



Universidade Estadual de Londrina
Centro de Tecnologia e Urbanismo
Departamento de Engenharia Elétrica

Olair Ricardo Junior

Sistema de monitoramento residencial baseado em Internet das Coisas

Londrina

2017

Universidade Estadual de Londrina

Centro de Tecnologia e Urbanismo
Departamento de Engenharia Elétrica

Olair Ricardo Junior

**Sistema de monitoramento residencial baseado em
Internet das Coisas**

Trabalho de Conclusão de Curso orientado pelo Prof. Dr. Francisco Granziera Junior intitulado “Sistema de monitoramento residencial baseado em Internet das Coisas” e apresentado à Universidade Estadual de Londrina, como parte dos requisitos necessários para a obtenção do Título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Francisco Granziera Junior

Londrina
2017

Ficha Catalográfica

Olair Ricardo Junior

Sistema de monitoramento residencial baseado em Internet das Coisas - Londrina, 2017 - 67 p., 30 cm.

Orientador: Prof. Dr. Francisco Granziera Junior

1. Internet das coisas. 2. ESP8266. 3. NodeMCU. 4. MQTT. 5. Lua.

I. Universidade Estadual de Londrina. Curso de Engenharia Elétrica. II. Sistema de monitoramento residencial baseado em Internet das Coisas.

Olair Ricardo Junior

Sistema de monitoramento residencial baseado em Internet das Coisas

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Elétrica da Universidade Estadual de Londrina, como requisito parcial para a obtenção do título de Bacharel em Engenharia Elétrica.

Comissão Examinadora

Prof. Dr. Francisco Granziera Junior
Universidade Estadual de Londrina
Orientador

Prof. Dr. Marcelo Carvalho Tosin
Universidade Estadual de Londrina

Prof. Dr. Walter Germanovix
Universidade Estadual de Londrina

Londrina, 16 de fevereiro de 2018

Dedico este trabalho a todos aqueles que, de alguma forma,
auxiliaram para a concretização desta etapa.

Agradecimentos

Agradeço primeiramente a minha família e amigos, por fornecer-me as condições necessárias para conclusão desta importante etapa da minha vida.

“O destino é inexorável.”
(Bernard Cornwell)

Olair Ricardo Junior. **Sistema de monitoramento residencial baseado em Internet das Coisas.** 2017. 67 p. Trabalho de Conclusão de Curso em Engenharia Elétrica - Universidade Estadual de Londrina, Londrina.

Resumo

Este trabalho apresenta um sistema de monitoramento residencial de baixo custo, que utiliza conceitos de Internet das Coisas. A principal ferramenta utilizada foi o microcontrolador ESP8266, que através do protocolo MQTT envia pela *internet* os dados de sensores que monitoram uma residência. Foram utilizados sensores de temperatura, para alertarem possíveis incêndios, e *reed switches* para verificar se portas e janelas estão abertas ou fechadas. Foi utilizado o protocolo *Network Time Protocol* para sincronizar o relógio do microprocessador, desta maneira foi possível obter a data e hora exata em que os dados dos sensores foram adquiridos. Foi instalado um servidor MQTT localmente para evitar o uso de servidores públicos, onde não há privacidade, e também evitar o uso de servidores pagos, reduzindo o custo total do projeto. Foi citado neste trabalho as pilhas de protocolos TCP/IP e o protocolo MQTT. Para a programação do microcontrolador foi utilizada a linguagem de programação Lua juntamente com o *software* NodeMCU, que contém as bibliotecas de funções necessárias para a realização do trabalho. Os dados dos sensores dados podem ser lidos por meio de um aplicativo para celulares Android ou acessando o computador onde o servidor local foi instalado.

Palavras-Chave: 1. Internet das coisas. 2. ESP8266. 3. NodeMCU. 4. MQTT. 5. Lua.

Olair Ricardo Junior. **An Internet of Things based domestic monitoring system.** 2017. 67 p. Monograph in Electrical Engineering - Londrina State University, Londrina.

Abstract

This work presents a low cost home monitoring system based on Internet of Things. The main tool is the microcontroller ESP8266, that uses MQTT protocol to send data gathered by home monitoring sensors. Temperature sensors were used to monitor room temperature and sense/alarm fire. Reed switches were used to monitor windows and doors. The Network Time Protocol was used to synchronize the microcontroller clock, thus, it was possible to acquire the exact time that data was gathered. An MQTT server was installed in a local machine to avoid public servers, where there are no privacy data. Other reason to use a local server was to keep a low cost system. This work has a brief introduction to TCP/IP protocol stack and MQTT protocol. To program the microcontroller the language Lua was used along the NodeMCU software, which is a library that contains all the function needed to develop the system. The sensors data can be read using a mobile app or through a computer where the server was installed.

Key-words: 1. Internet of things. 2. ESP8266. 3. NodeMCU. 4. MQTT. 5. Lua.

Listas de ilustrações

Figura 1 – A IoT surge entre 2008 e 2009	22
Figura 2 – Diagrama do sistema a ser construído	22
Figura 3 – Modelos OSI e TCP/IP	25
Figura 4 – Camadas de protocolos	27
Figura 5 – Inicialização da conexão entre <i>Client</i> e <i>Broker</i>	28
Figura 6 – Mensagem de conexão	28
Figura 7 – CONNACK message	29
Figura 8 – Publish message	30
Figura 9 – Subscriber message	31
Figura 10 – SUBACK message	31
Figura 11 – Níveis de qualidade de serviço no MQTT	33
Figura 12 – Exemplo de barramento I2C	34
Figura 13 – Condição de <i>start</i> e <i>stop</i> no barramento I2C	34
Figura 14 – Diagrama de blocos do ESP8266EX	35
Figura 15 – Placa de desenvolvimento NodeMCU V1	39
Figura 16 – Tela de Preferências do Arduino	40
Figura 17 – PCI contendo o TMP100	42
Figura 18 – Protótipo de testes	44
Figura 19 – Circuito esquemático da conexão do <i>reed switch</i> ao microcontrolador	45
Figura 20 – Diagrama de blocos do código que envia dados dos sensores através de uma comunicação MQTT	46
Figura 21 – Conexão criada no MQTT Spy	46
Figura 22 – Placa de desenvolvimento reconhecida pelo Windows através da comunicação USB.	49
Figura 23 – Console do ESPlorer mostrando a temperatura lida do TMP100	50
Figura 24 – Console do ESPlorer mostrando a comunicação MQTT sendo estabelecida	50
Figura 25 – Dashboard do HiveMQTT mostrando o tópico utilizado pelo autor	51
Figura 26 – MQTT Spy mostrando dados recebidos	51
Figura 27 – Aplicativo IoT MQTT Dashboard mostrando os dados de temperatura	52
Figura 28 – Aplicativo IoT MQTT Dashboard mostrando os dados de temperatura e quando estes dados foram lidos	53
Figura 29 – Aplicativo IoT MQTT Dashboard mostrando dados do <i>reed switch</i>	54
Figura 30 – <i>Prompt</i> de comando do Windows exibindo as conexões ativas	55
Figura 31 – <i>Broker</i> local sendo utilizado	55

Figura 32 – Cliente recebendo dados do *Broker* local 56

Lista de tabelas

Tabela 1 – Canais e suas respectivas frequências	36
Tabela 2 – Comparativo entre microcontroladores	36

