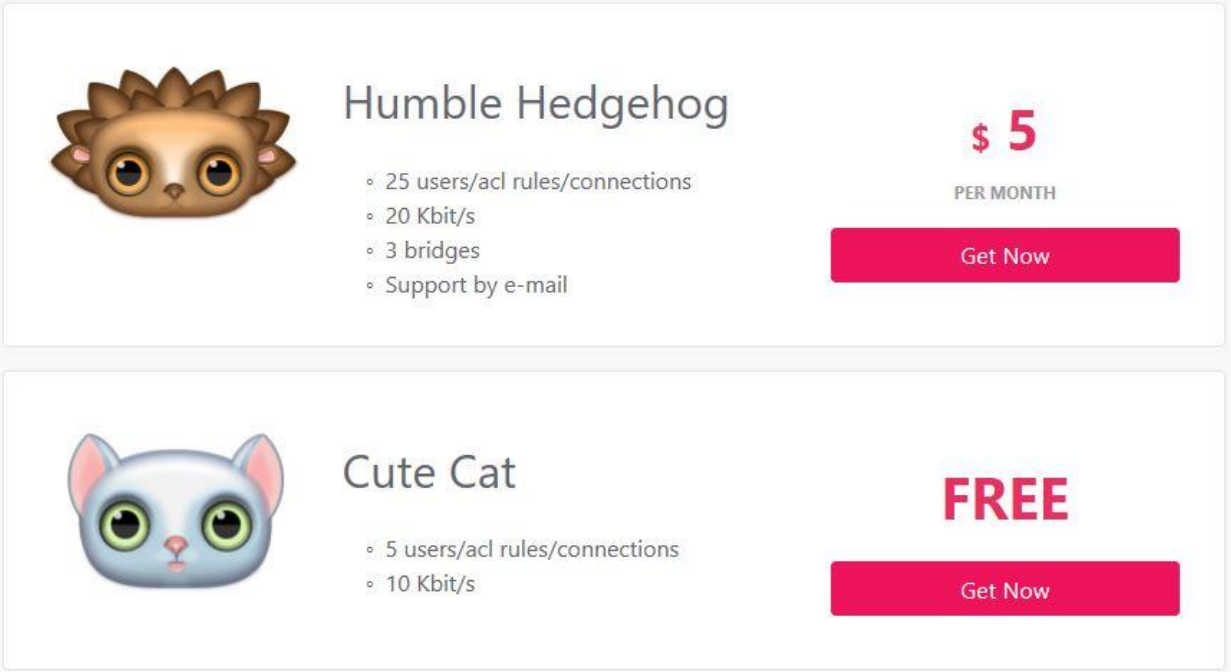


Fonte: AUTOR.

Ato contínuo, se fará uma explicação de como fazer uso da *Platform as a Service* (PaaS) CloudMQTT, e alguns recursos oferecido pela mesma.

A Figura 20 mostra os pacotes mais acessíveis oferecidos pela plataforma CloudMQTT, que variam de gratuito a duzentos e noventa e nove dólares por mês. O pacote gratuito suporta até 5 usuários e taxa máxima de transmissão de dados de 10Kbit/s, fazendo-o ideal para testes ou projetos de pequeno porte.

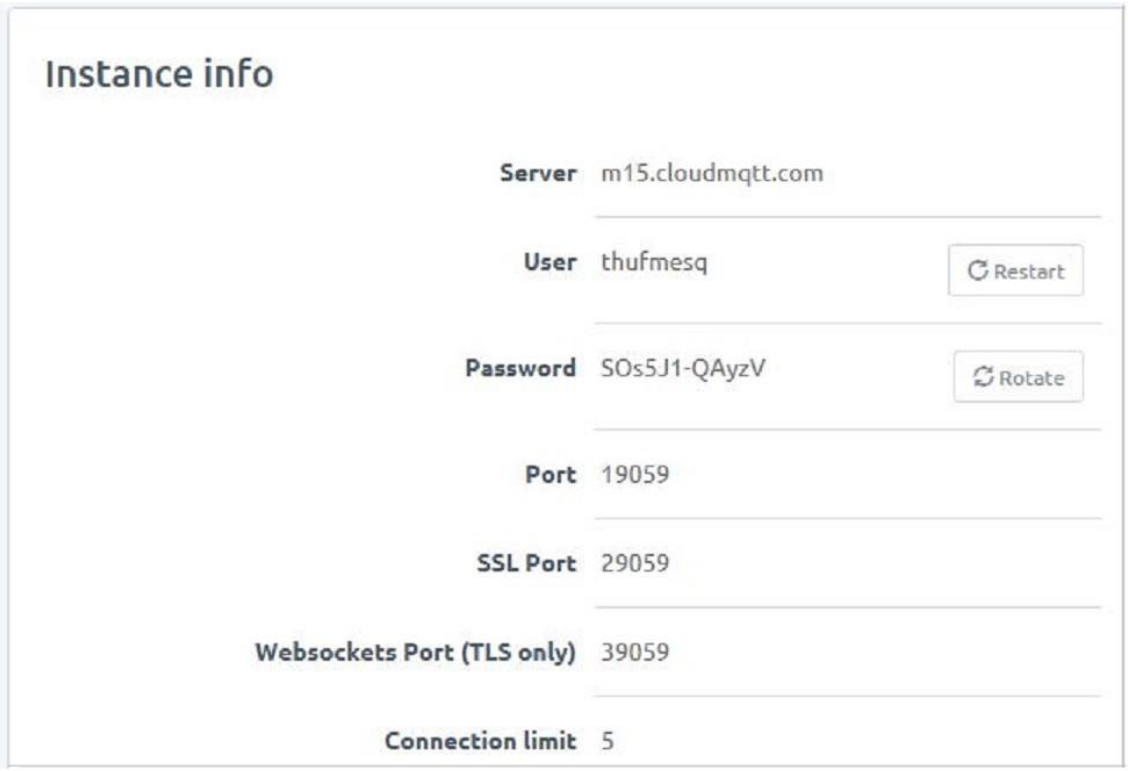
Figura 20 – Pacotes mais simples oferecido pela PaaS CloudMQTT



Fonte: AUTOR.

Após criada a conta, na aba *Details* encontra-se as informações para estabelecer a conexão à nuvem do *Mosquitto*, que serão inseridos no código do algoritmo, como o nome do servidor, nome e senha do usuário e a porta à qual se fará a conexão, esta última disponível com diferentes tipos de segurança, conforme explicita a Figura 21.

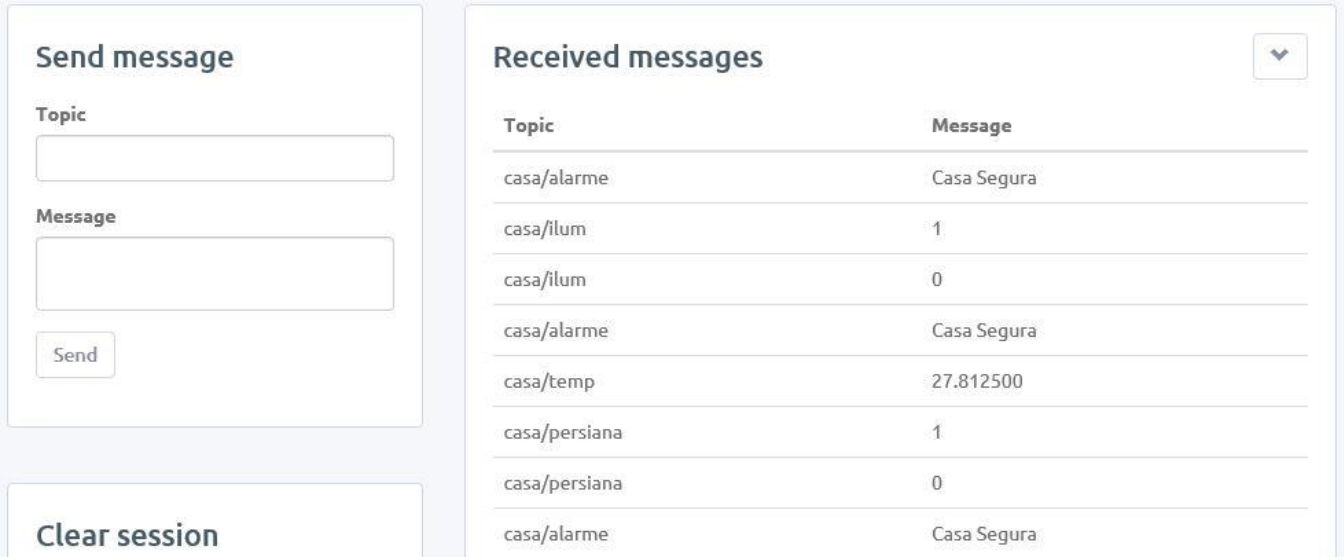
Figura 21 – Detalhes da conta no CloudMQTT



Fonte: AUTOR.

Com o sistema em funcionamento, na aba WEBSOCKET UI (*User Interface*), podemos visualizar e monitorar as mensagens e os respectivos tópicos atribuídos a estas, mostrado na Figura 22, bem como publicar uma mensagem/comando num determinado tópico para ligar uma luz ou fechar uma janela, como exemplo.