Lista de Siglas e Abreviaturas

ACK	<i>Acknowledgement</i>
ADC	<i>Analog-to-Digital Converters</i>
API	<i>Application programming interface</i>
AWS	<i>Amazon Web Services</i>
CONNACK	<i>Connect Acknowledgement</i>
CPU	<i>Central Processing Unit</i>
eLua	<i>embedded Lua</i>
GPIO	<i>General-Purpose Input/Outputs</i>
GPS	<i>Global Positioning System</i>
HTTP	<i>Hypertext Transfer Protocol</i>
IBM	<i>International Business Machines</i>
IDE	<i>Integrated Development Environment</i>
IEEE	<i>Institute of Electrical and Electronics Engineers</i>
I2C	<i>Inter-Integrated Circuit</i>
IoT	<i>Internet of Things</i>
JVM	<i>Java Virtual Machine</i>
LLC	<i>Logic Link Control</i>
μ C	Microcontrolador
MAC	<i>Media Access Control</i>
MQTT	<i>Message Queue Telemetry Transport</i>
NACK	<i>Not Acknowledge</i>
NTC	<i>Negative Temperature Coefficient</i>
NTP	<i>Network Time Protocol</i>
OLED	<i>Organic Light-Emitting Diode</i>
OSI	<i>Open System Interconnection</i>
PCI	Placa de Circuito Impresso
PUB	<i>Publisher</i>
QoS	<i>Quality of Service</i>
RFID	<i>Radio frequency identification</i>
RTC	<i>Real Time Clock</i>
SCL	<i>Serial Clock Line</i>
SDA	<i>Serial Data Line</i>
SDK	<i>Software Development Kit</i>
SMS	<i>Short Message Service</i>
SNTP	<i>Simple Network Time Protocol</i>

SPI	<i>Serial Peripheral Interface</i>
SUB	<i>Subscriber</i>
TCP	<i>Transmission Control Protocol</i>
UART	<i>Universal Asynchronous Receiver/Transmitter</i>
UDP	<i>User Datagram Protocol</i>
USB	<i>Universal Serial Bus</i>
WLAN	<i>Wireless Local Area Network</i>

Sumário

1	INTRODUÇÃO	21
1.1	Trabalhos relacionados	23
1.2	Justificativa	24
1.3	Objetivos	24
2	FUNDAMENTAÇÃO TEÓRICA	25
2.1	Pilha de protocolos TCP/IP	25
2.2	MQTT	27
2.3	I2C	33
2.4	ESP8266	35
2.5	Lua, eLua e NodeMCU	36
2.6	Sensores	37
3	DESENVOLVIMENTO	39
3.1	ESP8266	39
3.2	Testes iniciais	40
3.3	Utilização do protocolo MQTT	43
4	RESULTADOS E DISCUSSÕES	49
5	CONCLUSÕES	57
	REFERÊNCIAS	59
	Apêndice	66

1 Introdução

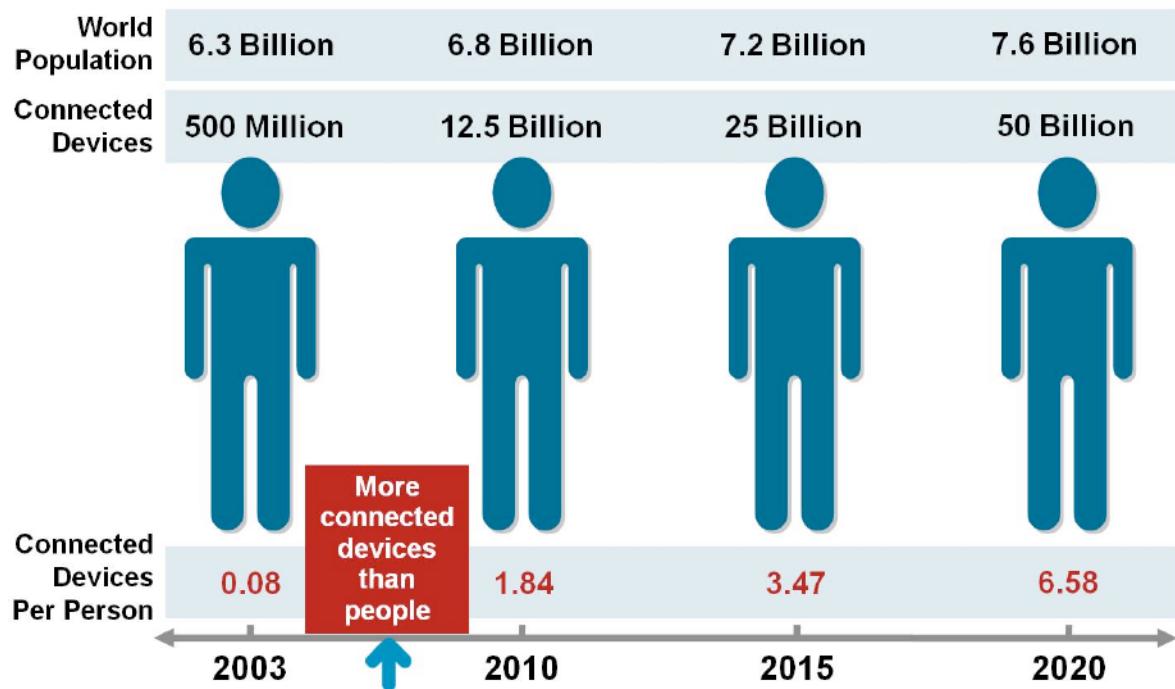
A Internet das Coisas é uma inovação que vem sendo muito discutida nos últimos anos devido aos avanços tecnológicos que tornaram tal conceito viável. Porém, foi em 1999 que o termo *Internet of Things* surgiu, como visto em Ashton (2009). Esse termo foi criado para nomear uma rede onde vários objetos estão conectados, e a partir dela, tais objetos não dependam dos humanos para adquirir dados e tomar decisões. Ou seja, os objetos estarão conectados entre si e em rede, de modo inteligente, e passarão a "sentir" e interagir com o ambiente ao redor. Ainda de acordo com Ashton (2009), quase toda a informação contida na internet foi adquirida por humanos, que devido a diversos fatores, não são bons o suficientes para capturar dados do mundo real. Tanto nós como o ambiente em que vivemos é físico. Então se os computadores fossem capazes de obter todas as informações das coisas físicas que nos cercam, coletando dados sem ajuda de humanos, seria possível contabilizar todos os recursos, desta forma, reduzir os custos, desperdícios e perdas. Sem a limitação humana de coletar dados, os computadores com a tecnologia de RFID e sensores poderiam observar, identificar e compreender o mundo. A Internet das Coisas tem o potencial de mudar o mundo.

O conceito básico da A Internet das Coisas, segundo Atzori Antonio Iera (2010), é a presença entre nós de vários objetos e coisas, tais como etiquetas RFID, sensores, atuadores, celulares, que são capazes de interagir entre si e cooperar com os dispositivos próximos para alcançar objetivos em comum. Tal tecnologia tem o potencial de causar um grande impacto no cotidiano dos usuários. Desde monitoramento remoto de pacientes, *e-health*, possibilidade de criar ambientes inteligentes, até tornar uma residência mais segura e também economizar energia, como mostrado em Pinto (2010).