Figura 22 – Monitoramento das mensagens trafegadas na rede

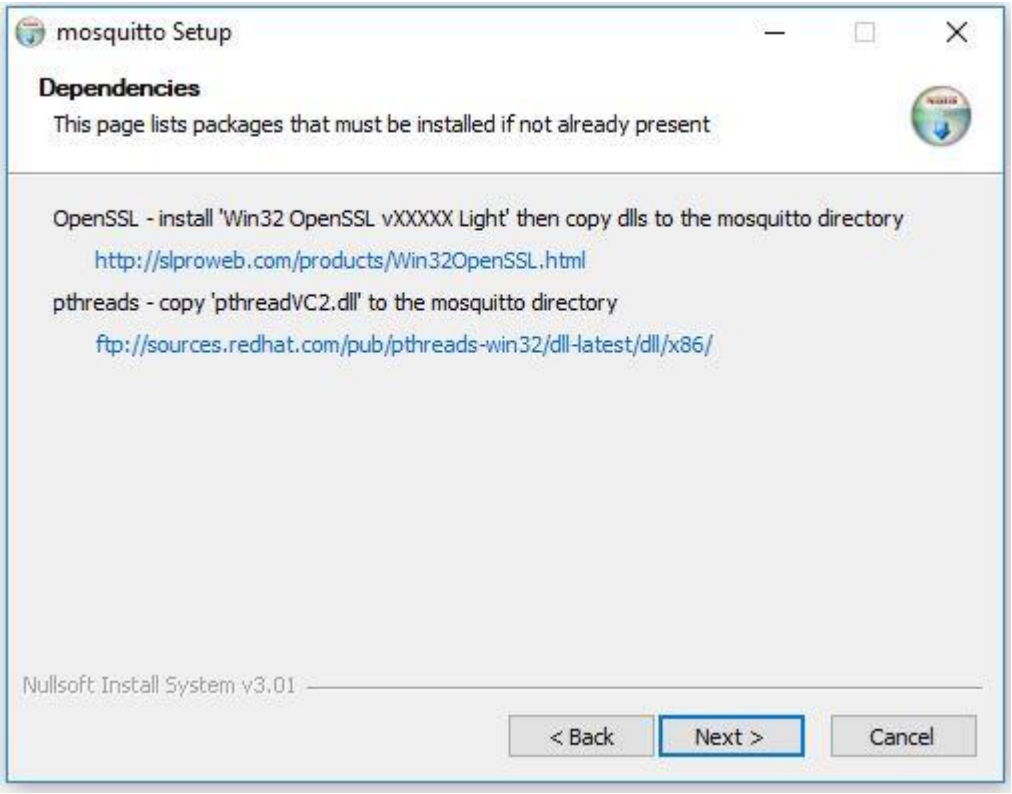


Fonte: AUTOR.

Após compreender o uso do servidor remoto, segue-se abordando o *software Mosquitto*, que é rodado tanto localmente quanto na nuvem, embora nesta seja de modo mais abstrato. Sua instalação não é de todo trivial, e seguindo-se os passos corretos descritos a seguir, pode ser efetuada com sucesso.

A versão 1.4.11 de 32 bits do *Mosquitto* para SO *Windows* 10 foi a utilizada neste projeto, baixada diretamente do site oficial do programa. Ao abrir o arquivo .exe de instalação, será necessário direcionar-se para a URL do programa *OpenSSL*, que é instalado em conjunto com o *Mosquitto*, e do arquivo *pthreadVC2.dll*, este que será copiado para o interior da pasta de arquivos do Mosquitto ao final de sua instalação. A janela de instalação descrita pode ser vista na Figura 23. Na janela posterior, deve-se deixar selecionado o componente s*ervice* e prosseguir com a instalação padrão.

Figura 23 – Instalação do broker Mosquitto no servidor local

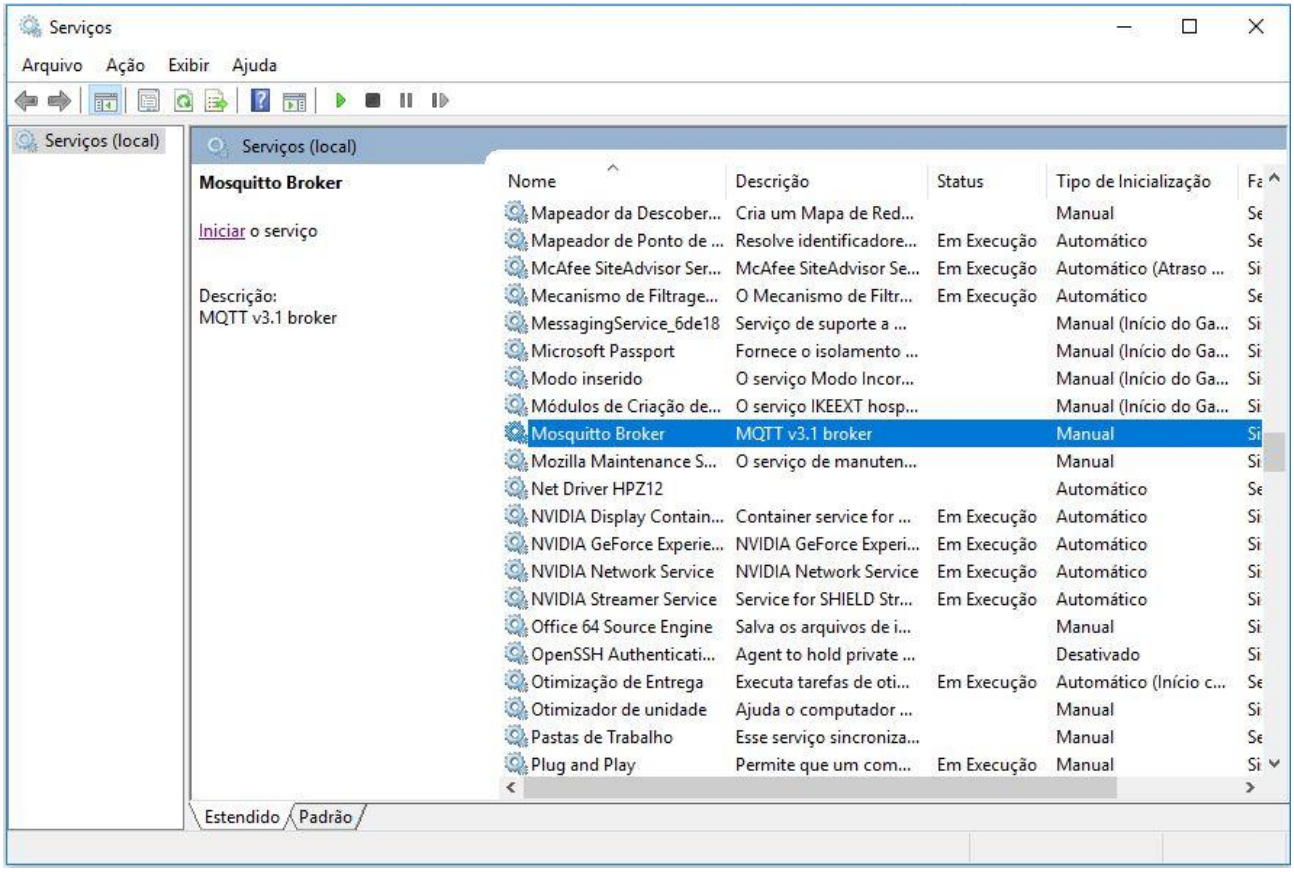


Fonte: AUTOR.

Após o término, então, copia-se o arquivo *pthreadVC2.dll* para a pasta recém instalada do *Mosquitto*, o que finaliza a sua instalação.

O próximo passo é acessar a lista de serviços do *Windows* e iniciar sua execução, como pode ser visualizado na Figura 24. Desta forma, o protocolo MQTT está hábil para ser executado localmente, sem o uso da Internet.

Figura 24 – Inicialização do serviço do Broker Mosquitto



Fonte: AUTOR.