Para Evans, D. (2011) a Internet das Coisas surgiu quando o número coisas e objetos conectados à internet ultrapassou a quantidade de pessoas conectadas. IoT pode ser considerada a próxima evolução da internet, evoluindo também a coleta, análise e distribuição de dados, estes que podem se transformar em conhecimento e sabedoria. Seguindo a definição exposta logo acima, a Figura 1 mostra quando o Internet das Coisas "nasceu".

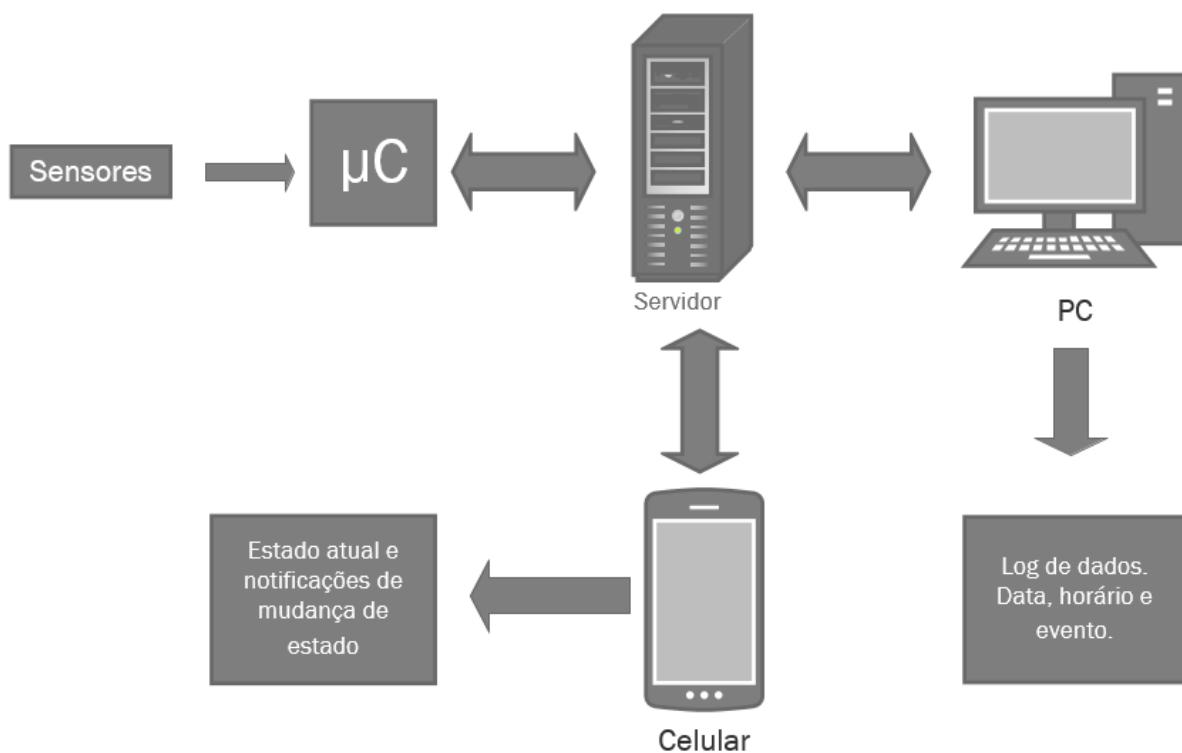
Para implementar um sistema de monitoramento residencial serão estudados os componentes necessários para conectar objetos. Estes componentes são: um microcontrolador com conexão Wi-Fi, os protocolos que regem a comunicação sem fio, os sensores para aquisição de dados. O sistema deve contar com diversos sensores conectados à um microcontrolador, que por sua vez estará conectado à internet e enviará os dados dos sensores para um servidor MQTT, este servidor deve distribuir os dados para um computador e para celulares. Um registro de dados ficará disponível com todos os dados recebidos pelo computador, já nos celulares, os usuários dos sistema poderão acompanhar os dados em tempo real. A Figura 2 apresenta o diagrama de blocos do sistema proposto.

Figura 1 – A IoT surge entre 2008 e 2009



Fonte: Evans, D. (2011).

Figura 2 – Diagrama do sistema a ser construído



Fonte: Autor.

1.1 Trabalhos relacionados

Atualmente existem vários trabalhos que abordam o monitoramento e automação residencial e monitoramento em geral, muitos utilizam hardwares que também serão utilizados neste trabalho. Este fato evidencia a viabilidade de um projeto de monitoramento residencial de baixo custo e seguro. Alguns trabalhos são citados abaixo.

Kodali e Mahesh (2016) monitoram a temperatura e a umidade de um ambiente utilizando módulo Wi-Fi ESP8266, um sensor DHT22 e um *display* OLED. A linguagem de programação utilizada foi a Micropython, que é uma implementação leve e eficiente da linguagem python 3. O trabalho justifica a utilização do Micropython fazendo uma comparação entre C/C++, e conclui que o desenvolvimento do trabalho é mais rápido com Micropython. Por fim é revelado que para tal projeto o *hardware* escolhido é economicamente viável, o sensor utilizado apresenta resultados precisos, mostrando que ele é confiável.

O trabalho feito por Kaur e Jasuja (2017) mostra um sistema de monitoramento de sinais vitais do corpo humano. Tal sistema é capaz de monitorar batimentos cardíacos e temperatura corporal. Os autores utilizam o Arduino UNO para realizar a leitura dos sensores e um Raspberry PI para enviar os dados da leitura pela internet. Foi utilizado o protocolo MQTT para a comunicação entre o sistema de monitoramento e o serviço de nuvem IBM Bluemix. O sistema proposto fornece dados precisos com baixo consumo de energia e baixo custo.

Foi apresentado por Kang, Park e Kim (2017) um serviço de IoT que monitora a temperatura de um ambiente e apresenta um alarme com um supressor de incêndio. Os autores implementam um *broker* MQTT no *Amazon Web Services* (AWS). Foi simulado um ambiente doméstico com sensor de temperatura, um alarme e um *sprinkler*. Os autores concluíram que o MQTT e o *Amazon Web Services* são uma boa escolha para serviços de IoT em pequenas aplicações, pois com o AWS é possível um acesso global à aplicação e os problemas de manutenção de servidor são eliminados.

Alzahrani (2017) apresenta diversas aplicações para a Internet das Coisas, como aquisição de dados de GPS, serviços de previsão do tempo, sensor de fumaça e um sistema de irrigação automatizado. Todas essas aplicações utilizaram um Arduino UNO R3 para fazer a leitura dos diversos sensores e um Arduino Wi-Fi *shield* para notificar um *smart phone*.

Os autores Saha e Majumdar (2017) propuseram um sistema de IoT para monitorar a temperatura de um centro de processamento de dados, e em caso da temperatura ultrapassar um limiar estabelecido notificações são enviadas em forma de SMS e *email*. O sistema consiste em posicionar estrategicamente sensores de temperatura conectados ao μ C ESP8266 que envia os dados, em intervalos regulares, para a nuvem da Ubidots. O microcontrolador foi programado em linguagem C utilizando o Arduino IDE. Um teste

revelou a estabilidade na performance do ESP8266. Apesar do *data center* já apresentar um sistema de monitoramento, as notificações tornaram o sistema mais robusto.