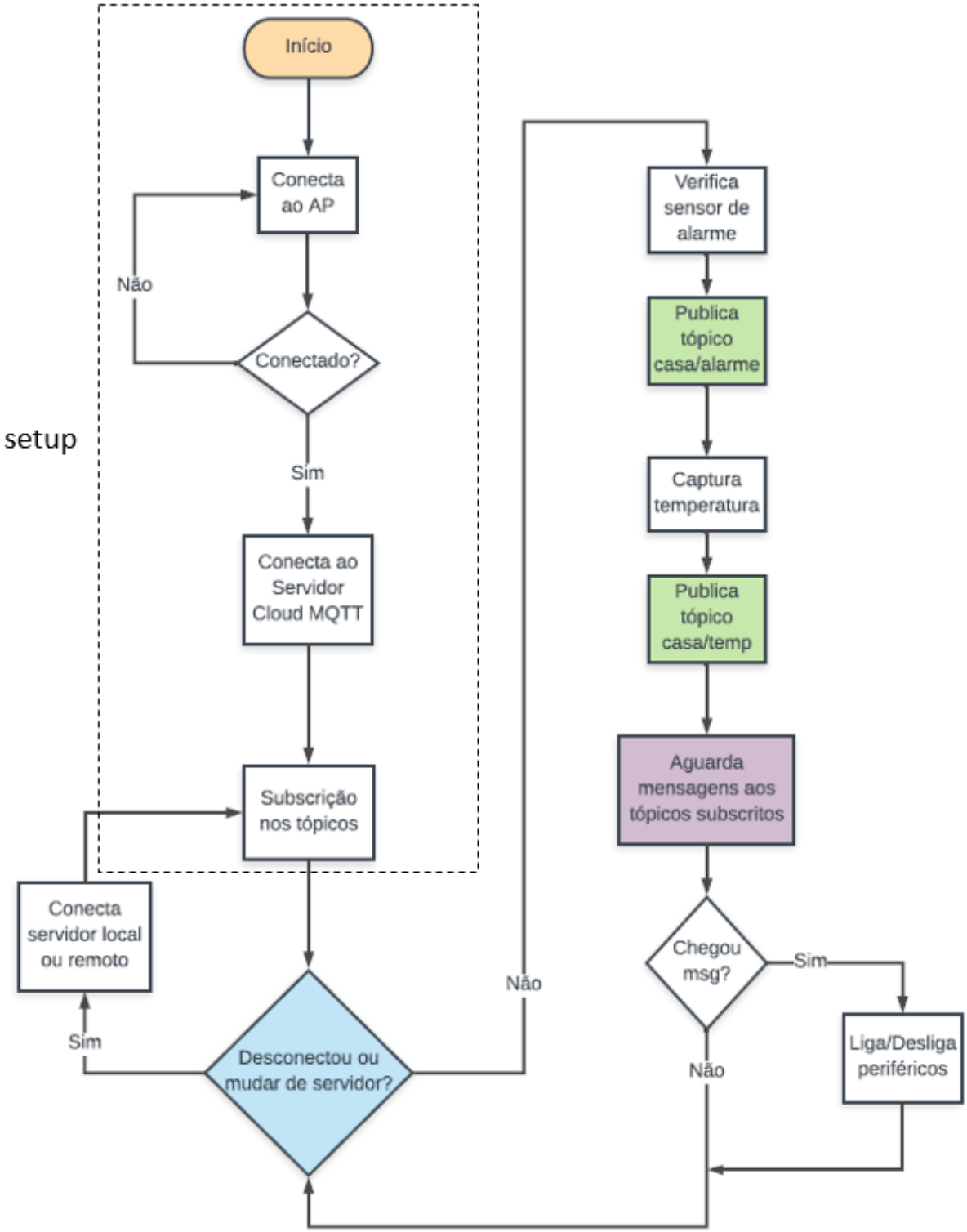
4.2 Algoritmo embarcado

O algoritmo a ser executado no microcontrolador embarcado com sensor/atuador é composto de 6 funções prioritárias:

- *setup*: a qual são configurados a conexão ao *access point* e ao servidor CloudMQTT
- *reconect*: em caso de perda da conexão ao servidor ou solicitação de mudança de servidor (local ou remoto), efetua o estabelecimento da rede
- *temp*: faz a aquisição da temperatura do ambiente e publica ao *broker* (servidor)
- alarme: monitora o estado do sensor e envia a informação a cada cinco segundos
- *receivedCallback*: Trata as mensagens recebidas nos tópicos inscritos. Esta função faz parte da biblioteca *PubSubClient.h*, que implementa o protocolo MQTT
- *loop*: função principal do programa, que é executada *ad infinitum*

Na Figura 25, um fluxograma sintetizado da lógica do programa é mostrado, onde os itens em verde e roxo destacam a ação de publicação e recebimento das mensagens/comandos, respectivamente.

Figura 25 - Diagrama de fluxo do código do ESP8266.



Fonte: AUTOR.

Válido salientar que, embora todos os sensores e atuadores tenham sido instalados num mesmo ESP8266, apenas para efeitos didáticos, numa aplicação real cada microcontrolador (que é um *endpoint* na topologia da rede) fica incumbido de controlar um periférico, ou seja, estarão distantes entre si, contendo no código do programa, portanto, apenas a rotina referente ao respectivo periférico.

Uma explicação sucinta e objetiva do algoritmo é feita para uma melhor compreensão da programação utilizada no microcontrolador em questão. O ESP8266 se conecta a um *access point* com acesso à internet, e então conecta-se ao servidor CloudMQTT na nuvem a priori. Com essas duas conexões efetivadas com sucesso, o *server* e a função de tratamento das mensagens recebidas são definidas, pelos comandos *client.setServer* e *client.setCallback*, e a subscrição aos tópicos interessados é realizada, o que finaliza a função *setup*.

A próxima função, *reconect*, é usada para caso haja queda da conexão com o servidor (local ou remoto) e/ou quando o usuário estiver em sua residência e precisar, em caso de perda da Internet, ou quaisquer outros motivos, mudar para o servidor local instalado no PC da própria residência.

As funções *temp* e alarme efetuam constante execução para monitoramento das respectivas variáveis, com intervalos de trinta segundos e de cinco segundos, respectivamente, e, ao fim, realizam a publicação dessas informações no tópico referente a cada uma delas.

A rotina *receivedCallback* é uma função intrínseca ao MQTT, e realiza o processamento das mensagens recebidas, utilizando o *payload* e os tópicos para realizar as atividades requeridas pelo usuário. Ela não é chamada explicitamente, mas através da função *client.loop*, onde *client* é uma instância da biblioteca *PubSubClient.h*, que contém o protocolo MQTT.

A função *loop* contém a chamada das funções *reconect*, *temp*, alarme, e *cliente.loop*, mencionada no parágrafo anterior, que também tem a atribuição de manter a conexão com o servidor.

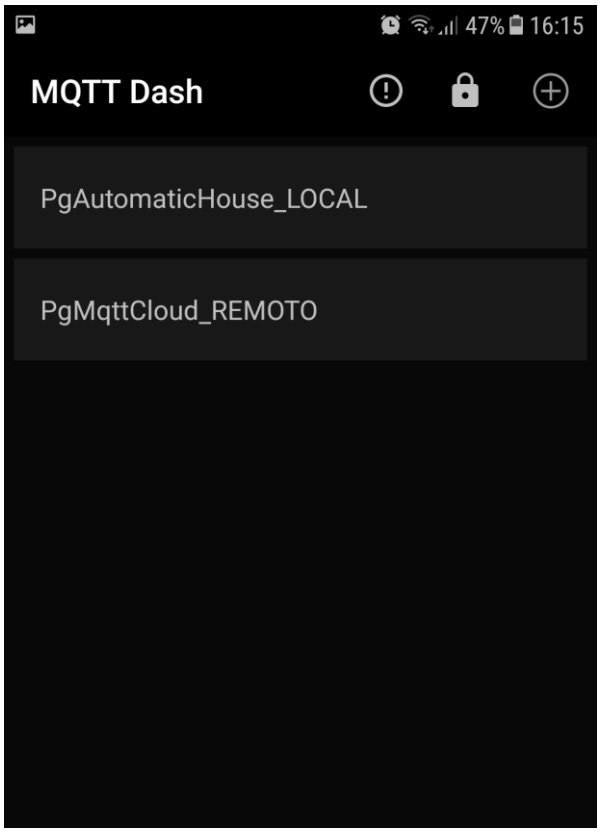
4.3 Automação IoT

Nesta seção será apresentado o aplicativo para interface do sistema com o usuário final e explanação da prototipagem elaborada para testes.

Após alguns testes, com principalmente dois aplicativos específicos para o protocolo MQTT, não obstante o *app MQTT Dashboard* se mostrar também muito eficiente, foi escolhido para interface com o usuário o *MQTT Dash*, pois este último se mostrou com funcionalidades interessantes e de manuseio mais agradável.

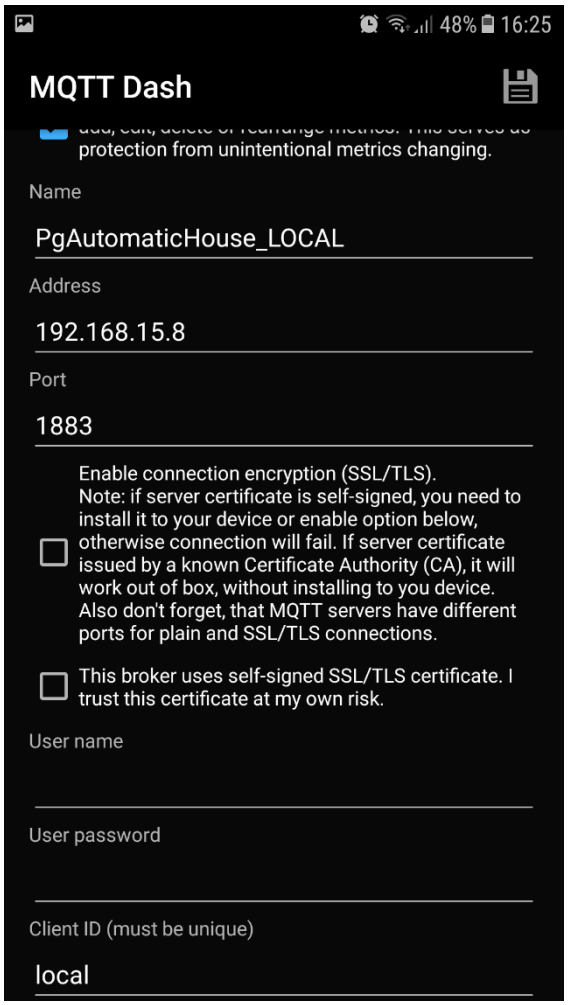
Para a configuração de um servidor, basta acessar o ícone de '+' na tela inicial do aplicativo e preencher com os dados da conta criada no CloudMQTT ou os dados do *server* local. Neste último caso, o IP do notebook e a porta padrão do protocolo MQTT, 1883, como visto na Figura 27; o recurso de encriptação foi deixado em branco. A Figura 26 mostra os servidores remoto e local já configurados.

Figura 26 – *Servers* local e remoto configurados para operar o sistema



Fonte: AUTOR.

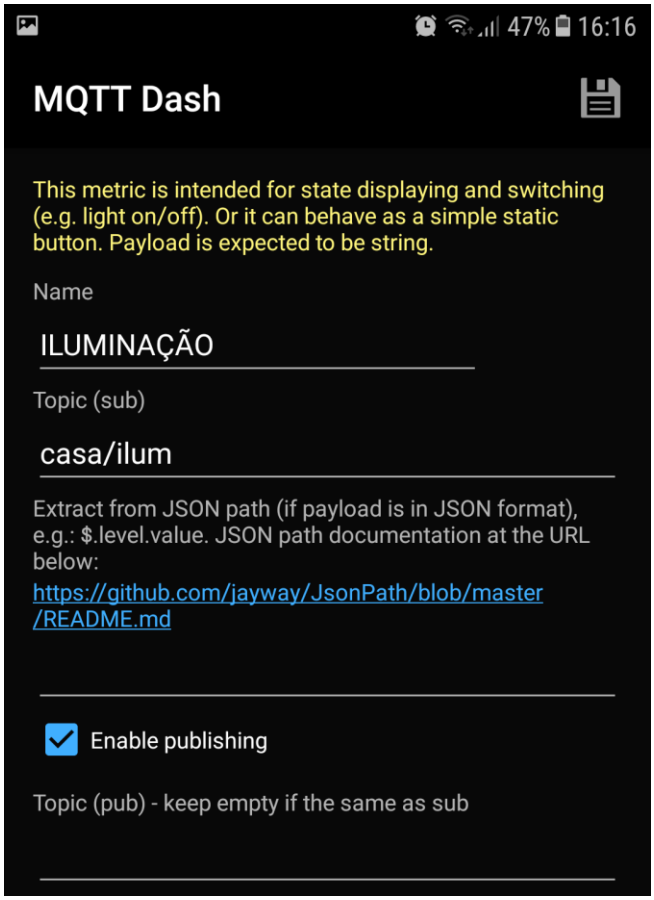
Figura 27 – Configuração do servidor no *app MQTT Dash*



Fonte: AUTOR.

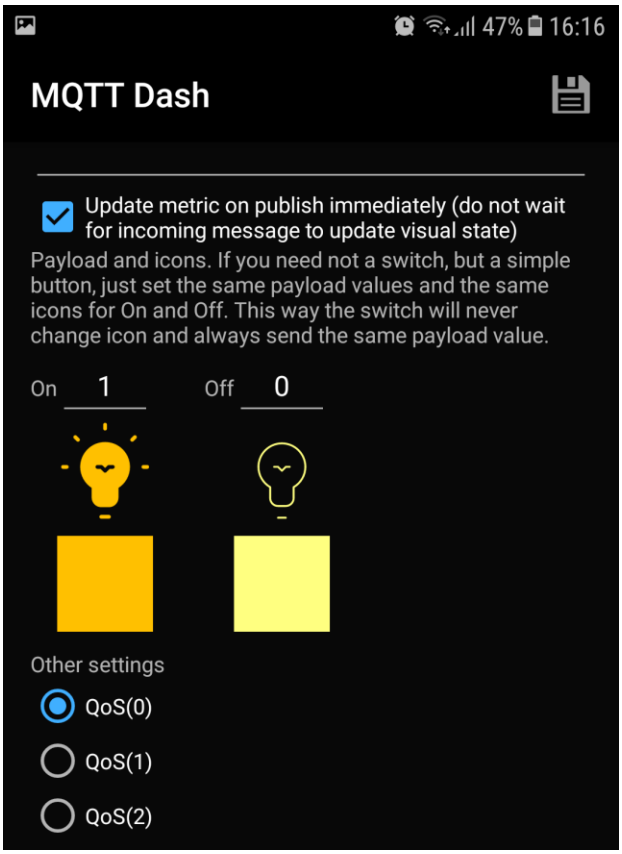
Ao escolher um dos servidores, prossegue-se à configuração das *metrics*, que são os ícones que desempenharão cada função desejada, tanto para publicar quanto subscrever-se nos tópicos para recebimento das mensagens. As duas próximas figuras ilustram a configuração de uma *metric* para comando de iluminação, onde definimos o tópico *subscribe* e *publish* para receber e enviar os comandos, respectivamente, e também o *payload* - conteúdo da mensagem, e o nível de qualidade de serviço pretendido.

Figura 28 – Configuração da *metric* para comando da iluminação.



Fonte: AUTOR.

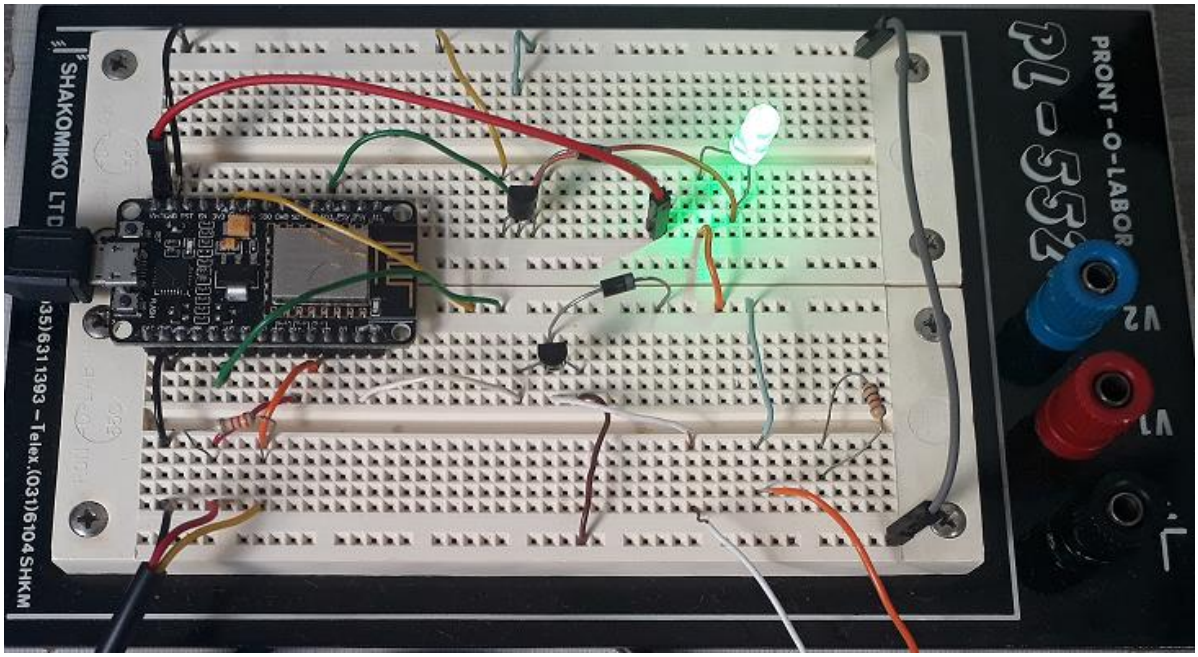
Figura 29 – Configuração do *payload* e QoS para a *metric* Iluminação



Fonte: AUTOR.

Na Figura 30 a seguir pode-se visualizar o circuito montado para ligar as portas do microcontrolador ao dispositivos, a fim de efetuar o controle dos mesmos.