Em Kodali e Mandal (2016) é apresentado uma estação meteorológica. Foi utilizado o μC ESP8266 para adquirir os dados dos sensores de temperatura e umidade, pressão, detector de chuva e um sensor de luminosidade. O código foi escrito na linguagem de programação C utilizando o Arduino IDE. Os dados dos sensores são enviados para o serviço de nuvem IBM Bluemix. Quando a leitura destes sensores ultrapassa valores predeterminados são enviadas notificações para o usuário da aplicação.

1.2 Justificativa

O trabalho é justificado por abordar um assunto relevante e atual; pela possibilidade de implementação de projetos de baixo custo de monitoramento residencial; pela possibilidade de maior conforto e qualidade de vida dos usuários do sistema desenvolvido.

1.3 Objetivos

Este trabalho tem como objetivo desenvolver um sistema de monitoramento residencial de baixo custo e confiável utilizando conceitos de Internet das Coisas. Para que o projeto tenha uma baixo custo serão utilizados *softwares* livres. Este sistema contará com diversos sensores capazes monitorar de uma residência, como por exemplo: se há pessoas em determinado ambiente da casa, isto é feito utilizando sensores de presença e sensores de pressão; se as portas estão fechadas ou abertas, utilizando sensores magnéticos; a temperatura dos ambientes. Todos estes dados estarão disponíveis em tempo real em um aplicativo para celulares *Android* e ficarão salvos em um registro de dados.

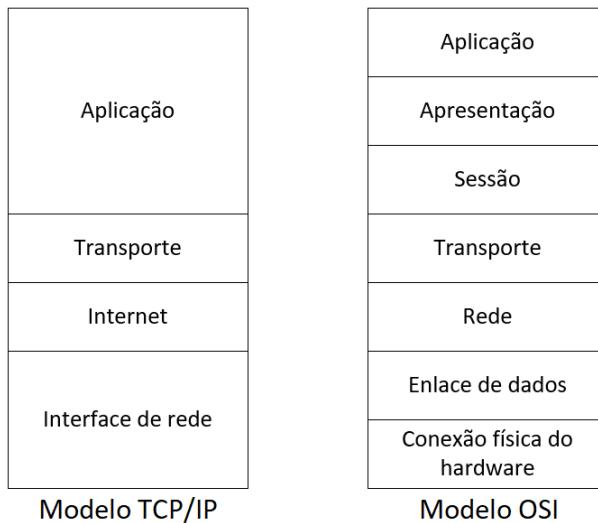
2 Fundamentação Teórica

Este capítulo apresenta os fundamentos teóricos necessários para o desenvolvimento do projeto.

2.1 Pilha de protocolos TCP/IP

A pilha de protocolos TCP/IP é um conjunto de protocolos utilizados na comunicação de computadores pela internet, (Comer, D., 2006). O protocolo TCP, em inglês *Transmission Control Protocol*, e o protocolo IP, *Internet Protocol*, integram o TCP/IP. Esta pilha de protocolos é formada por quatro camadas, são elas: Aplicação; Transporte; Internet; Interface de Rede. Outro modelo equivalente ao TCP/IP é o Modelo OSI, que possui 7 camadas. A Figura 3 mostra as camadas que compõem os modelos citados acima.

Figura 3 – Modelos OSI e TCP/IP



Fonte: Autor.

A camada de Aplicação é o nível mais alto da pilha, ela é responsável por fazer a comunicação entre os programas invocados pelos usuários e os protocolos de transporte. Os protocolos HTTP e MQTT estão nessa camada. Uma aplicação interage com algum protocolo da camada de transporte para receber ou enviar dados.

A camada de transporte tem a função de realizar a comunicação de um programa para outro, essa comunicação também é conhecida como comunicação de ponta a ponta. Os protocolos desta camada dever certificar que os dados cheguem sem erros e na sequência ao receptor. Para isso, os protocolos desta camada devem instruir o receptor a enviar de volta confirmações e o transmissor a reenviar pacotes perdidos. O protocolo TCP é o mais utilizado nesta camada, ele pode dividir o fluxo de dados em partes menores, chamadas de pacotes, e reagrupá-los. A cada pacote é associado um cabeçalho, que contém a porta

de origem e a de destino, número de sequência, *checksum*, entre outras informações. As portas de origem e destino contêm os números de porta TCP, cuja função é identificar a aplicação. Por exemplo, o HTTP utiliza a porta 80 e o MQTT utiliza a porta 1883. O número de sequência identifica o pacote no fluxo de dados. Como já foi dito, o TCP pode tanto dividir o dado a ser transmitido em pacotes como juntar esses pacotes na ordem correta, o que o TCP também faz é checar a integridade dos pacotes, tornando a conexão mais robusta e mais imune a erros.

O protocolo IP está localizado na camada da *internet*. Este protocolo recebe os pacotes enviados pela camada de transporte e os adiciona um cabeçalho, que contém o identificador único de endereço virtual do computador conectado à rede, este identificador é chamado de endereço IP. O protocolo IP tem três funções principais: a de especificar o formato exato de todos os dados; realizar o roteamento, que é determinar o caminho por qual os pacotes vão percorrer até chegar ao seu destino final; definir como os *Hosts* e os roteadores dever processar os pacotes. O roteamento é feito por um dispositivo chamado roteador, cuja função é ligar a rede local a *internet*. Os roteadores possuem endereços de outros roteadores, então quando é necessário transmitir um dado, o roteador verifica se o destinatário está em sua rede, se não, ele envia o dado para outro roteador, isso se repete até que pacote chegue ao destinatário.

Finalmente, a camada de rede estabelece a comunicação entre os pacotes da camada Internet e a rede física. O que define esta camada é o tipo de rede física que está em uso, normalmente é utilizada a rede *Ethernet*. Esta rede corresponde às camada 1 e 2 do modelo OSI, que são as camadas que trabalham os aspectos físicos de uma comunicação. A *Ethernet* é composta por três camadas, *Logic Link Control* (LLC), *Media Access Control* (MAC) e a Física. A camada LLC tem a função de direcionar os pacotes recebidos da rede para os protocolos da camada Internet. A camada MAC monta o quadro que será transmitido pela rede, este quadro contém os endereços MAC da fonte e do destino. Tal endereço MAC está associado a placa de rede de cada computador. A última camada, a Física é responsável por converter o quadro gerado na camada MAC em sinais elétricos.

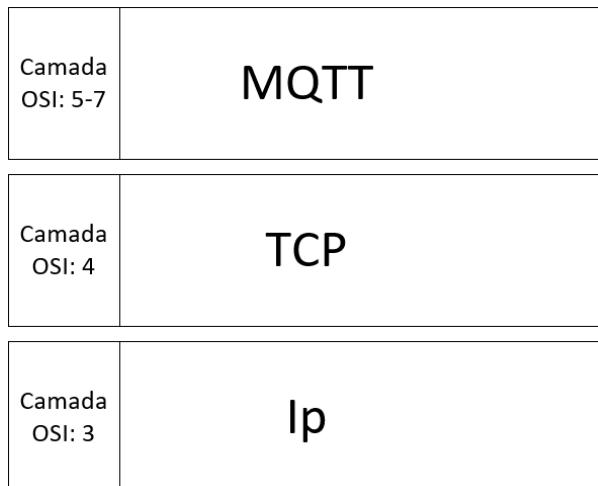
O Wi-Fi está incluído na última camada do modelo TCP/IP, ou nas camadas 1 e 2 do modelo OSI. O Wi-Fi surgiu para melhorar a comunicação sem fio, aumentando a taxa de transferência de dados. Wi-Fi é uma tecnologia de WLAN (Wireless Local Area Network) que segue o padrão IEEE 802.11. Esse padrão estabelece normas para criação de redes sem fio. A comunicação é feita por radiofrequência com o alcance de algumas centenas de metros. O padrão IEEE 802.11 foi apenas o início da redes sem fio, sua taxa de dados era muito baixa, próximo a 2Mbps. Atualmente o padrão 802.11g é o mais utilizado, ele trabalha na faixa de frequência de 2.4GHz e possui um taxa de transferência de dados de até 54 Mbps. Outros padrões como o 802.11n e o 802.11ac têm uma taxa de transferência maior que a do 802.11g, porém o 802.11n tem um custo mais elevado e o 802.11ac ainda é muito atual e possui poucos dispositivos compatíveis no mercado.

2.2 MQTT

Para integrar vários dispositivos em uma rede é necessário que haja uma padronização de comunicação, que é feita por meio de um protocolo de comunicação de nível de aplicação. Neste trabalho será utilizado o protocolo MQTT , (*Message Queue Telemetry Transport*), que foi criado Dr. Andy Stanford-Clark da IBM e Arlen Nipper da Arcom no ano de 1999. Este protocolo possui uma licença livre de *royalties*, é leve e requer pouca largura de banda, permite vários clientes conectados simultaneamente a um único servidor e garante que a mensagem seja enviada, tais características tornam o protocolo ideal para aplicações de IoT. O MQTT foi criado para ser um protocolo de mensagem usado sobre o protocolo TCP/IP e baseia-se em um esquema de *subscriber/publisher*, Michael Yuan (2017).

Nos tópicos anteriores foi mostrado que o protocolo MQTT é baseado no topo dos conjuntos de protocolos TCP/IP, sendo classificado como protocolo de aplicação no modelo TCP/IP, Figura 4.

Figura 4 – Camadas de protocolos



Fonte: Autor.

Como é possível observar em MQTT (2017) a arquitetura do protocolo envolve três componentes principais, o *Publisher* e o *Subscriber*, que também são denominados de *Clients*, o último componente é conhecido como *Broker*. O *Publisher* é o dispositivo que conecta-se ao servidor para enviar informações. O *Subscriber* é o dispositivo que se conecta e escolhe as informações a receber. Já o *Broker* é o servidor e faz a intermediação entre o *Publisher* e *Subscriber*, ele recebe e organiza as mensagens do *Publisher* e envia para o *Subscriber*. Uma característica importante do MQTT é que um *Client* pode ser *Publisher* e *Subscriber* ao mesmo tempo.

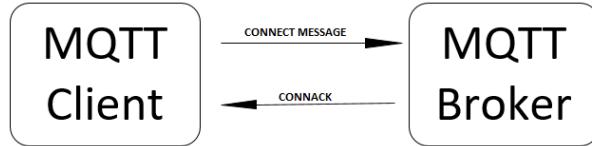
As mensagens são organizadas em tópicos, desta forma o *Publisher* deve informar o tópico da mensagem que ele está enviando. Da mesma maneira, o *Subscriber* se inscreve em um ou mais tópicos.

O MQTT faz um desacoplamento espacial entre o *Publisher* e o *Subscriber*, desta maneira *Pub* e *Sub* não têm conhecimento um do outro, só é preciso o IP e a porta do *Broker* para enviar/receber mensagens. Também há uma separação no tempo, ou seja, *Pub* e *Sub* não precisam estar *online* ao mesmo tempo. Por fim, há um desacoplamento de sincronismo, com isso a operação de um componente não interfere no outro, ou seja, *Pub* e *Sub* funcionam normalmente durante uma publicação ou recebimento de mensagem.

Este protocolo difere-se de um simples *Message Queue* das seguintes maneiras: em um *Message Queue* a mensagem fica na fila até ela ser consumida e é impossível que a mensagem não seja consumida, já no MQTT se nenhum dispositivo estiver inscrito do tópico da mensagem ela não é lida. Em um MQ simples, a mensagem é processada por apenas um cliente, já no MQTT vários clientes podem se inscrever no mesmo tópico e receber a mesma mensagem.

Como já foi dito anteriormente, a conexão é feita somente entre *Broker* e *Client*, não é possível dois *Clients* se conectarem diretamente. Sabendo disso, para iniciar a comunicação o *Client* deve mandar uma mensagem de conexão para o *Broker*, então o *Broker* deve responder um CONNACK, Figura 5.

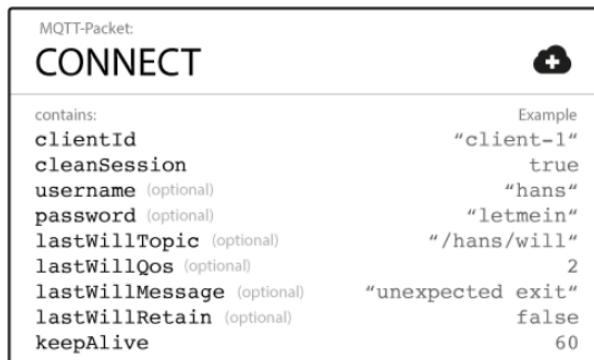
Figura 5 – Inicialização da conexão entre *Client* e *Broker*



Fonte: Autor.

O *frame* da mensagem de conexão pode ser visto na Figura 6.

Figura 6 – Mensagem de conexão



Fonte: HiveMQTT.

O *clientId* é o identificador do cliente, que deve ser único para cada *Broker*.

CleanSession é um *flag* que indica ao *Broker* se o *Client* deseja estabelecer uma sessão persistente ou não. Quando uma sessão persistente é utilizada, a *flag* *cleanSession* é falsa, o *Broker* irá salvar todos os tópicos em que o *Client* estava inscrito e também

serão salvas as mensagens, de QoS 1 ou 2, perdidas pelo *Client*. Se a *flag* for configurada para verdadeira, o *Broker* não irá salvar nenhuma informação para o *Client* e todas as informações de sessões persistentes anteriores do *Client* serão apagadas.

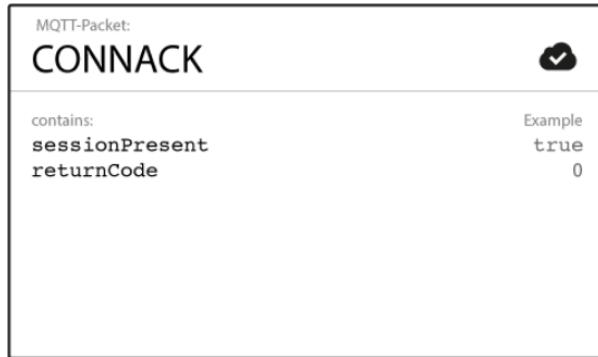
Os campos de *username* e *password* poder ser utilizados para a autentificação do *Client*. Porém o *password* é enviado sem encriptação. Dependendo do *Broker* utilizado é possível utilizar algum tipo de certificado de segurança.