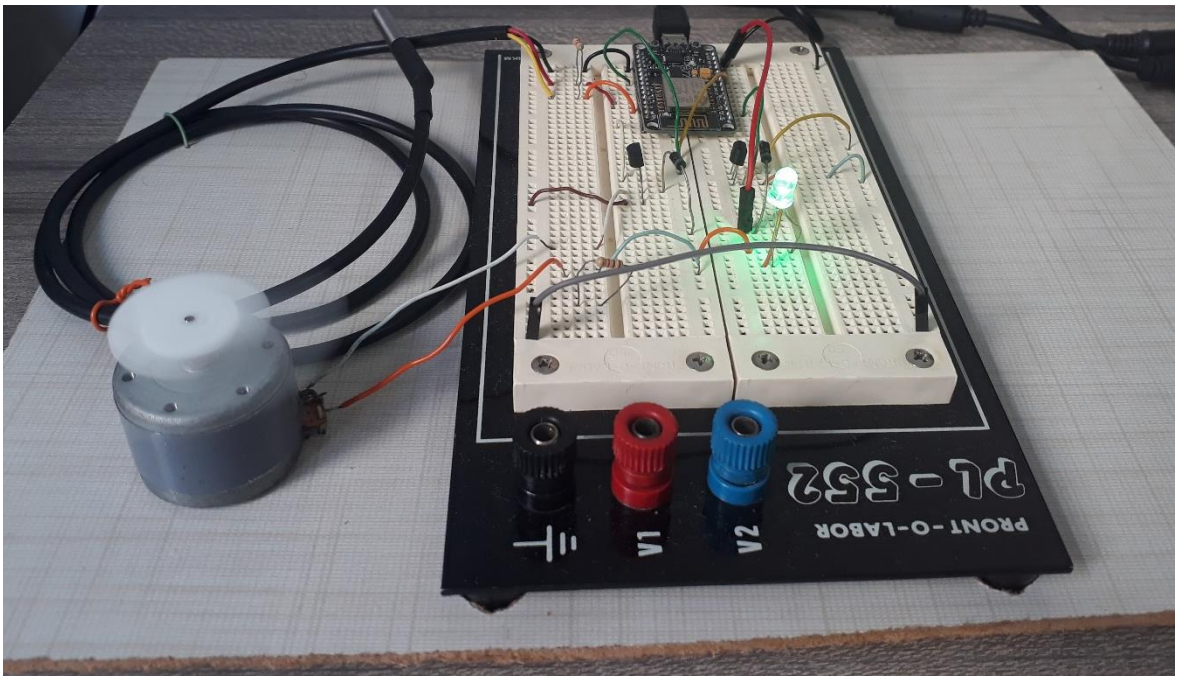
Figura 30 – Circuito elaborado para realização dos testes.



Fonte: AUTOR.

O protótipo para testes foi construído em uma plataforma de madeira MDF, acomodando o *protoboard* centralmente e os periféricos implantados lateralmente, como pode ser visualizado na Figura 31. Os periféricos para testes foram quatro, a saber, uma lâmpada, um motor DC de 5V, um sensor de temperatura e um sensor de alarme.

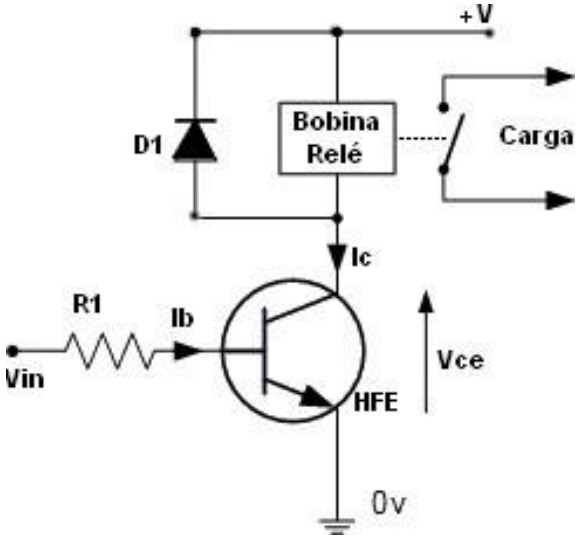
Figura 31 – Protótipo de testes com as cargas de iluminação e motor ligadas



Fonte: AUTOR

A fim de isolar os periféricos de potência, tais quais o motor e a lâmpada, foram utilizados transistores NPN como chave, ligando o terminal base à saída do ESP8266, de acordo com o esquema da Figura 32, dado que as saídas do microcontrolador suportam corrente máxima de 12mA, e no caso da lâmpada o transistor aciona um relé de cinco pontos, pois esta será ligada diretamente à rede elétrica.

Figura 32 – Diagrama elétrico do transistor para acoplar circuito de potência



Fonte: INTERNET.

5 RESULTADOS E DISCUSSÕES

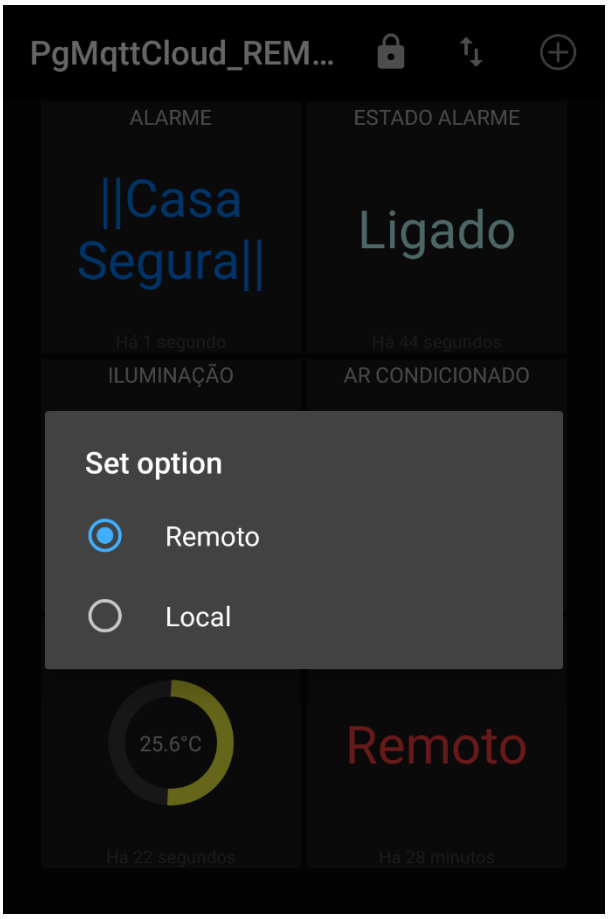
O grande obstáculo a ser transposto no corrente projeto foi a extensa pesquisa e reunião de todas as informações e métodos pertinentes para agregação final do projeto, sendo majoritariamente material de leitura, tais quais a especificação do protocolo utilizado e TCC's passados, estes últimos de grande valia para idealização do projeto, todos constando nas Referências Bibliográficas, bem como a pesquisa em sites que englobam a temática e vídeos explicativos que também auxiliaram em muito.

A interface gráfica da tela do aplicativo ficou como apresentado na Figura 33, onde visualizamos seis *metrics*, dois para indicar a situação do monitoramento e o estado do alarme, dois para controle de cargas, a saber, da iluminação e uma carga a motor qualquer, um para recebimento da temperatura do ambiente da residência e o último para a escolha do servidor MQTT utilizado, se remoto ou local, como pode ser conferido na Figura 34.

Figura 33 – Tela do app MQTT Dash de controle dos periféricos

Fonte: AUTOR.

Figura 34 – Escolha do servidor com rede local ou via Internet



Fonte: AUTOR.

Na Figura 35, podemos visualizar os ícones quando as cargas estão acionadas e a indicação da condição do alarme, no caso, indicando uma simulação de arrombamento.

Figura 35 – Cargas acionadas e em funcionamento

Fonte: AUTOR.

6 CONCLUSÕES

Ao fim e ao cabo, o trabalho atingiu com resultados promissores o objetivo a que se propôs ao entregar um sistema de automação que controla cargas em uma residência bem como monitora a situação do ambiente, como temperatura e mediante um esboço de implantação de um sistema de segurança através de um aparelho *smartphone*. E ainda, podendo ser controlada/monitorada em rede local bem como via acesso remoto por meio da internet.

A partir do contato com o atual estado da arte, o projeto foi desenvolvido utilizando as tecnologias mais atuais presentes na literatura, e, ao ter se mostrado funcional, com alguma ou nenhuma modificação poderia ser implementado num caso real. Nada impede, no entanto, que o presente projeto possa ser ampliado e aprimorado para trabalhos futuros, ou para ser colocado disponível comercialmente.

Válido salientar a oportunidade depreendida de aprimorar os conhecimentos adquiridos ao longo da formação, dada a extensão de possibilidades que o tema propicia, como por exemplo: aprender a sintaxe da linguagem HTML, que é utilizada principalmente nos *sites* da internet, (foi estudada a possibilidade de implementar a interface com o usuário via navegador da internet, com o uso do protocolo convencional HTTP); a dinâmica de funcionamento do termômetro digital; adquirir domínio acerca dos recursos e utilização de microcontroladores; estudar a especificação do protocolo da camada de aplicação MQTT.

Melhorias acerca podem ser feitas em projetos futuros, como sugestão, utilizando o próprio *switch* da residência como hospedeiro do *Broker Mosquitto*, eliminando a necessidade de manter o PC, até então utilizado como o servidor, sempre ligado. Isto pode ser feito com o *OpenWrt Project*, dentre outras opções, um mini SO *Linux* visando dispositivos embarcados, o que permite a instalação de aplicações diversas no firmware do dispositivo.

REFERÊNCIAS BIBLIOGRÁFICAS

TANENBAUM, A. S.; WETHERALL, D. **Redes de computadores**. 5. ed. São Paulo: Pearson Prentice Hall, 2011.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores e a Internet**: uma abordagem top-down. 5.ed. São Paulo: Addison Wesley, 2010.

MURATORI, J.R.; BÓ, P.H.D. **Automação Residencial: conceitos e aplicações.** 2. ed. – 1. reimpressão. Belo Horizonte: Educere Editora, 2017.

DEKEL, E. **Low-power Internet connectivity over Wi-Fi®**. Dallas: Texas Instrument Incorporated, 2016.