Os campos de *lastWill* informam o *Broker* que ele deve notificar os outros *Clients* quando o *Client* em questão for desconectado inesperadamente.

O *keepAlive* é um intervalo de tempo em que o *Broker* deverá manter a conexão aberta caso o *Client* não estiver ativo.

Essas são as informações que devem estar contidas na mensagem de conexão enviada para o *Broker*. Após o *Broker* receber esta mensagem ele deve responder uma mensagem de CONNACK, cujo *frame* pode ser observado na Figura 7.

Figura 7 – CONNACK message



Fonte: HiveMQTT.

Essa mensagem contém apenas dois dados; o *sessionPresent* que indica se o *Client* possui uma *persistent session* anterior; e o *returnCode* que é a *flag* de *connect acknowledge*. A função desta *flag* é sinalizar ao *Client* se a tentativa de conexão foi bem sucedida ou não, identificando o problema encontrado.

Após obter uma conexão entre *Client* e *Broker*, o *Client* pode assumir o papel de *Publisher* e enviar dados, para isso é necessário que ele envie uma *publish message*, o quadro desta mensagem pode ser visto na Figura 8.

Figura 8 – Publish message



Fonte: HiveMQTT.

PacketId é o identificador único entre o *Client* e o *Broker*. Sua função é identificar a mensagem, porém esse Id só é útil quando a qualidade de serviço for maior que zero.

TopicName é o nome dado ao tópico em que o *Publisher* deseja enviar a mensagem. Como foi dito anteriormente, o *Broker* organiza as mensagens em tópicos e os *Subscribers* selecionam de qual tópico desejam receber mensagens.

QoS é a qualidade de serviço da mensagem. Esse assunto é explicado mais adiante.

RetainFlag é a *flag* que determina se o *Broker* deve salvar a última mensagem válida enviada pelo *Publisher*. Com esta *flag* ativada, todos os novos *Subscribers* inscritos neste tópico receberão a mensagem que foi salva pelo *Broker*.

Payload é o dado a ser enviado propriamente dito. É possível enviar dados criptografados, imagens, textos em diversas formatações, desde arquivos binários simples até JSON e XML. O protocolo MQTT determina que uma mensagem deve ter no máximo 250MB.

DupFlag indica se a mensagem que está sendo enviada é na verdade uma mensagem reenviada, pois a mensagem original não tenha sido reconhecida pelo destinatário. Essa *flag* só tem significado quando a qualidade de serviço é maior que zero.

Essas informações são o conteúdo da mensagem de publicação. Quando o *Broker* recebe essa *Publish message*, para $QoS \geq 1$, ele deve confirmar que recebeu enviando uma *PUBACK message* para o transmissor. Essa *PUBACK message* contém o mesmo *packetId* da *Publish message*. Então a mensagem é processada, no caso do *Broker* ser o receptor, o processo consiste em determinar os clientes que assinam o tópico e enviar a mensagem. A única preocupação do *Publisher* é enviar a mensagem para o *Broker*, uma vez que o *Broker* recebe a mensagem o trabalho do *Publisher* acaba. Deste ponto em diante é responsabilidade do *Broker* enviar a mensagem para os *Subscribers*, o *Publisher* não recebe *feedback* algum se sua mensagem foi entregue a um *Subscriber*.

Depois que mensagem de publicação é enviada ao *Broker* é possível que *Subscribers* a recebam, mas antes é necessário que o *Subscriber* envie uma mensagem de inscrição, uma *Subscriber message*. O quadro da mensagem pode ser observado na Figura 9.

Figura 9 – Subscriber message



Fonte: HiveMQTT.

Esta mensagem contém o *packetId*, que é um identificador único entre o *Broker* e o *Subscriber*. A mensagem carrega uma lista de inscrições, esta lista é formada por um par de informações, o QoS e o tópico. O numero de pares contidos na mensagem é arbitrário. Após o *Broker* receber a *Subscriber message* ele deve confirmar, para isso ele envia uma *SUBACK message* para o *Client* que lhe enviou a *Subscriber message*. O *frame* desta mensagem está na Figura 10.

Figura 10 – SUBACK message



Fonte: HiveMQTT.

O *packetId* aparece novamente, sua função é identificar a mensagem. Nesta mensagem de *SUBACK* o *packetId* é o mesmo utilizado na *Subscriber message*. O *Broker* deve enviar um *returnCode* para cada par de QoS e tópico enviado pelo *Subscriber*. Esse *returnCode* tem a função de notificar o *Subscriber* em quais tópicos ele foi inscrito com sucesso. Caso não foi possível fazer a inscrição, seja porque o tópico não foi criado adequadamente ou seja porque o *Subscriber* não tem permissão para receber as mensagens deste tópico, o *Broker* responde com um *returnCode* de falha. Após a inscrição ser efetuada com sucesso, o *Broker* começa a encaminhar as mensagens para o *Subscriber*. A mensagem encaminhada tem o mesmo formato da *Publish message*.

Do mesmo modo que um *Subscriber* pode se inscrever em um tópico, ele também pode se desinscrever. O processo de desinscrição é bem parecido com o de inscrição,

basta o *Client* enviar uma *Unsubscribe message* contendo os tópicos que deseja cancelar a inscrição. Então o *Broker* responde com uma mensagem de *UNSUBACK*.

A qualidade de serviço, em inglês *Quality of Service* (QoS), determina a garantia da mensagem ser entregue com sucesso. O MQTT utiliza três níveis de QoS: No máximo uma vez(0); Pelo menos um vez(1); Exatamente uma vez(2). O QoS deve ser escolhido de acordo com a aplicação utilizada. Deve-se utilizar QoS 0 quando há uma conexão estável entre o dispositivo que envia a mensagem e o que recebe, também recomenda-se utilizar QoS 0 quando perder um pacote de dados de vez em quando não for um problema grande. O QoS 1 deve ser usado quando todas as mensagens devem ser entregues e quando não há problema se uma mensagem for duplicada. O QoS 2 deve ser utilizado quando a condição de não receber mensagens duplicadas for exigida. A QoS é um componente importante no protocolo MQTT, pois a qualidade de serviço garante que a mensagem será entregue mesmo em condições de conexão instável. Para descrever como a qualidade de serviço é implementada no protocolo MQTT será discutido cada nível separadamente.

Nas descrições a seguir *Client* e *Broker* podem assumir o papel de transmissor ou receptor. Por exemplo, o *Publisher* (transmissor) envia uma mensagem para o *Broker* (receptor), ou o *Broker* (transmissor) envia uma mensagem para o *Subscriber* (receptor). Também é importante sinalizar que quando o servidor está entregando mensagens para mais de um cliente, cada cliente é tratado de forma independente. O nível de QoS usado para enviar uma mensagem para um cliente pode ser diferente do nível da mensagem recebida pelo servidor.

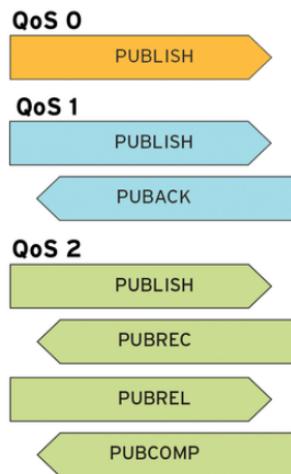
Em QoS 0 o transmissor envia a mensagem uma única vez. Não é necessário que o receptor reconheça que a mensagem foi entregue. Também não é preciso que o transmissor armazene a mensagem nem que reenvie. Neste nível o transmissor apenas envia a mensagem e a exclui, este modo também é chamado de "*fire and forget*".

Quando é utilizado QoS 1, pelo menos uma vez a mensagem é entregue, ou seja, há garantia de que a mensagem será entregue, mas existe a possibilidade da mesma mensagem ser entregue mais de uma vez, isso pode ser um problema em algumas aplicações. Neste nível de QoS o transmissor armazena a mensagem até que ele receba um *PUBACK* do receptor, este *PUBACK* deve conter o mesmo *packetId* da *Publish message*. Se o *PUBACK* não for recebido em um tempo determinado o transmissor envia novamente a *Publish message* com a *dupFlag* ativada. A mensagem é deletada após o transmissor receber a confirmação do receptor. Depois do receptor enviar a mensagem de *PUBACK* ele deve tratar qualquer *Publish message* que contenha o mesmo *packetId* como sendo uma nova publicação, mesmo que a *DUP flag* esteja ativada, desta forma mensagens duplicadas são geradas.

O nível mais alto de qualidade de serviço, QoS 2, é o mais seguro e também o mais lento. Neste nível é garantido que o receptor receberá apenas uma vez a mensagem. Essa garantia é feita através de dois pares de mensagens trocadas entre o transmissor e o receptor. O

primeiro par de mensagem contém a *Publish message* e a *PUBREC message*. A *PUBREC message* é uma mensagem de confirmação que indica ao transmissor que o receptor já armazenou a *Publish message*. Se o transmissor não receber a *PUBREC message* ele reenvia a *Publish message* com a *dupFlag* ativada. Então o transmissor deleta a *Publish message* após receber a confirmação da *PUBREC*. O segundo par de mensagens contém a *PUBREL message* e a *PUBCOMP message*. Esta primeira mensagem é enviada pelo transmissor e diz ao receptor que ele pode processar a mensagem, a segunda mensagem é apenas uma confirmação. Se o transmissor não receber esta confirmação ele reenvia a *PUBREL message* até que a confirmação seja feita. Uma vez enviada a *PUBREL* o transmissor não deve reenviar a *Publish message*. Até o receptor receber a *PUBREL message*, ele deve reconhecer as *Publish message* com o mesmo *packetId* enviando um *PUBREC*. A Figura 11 mostra de forma resumida a comunicação entre o transmissor e o receptor de acordo com o nível de QoS, .

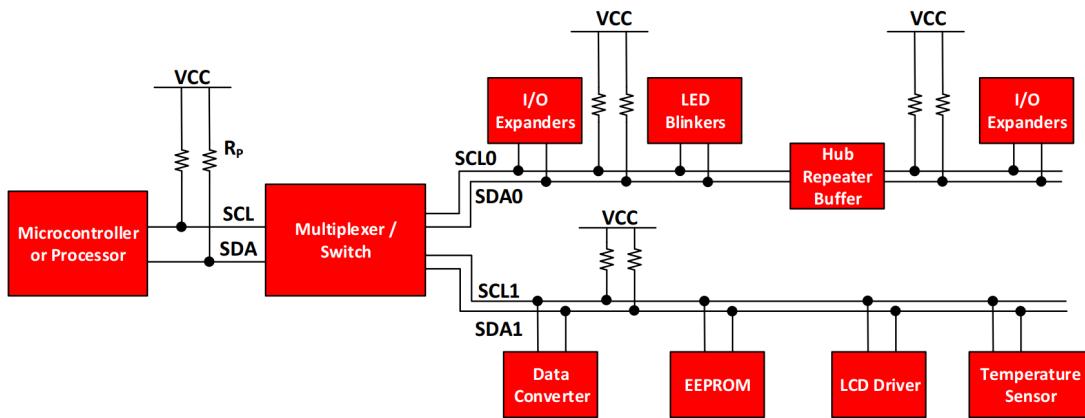
Figura 11 – Níveis de qualidade de serviço no MQTT



Fonte: (MENS, 2015)

2.3 I²C

A utilização do barramento I²C é muito popular e poderosa para fazer comunicação entre um dispositivo mestre e vários outros dispositivos escravos, (Valdez, J. and Beck J., 2015). Diversos sensores e conversores A/D utilizam a comunicação *I²C*. Esse protocolo permite que um dispositivo, denominado mestre, se comunique com outros dispositivos, chamados de escravos, através de uma barramento que utiliza 2 fios (*clock* e informação) na configuração de *wired or*. Com a configuração de circuito com dois resistores *pull-up* o barramento sempre está em alto quando não há nenhuma comunicação, como visto na Figura 12.

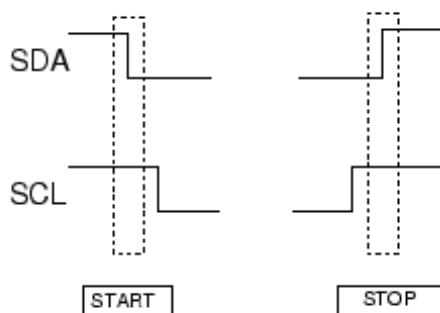
Figura 12 – Exemplo de barramento I²C

Fonte: (Valdez, J. and Beck J., 2015).

Quando um dispositivo deseja enviar "falar zero" no barramento, ele drena corrente da linha, no caso de enviar "um" o dispositivo entra em alta impedância, fazendo com que o resistor de *pullup* force a linha para 1. Desta forma, quando dois dispositivos tentam a comunicação simultânea não ocorre um curto-circuito. A linha SCL é o *clock* que é provido apenas pelo mestre, e a linha SDA é por onde os bits de comunicação são enviados.

Como dito anteriormente, a comunicação através de um barramento I^2C se define por dois dispositivos, um atuando como mestre e outro como escravo. Ambos conseguem ler e escrever dados no barramento, no entanto, para o escravo fazer essas ações, é necessário a permissão do mestre.

No início de uma comunicação o mestre deve enviar a condição de *Start*, que faz com que o barramento SDA, anteriormente em alta, seja levado para zero, resultando em uma borda de descida enquanto o barramento SCL está em alto. Quando o mestre deseja finalizar a comunicação, a condição de *Stop* é enviada. Esta condição ocorre quando o barramento de clock está em alta e há uma borda de subida no barramento de dados, a Figura 13 resume o texto acima.

Figura 13 – Condição de *start* e *stop* no barramento I²C

Fonte: Autor.