REITER, G. **Wireless connectivity for the Internet of Things**: One size does not fit all. Dallas: Texas Instrument Incorporated, 2014.

BAUM, A. **A link to the Internet of Things**: IoT made easy with SimpleLink™ Wi-Fi® solutions. Dallas: Texas Instrument Incorporated, 2014.

TEXAS INSTRUMENT. **CC3200 SimpleLink™ Wi-Fi® and IoT Solution with MCU LaunchPad Hardware User's Guide.** Dallas: Texas Instrument Incorporated, 2015.

ESPRESSIF SYSTEMS. **ESP8266-DevKitC Getting Started Guide Version 1.0.** Espressif IoT Team. Espressif Inc., 2018. Disponível em: <https://www.espressif.com>. Acesso em: set. 2018.

ESPRESSIF SYSTEMS. **ESP8266EX Datasheet Version 6.0.** Espressif IoT Team. Espressif Inc., 2018. Disponível em: <https://www.espressif.com>. Acesso em: set. 2018.

EVANS, D. **A Internet das Coisas: Como a próxima evolução da Internet está mudando tudo**. [S.L.]: Cisco IBSG, 2011.

SEEWALD, B.R. **Sistema de automação residencial de baixo custo para redes sem fio**. 2014. Trabalho de Graduação (Bacharelado em Engenharia de Computação) - Instituto de Informática, UFGRS, Porto Alegre.

JUNIOR, O.R. **Sistema de monitoramento residencial baseado em Internet das Coisas**. 2017. Trabalho de Conclusão de Curso (Bacharel em Engenharia Elétrica) – Departamento de Engenharia Elétrica, UEL, Londrina.

OLIVEIRA, R.R. **Uso do microcontrolador ESP8266 para automação residencial**. 2017. Projeto de Graduação (Título de Engenheiro de Controle e automação) – Escola Politécnica, UFRJ, Rio de Janeiro.

PITON, O.H.G. **Automação residencial utilizando a plataforma em nuvem IBM Bluemix**. 2017. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) – Escola de Engenharia de São Carlos, USP, São Carlos.

VICENTE, T.P.R. **Controle Inteligente de Vagas para Estacionamento Utilizando o Conceito de Internet das Coisas**. 2016. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Elétrica) - Escola de Engenharia de São Carlos, USP, São Carlos.

OASIS STANDARD. **MQTT Version 3.1.1.** Editado por Andrew Banks and Rahul Gupta.2014.

EGLI, P.R. **An introduction to MQTT, a protocol for m2m and IoT applications**. Disponível em <http://peteregli.net/content/iot/iot.html>. Acesso em 16 de nov. 2018

MICROCONTROLADORES. Disponível em: <http://www.roboliv.re/conteudo/microcontroladores>. Acesso em: 20 de jul. 2017.

MARCELO. **“IOT feito fácil”: Brincando com o ESP32 no Arduino IDE**. Disponível em: < https://mjrobot.org/2017/09/26/iot-feito-facil-brincando-com-o-esp32-no-arduino-ide/>. Acesso em: 8 de set. de 2018.

DOS REIS, V.R. **Como medir temperatura com um DS18B20**. Disponível em: <http://www.arduinobr.com/arduino/arduino-sensor/como-medir-temperatura-com-um-ds18b20/>. Acesso em: 17 set. 2018.

TUAN. **Demo 14: How to use MQTT and Arduino ESP32 to build a simple Smart home system**. Disponível em: < http://www.iotsharing.com/2017/05/how-to-use-mqtt-to-build-smart-home-arduino-esp32.html>. Acesso em: out. e nov. 2018.

O'LEARY, N. **Arduino Client for MQTT.** Disponível em: <https://pubsubclient.knolleary.net>. Acesso em: out. 2018.

YUAN, M. **Conhecendo o MQTT**. Disponível em: <https://www.ibm.com/developerworks/br/library/iot-mqtt-why-good-for-iot/index.html>. Acesso em: 18 de nov. 2018.

BRAGA, N.C. **Como funciona o Reed-Switches (MEC089)**. Disponível em: <http://www.newtoncbraga.com.br/index.php/como-funciona/3860-mec089>. Acesso em: 05 de dez. 2018.

KUMAR, N. ResearchGate. Disponível em: <https://www.researchgate.net/figure/Basic-Reed-switch-structure_fig4_282188064>. Acesso em: 5 de dez. 2018.

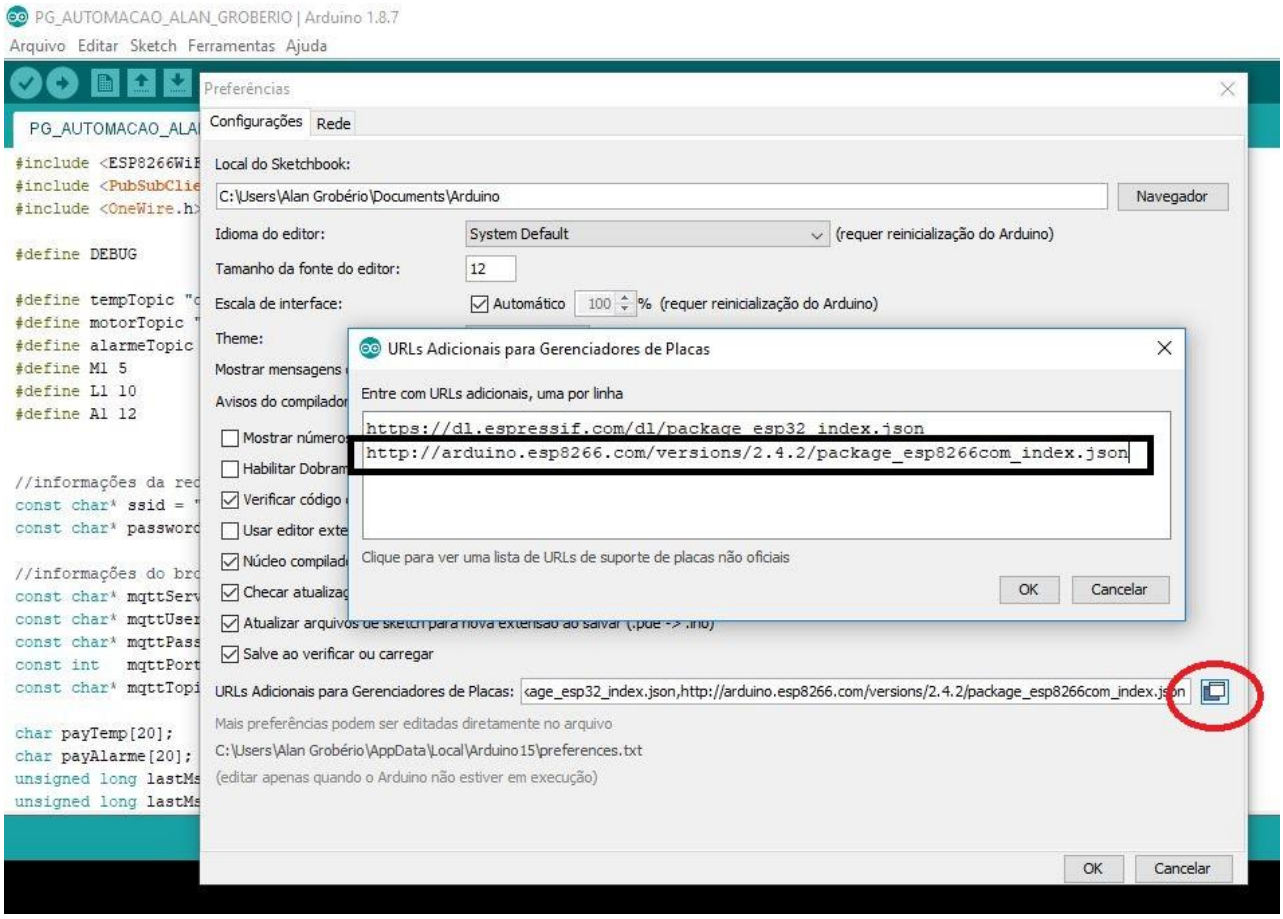
APENDICE A – INSTALAÇÃO DO ESP8266 NA IDE ARDUINO

No menu Arquivo/Preferências, deve-se adicionar a URL para apontar para o local de download do pacote com os arquivos necessários para instalação, conforme indicação na Figura 36.

A endereço URL é o que segue, e deve ser copiado e colado no local indicado na figura supracitada:

<http://arduino.esp8266.com/versions/2.4.2/package_esp8266com_index.json>

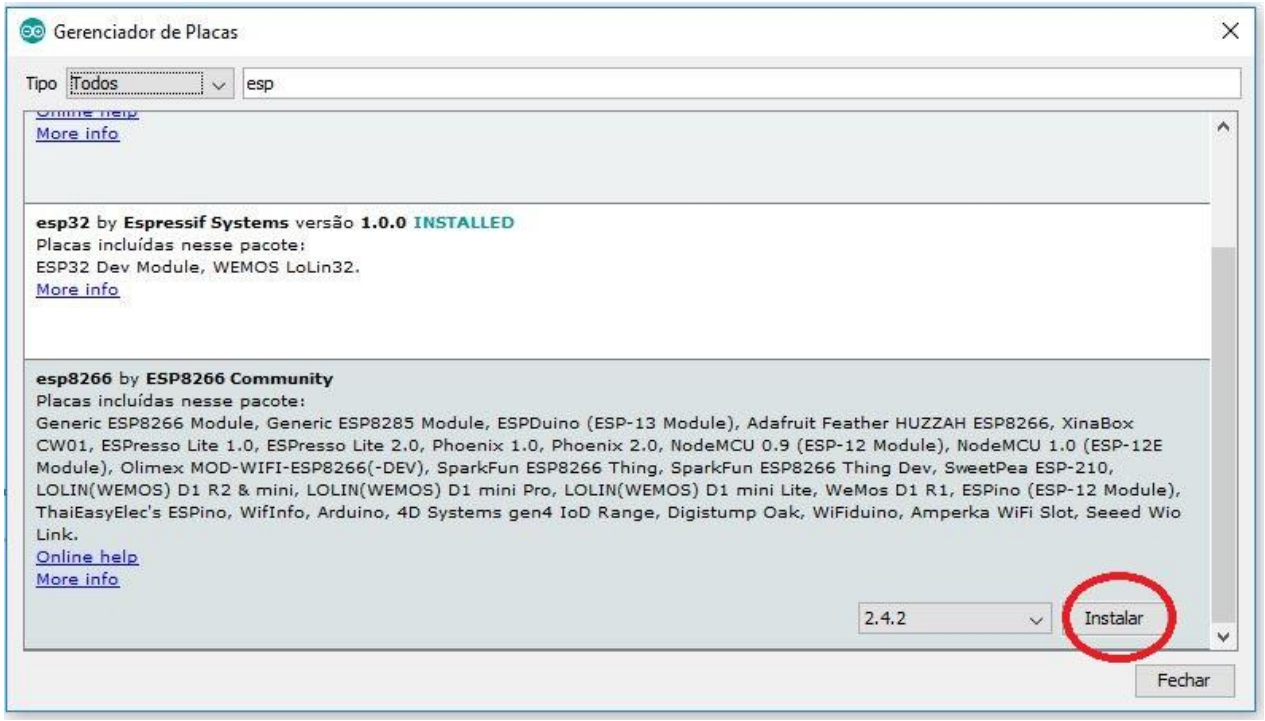
Figura 36 – Instalação do pacote do ESP8266 na IDE Arduino



Fonte: AUTOR.

O próximo passo aceder até Ferramentas/Placa/Gerenciador de placas/ para instalar a biblioteca relativa ao microcontrolador ESP8266, conforme visto na Figura 37, e aguardar o procedimento normal de instalação.

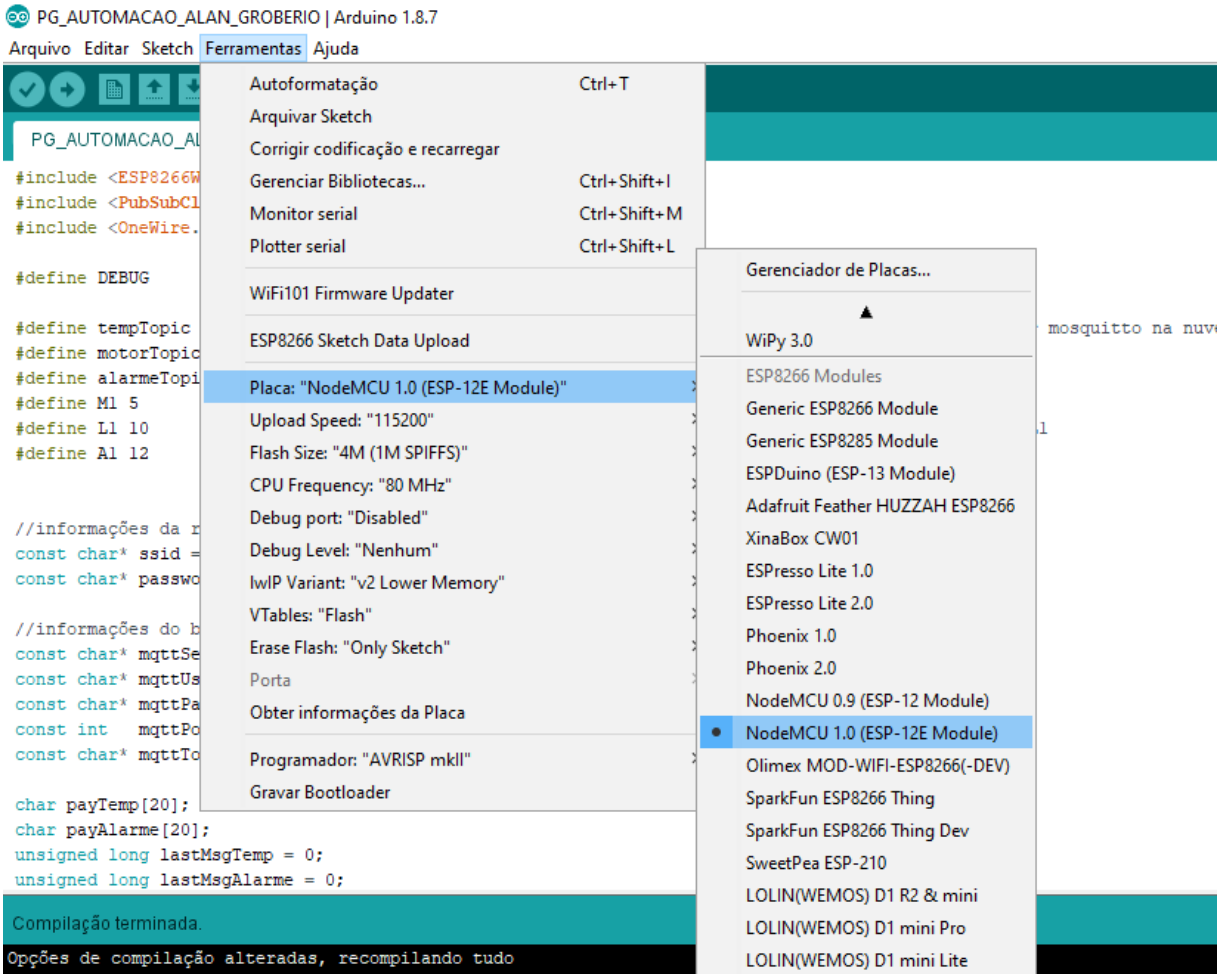
Figura 37 – Instalação da placa ESP8266



Fonte: AUTOR

Para selecionar a placa desejada para programá-la, procede-se de acordo com a Figura 38.

Figura 38 – Seleção da placa para programação



Fonte: AUTOR.

APÊNDICE B – CÓDIGO IOT

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <OneWire.h>

#define DEBUG

#define tempTopic "casa/temp"                    //tópico de temperatura a ser publicado pelo
broker mosquitto na nuvem do cloudmqtt
#define ilumTopic "casa/ilum"
#define motorTopic "casa/persiana"
#define alarmeTopic "casa/alarme"
#define estadoTopic "casa/alarme/estado"
#define serverTopic "automaçao/server"

#define M1 5                                     //motor
#define L1 10        //SDD3                        //pino de saida para acionamento da Lampada L1
#define A1 15        //pino gpio15 não pode estar em nível alto para upload do código
//PINO 3 só fica em 0; pino 12 nao funciona digitalWrite


//informações da rede WIFI
const char* ssid = "RedeWifi";                //SSID da rede WIFI
const char* password = "*****";           //senha da rede wifi

//informações do broker MQTT - Verifique as informações geradas pelo CloudMQTT
const char* mqttServerLocal = "192.168.15.8";
const char* mqttServer = "m15.cloudmqtt.com";   //server
const char* mqttUser = "thufmesq";              //user
const char* mqttPassword = "SOs5J1-QAyzV";      //password
const int   mqttPort = 19059;                   //port
```

```
char payTemp[20];
char payAlarme[20];
char payServer;
unsigned long lastMsgTemp = 0;
unsigned long lastMsgAlarme = 0;
int intruso = 0;
int flagServer = 1;                    // 1 => Servidor Remoto    2 => Servidor local

WiFiClient espClient;
PubSubClient client(espClient);

OneWire ds(2);                         //instancia um objeto sensor no GPIO2

void setup() {

  // Atribui os modos de operação dos GPIO's
  pinMode(L1, OUTPUT);
  pinMode(M1, OUTPUT);
  pinMode(A1, INPUT);
  digitalWrite(A1, LOW);
  Serial.begin(115200);

  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
```

```
  Serial.println("");
  Serial.println("WiFi connected");
                                                    //Serial.println("IP address: ");
                                                    //Serial.println(WiFi.localIP());

  // configure the MQTT server with IPaddress (DNS) and port
  client.setServer(mqttServer, mqttPort);
  client.setCallback(receivedCallback);                          /* this receivedCallback
function will be invoked
                                                          when client received subscribed topic */

   while (!client.connected()) {

    Serial.println("Conectando ao Broker MQTT Remoto...");

    if (client.connect("ESP8266ClientRemoto", mqttUser, mqttPassword )) {
      Serial.println("Conectado na nuvem MQTT");
      flagServer = 1;
    }
    else {
      Serial.print("falha estado  ");
      Serial.print(client.state());
      delay(2000);
    }
  }

  client.subscribe(ilumTopic);                  //subscreve no tópico para receber comando de
acionamento da lâmpada
  client.subscribe(motorTopic);
  client.subscribe(serverTopic);

}
```

```
void reconect() {                          //função para reconectar ao servidor MQTT

  if(flagServer == 1){                      // Se MQTT Remoto se desconectou
  while (!client.connected()) {

    Serial.println("Conectando ao servidor CloudMQTT");
     client.setServer(mqttServer, mqttPort);
     client.setCallback(receivedCallback);

    bool conectado = strlen(mqttUser) > 0 ?                                  // t ? x:y  ==  se a
condição t é verdadeira x é executado, se t é falso y será avaliado
                client.connect("ESP8266ClientRemoto", mqttUser, mqttPassword) :
                client.connect("ESP8266Client");                          //retorna TRUE se a
conexão é bem sucedida

     if(conectado) {
      Serial.println("Conectado!");


                }
    else {
      Serial.println("Falha durante a conexão.Code: ");
      Serial.println( String(client.state()).c_str());
      Serial.println("Tentando novamente em 10 s");

      delay(10000);                           //Aguarda 10 segundos
        }
       }
     }

    else if (flagServer = 2){
          while(!client.connected()){
```

```
    Serial.println("Conectando ao Broker MQTT local");
    client.setServer(mqttServerLocal,1883);
    client.setCallback(receivedCallback);

    if (client.connect("ESP8266ClientLocal")){
    Serial.println("Conectado localmente ao MQTT");
    //flagServer = 2;
            }
    else{
    Serial.print("falha estado  ");
    Serial.print(client.state());
    delay(2000);
      }
     }
    }

    if(client.connected()) {
    Serial.println("Realizando subscrição aos tópicos");
    delay(2000);

    client.subscribe(ilumTopic, 1);                //subscreve no tópico da lâmpada com nivel de
qualidade: QoS 1       e sai do while
    client.subscribe(motorTopic, 1);
    client.subscribe(serverTopic, 1);
   }
}

void temp(){
  byte i;                          //A byte stores an 8-bit unsigned number, from 0 to 255
  byte present = 0;
  byte type_s;                        //flag para determinar o modelo do sensor
  byte data[12];                       //Array de bytes para leitura da memória SCRATCHPAD
do sensor que contem apenas 9 bytes
```

```
  byte addr[8];                          //Recebe o serial code, ROM de 8 bytes ( addr[0]=CRC
formado pelo resto debytes ; addr[1-7]= serial number ; addr[8]= family chip number)
  float celsius, fahrenheit;                  //número decimal de 32 bits

  //bool flag = ds.search(addr);
  if ( !ds.search(addr)) {                    //Procura pelo próximo sensor de maneira ordenada.
Caso encontre, o serial code é armazenado na variável addr, e true é retornado.
    ds.reset_search();                        //refaz a busca por dispositivos.Internamente a biblioteca
cria uma tabela onde mantém uma lista ordenada de sensores encontrados, e o
    //delay(2000);                            //próximo ds.search iniciará no primeiro dispositivo da
tabela
    return;
  }
  else{
    Serial.println("Temperatura do lar:");
    Serial.println();
  }

  if (OneWire::crc8(addr, 7) != addr[7]) {      //realiza a validação dos dados pela checagem da
geração do CRC da ROM, não da scratchpad
      Serial.println("CRC is not valid!");
      return;
  }

  switch (addr[0]) {                          // the first ROM byte indicates which chip
    case 0x10:
      Serial.println("  Chip sensor = DS18S20");  // or old DS1820
      type_s = 1;
      break;
    case 0x28:
      Serial.println("  Chip sensor= DS18B20");
      type_s = 0;
      break;
```

```
    case 0x22:
      Serial.println("  Chip sensor= DS1822");
      type_s = 0;
      break;
    default:
      Serial.println("Device is not a DS18x20 family device.");
      return;
  }

  ds.reset();              // reinicia o barramento 1-Wire, necessário antes da comunicação com
qualquer dispositivo
  ds.select(addr);         // seleciona o dispositivo a ser feito a comunicação, e permanece até o
próximo ds.reset
  ds.write(0x44, 1);       // escreve o comando no barramento para conversão AD, o 1 significa
with parasite power on at the end

  delay(1000);     //tempo para realizar conversão AD, 12 bits => 750ms         // we might do a
ds.depower() here, but the reset will take care of it.

  present = ds.reset();
  ds.select(addr);
  ds.write(0xBE);         // Read Scratchpad


  for ( i = 0; i < 9; i++) {           // Lê memória SRATCHPAD
    data[i] = ds.read();
  }

  // Convert the data to actual temperature because the result is a 16 bit signed integer, it
should
  // be stored to an "int16_t" type, which is always 16 bits even when compiled on a 32 bit
processor.
```

```
  int16_t raw = (data[1] << 8) | data[0];       //concatena byte 1 e 0, raw contém a temperatura
lida     // <<8 desloca 8 bits para a esquerda, completando com 0.   | lógica OR bit a bit
  //Serial.print(raw, BIN);

  //verifica o tipo de sensor
  if (type_s) {                                  //se sensor DS18S20
    raw = raw << 3; // 9 bit resolution default
    if (data[7] == 0x10) {
      // "count remain" gives full 12 bit resolution
      raw = (raw & 0xFFF0) + 12 - data[6];
    }
  } else {                                       //se sensor DS18B20 ou DS1822
    byte cfg = (data[4] & 0x60);                         //cfg recebe o registrador de configuração,
que determina a resolução utilizada na conversão A/D
                                                 //60h = 01100000b e deixa apenas os bits R1 e R0
    // at lower res, the low bits are undefined, so let's zero them
    if (cfg == 0x00) raw = raw & ~7;  // 9 bit resolution, 93.75 ms       // & lógica AND bit a bit
~ lógiva NOT, inverte os bits       7=> 0111  ~7=> 1000  limpa 3 últimos bits
    else if (cfg == 0x20) raw = raw & ~3; // 10 bit res, 187.5 ms
3=> 0011  ~3=> 1100
    else if (cfg == 0x40) raw = raw & ~1; // 11 bit res, 375 ms
~1=> 1110
    //// default is 12 bit resolution, 750 ms conversion time
  }
  celsius = (float)raw / 16.0;
  fahrenheit = celsius * 1.8 + 32.0;
  Serial.print("  Temperature = ");
  Serial.print(celsius);
  Serial.print(" Celsius, ");
  Serial.print(fahrenheit);
  Serial.println(" Fahrenheit");
  Serial.println();
```

```
if (!isnan(celsius)) {
      snprintf (payTemp, 20, "%lf", celsius);
      client.publish(tempTopic, payTemp);              // publica a mensagem no tópico temp/sala
contendo o valor da temperatura
    }
}

void alarme() {

  digitalWrite(A1, LOW);
  intruso = digitalRead(A1);
  Serial.println(intruso);                          //debug

  if ( intruso == HIGH ){
      snprintf (payAlarme, 20, "%s", "Casa Segura" );
      client.publish(alarmeTopic, payAlarme);
      Serial.println("Alarme ATIVADO");
  }
  else {

      snprintf (payAlarme, 20, "%s", "*** INVASÃO***");
client.publish(alarmeTopic, payAlarme);
      Serial.println("***INVASÃO***");

  }
}


void receivedCallback(char* topic, byte* payload, unsigned int length) {

  Serial.print("Message received: ");
  Serial.println(topic);
```

```
Serial.print("payload: ");
for (int i = 0; i < length; i++) {

  Serial.print((char)payload[i]);
}
Serial.println();

//Comando para acender a lâmpada
if( String(topic) == ilumTopic ){

if ((char)payload[0] == '1') {
  digitalWrite(L1, HIGH);                  // we got '1' -> Liga
}
else {
  digitalWrite(L1, LOW);                   // we got '0' -> Desliga
  }
}

//Comando para ligar o motor DC
if( String(topic) == motorTopic ){

if ((char)payload[0] == '1') {
  digitalWrite(M1, HIGH);                   // we got '1' -> Liga
  Serial.println("Motor ATIVO");
}
else {
  digitalWrite(M1, LOW);                    // we got '0' -> Desliga
  Serial.println("Motor INATIVO");
  }
}

//Comando para comutação de server para Local ou Remoto
if (String(topic) == serverTopic){
```

```
    if ((char)payload[0] == '1'){
      client.disconnect();
      flagServer = 1;
      reconect();
    }

    else if ((char)payload[0] == '2'){
      client.disconnect();
      flagServer = 2;
      reconect;
    }

    else if((char)payload[0] == '0'){
      client.disconnect();
      flagServer = 0;
    }
  }
}

void loop() {

  if (!client.connected()) {
    reconect();
  }

  long now = millis();                //contabiliza o tempo em milessegundos desde que o microC
foi ligado ou reiniciado

  if(now - lastMsgAlarme >= 5000){
    lastMsgAlarme = now;
    alarme();
            }
```

```
  if(now - lastMsgTemp >= 15000){          //intervalo entre medições de temperatura
  lastMsgTemp = now;
  temp();
          }

  client.loop();
}
```