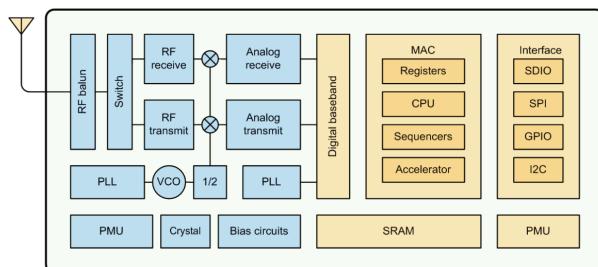
Todo *Byte* transmitido em I^2C , seja ele dado ou endereço, é reconhecido com um bit

chamado ACK, do inglês *acknowledge*. Quando o mestre finaliza o envio de um byte para o escravo, ele para de enviar sinal no barramento de dados e aguarda o envio de um ACK pelo escravo. O escravo então baixa o SDA para que o mestre envie um pulso de clock para sincronizar o bit ACK. Da mesma forma, quando o mestre finaliza a leitura de um *Byte*, ele zera o SDA para que o escravo saiba, através deste sinal de ACK, que a operação foi finalizada. Vale também ressaltar que um NACK funciona de maneira oposta, durante o ciclo de comunicação. Ou seja, durante o ciclo o barramento SDA é mantido em nível lógico alto.

2.4 ESP8266

O ESP8266 é um microcontrolador produzido pela empresa *Espressif Systems*. O diferencial deste microcontrolador é que ele é capaz de fazer comunicação via Wi-Fi e tem um baixo custo, algo em torno de 2 dólares, (Ebay, 2017). Devido a estes fatores ele é amplamente utilizado em projetos de IoT. Além da comunicação por Wi-Fi (seguindo o protocolo IEEE 802.11 bgn), este microcontrolador conta com 16 portas GPIO, comunicação I^2C , UART, SPI, um módulo ADC, um CPU da Tensilica L106 de 32-bit com a arquitetura Xtensa, clock de 80MHz podendo chegar a 160MHz, (Espressif, 2017). O diagrama de blocos do ESP8266 é apresentado na Figura 14.

Figura 14 – Diagrama de blocos do ESP8266EX



Fonte: (Espressif, 2017).

ESP8266 contém todas as ferramentas para se estabelecer uma comunicação Wi-Fi, como; conectores para antenas ou antena integrada, RF balun (dispositivo que casa impedâncias), amplificadores, filtros. O microcontrolador pode rodar aplicações que usam Wi-Fi e também pode servir de adaptador Wi-Fi para outro microcontrolador, utilizando uma comunicação simples como I2C ou UART. A tabela 1 mostra os canais que o transceptor do ESP8266 pode trabalhar, segundo o padrão IEEE 802.11 bgn.

A parte de RF do microcontrolador é formada pelos seguintes principais blocos: receptor, transmissor, gerador de *clock* de alta precisão, relógio de tempo real, reguladores e gerenciador de energia.

Tabela 1 – Canais e suas respectivas frequências

Número do canal	Frequência(MHz)	Númerodo canal	Frequência(MHz)
1	2412	8	2447
2	2417	9	2452
3	2422	10	2457
4	2427	11	2462
5	2432	12	2467
6	2437	13	2472
7	2442	14	2484

O receptor converte o sinal RF para um sinal em quadratura e em banda base, posteriormente este sinal é digitalizado por 2 conversores A/D de alta precisão e alta velocidade. Para contornar os problemas gerados pelas condições adversas do canal de comunicação, são integrados no ESP8266 filtros RF, controle automático de ganho (AGC) e circuitos que cancelam o *offset* do nível DC.

O transmissor converte o sinal em quadratura para um sinal em banda passante de 2,4GHz, e utiliza um amplificador CMOS para ligar a antena. Posteriormente são feitas algumas calibrações para minimizar os problemas gerados pelo *offset* DC, *carrier leakage* e I/Q *phase matching*.

O gerador de *clock* gera sinais em quadratura de 2,4GHz para serem utilizados pelo receptor e pelo transmissor. Todos os componentes utilizados pelo gerador de *clock* estão integrados no microcontrolador, incluindo o indutor, o varicap e o PLL.

O principal motivo da escolha deste microcontrolador foi seu preço, na Tabela 2 é possível verificar algumas especificações e o preço de diversos μC .

Tabela 2 – Comparativo entre microcontroladores

Modelo	RAM	Clock	CPU	Preço
NodeMCU V1 dev.kit.(ESP12-E)	64kB	80-160MHz	ESP8266	US\$3
Arduino Uno + WiFi Shield	2kB	16MHz	ATmega328P	US\$22+81
Arduino Uno + SparkFun WiFi Shield	2kB	16MHz	ATmega328P	US\$22+14
Photon	128kB	120MHz	STM32F205	US\$19

2.5 Lua, eLua e NodeMCU

A linguagem de programação Lua desenvolvida por uma equipe de pesquisadores da PUC-Rio (Pontifícia Universidade Católica do Rio de Janeiro) no ano de 1993, (LUA, 2017). Esta é uma linguagem que permite facilmente estender aplicações escritas em outras linguagens, como C, C++, Java. Também é simples estender programas escritos em outras linguagem com Lua. Esta é uma linguagem rápida, pequena e sua API é simples e bem documentada. Lua é tipada dinamicamente e é executada por meio interpretação

de *bytecodes* por uma máquina virtual. O IDE ESPPlorer foi desenvolvido especialmente para trabalhar com a plataforma ESP8266. Utilizando este IDE com a linguagem Lua, o programador tem muito controle sobre o microcontrolador, sendo possível trabalhar em diversas aplicações.

A partir do projeto da linguagem Lua surgiu o eLua, em inglês *Embedded Lua*, que fornece uma total implementação da linguagem Lua para sistemas embarcados, além de funções específicas para o desenvolvimento de software para sistemas embarcados. O eLua é executado diretamente no microcontrolador, não há necessidade de um sistema operacional.

O *software* NodeMCU é feito em eLua baseado no SDK ESP8266 NONOS da empresa Espressif, (NodeMCU, 2017). A maioria do código é escrita em C, como visto nos códigos fonte das bibliotecas disponíveis na referência citada logo acima, e é utilizado Lua para reunir e organizar as bibliotecas utilizadas neste *software*. NodeMCU possui um modelo de programação muito parecido com Node.js, porém é escrito em Lua. Portanto várias funções possuem parâmetros de funções de *callback*.

2.6 Sensores

De acordo com (NIST, 2009), um sensor é um transdutor capaz de converter parâmetros físicos, biológicos ou químicos em sinal elétrico, por exemplo: temperatura, pressão, fluxo ou vibração. Sensores também são capazes de mensurar parâmetros físicos e retornar dados digitais.

Sensores são dispositivos capazes de gerar respostas mensuráveis para mudanças de uma grandeza física, como por exemplo temperatura, pressão, campo magnético, áudio, etc, (BRAGA; RUIZ; NOGUEIRA, 2003).

Reed switch é um tipo de sensor magnético que se comporta como uma chave com acionamento magnético. Este componente é construído por duas placas ferromagnéticas separadas, ou seja, em condições normais a chave está aberta. As placas são envolvidas por uma cápsula de vidro, que dentro dela há um gás inerte para evitar a oxidação das placas. Quando algum campo magnético, um imã por exemplo, se aproxima do sensor as placas ferromagnéticas se encontraram, fechando a chave.

Sensores de presença têm o funcionamento baseado em emissão e recepção de ondas eletromagnéticas. Quando algum objeto passa pelo campo de ação do sensor, alguma das ondas emitidas serão refletidas de volta ao sensor, com isso é possível determinar a presença de uma pessoa em determinado ambiente.

Sensores de pressão, utilizando *strain gage*, são sensores que têm sua resistência alterada conforme uma deformação é aplicada em sua superfície. Com isso é possível determinar que em determinada área foi aplicada uma força.

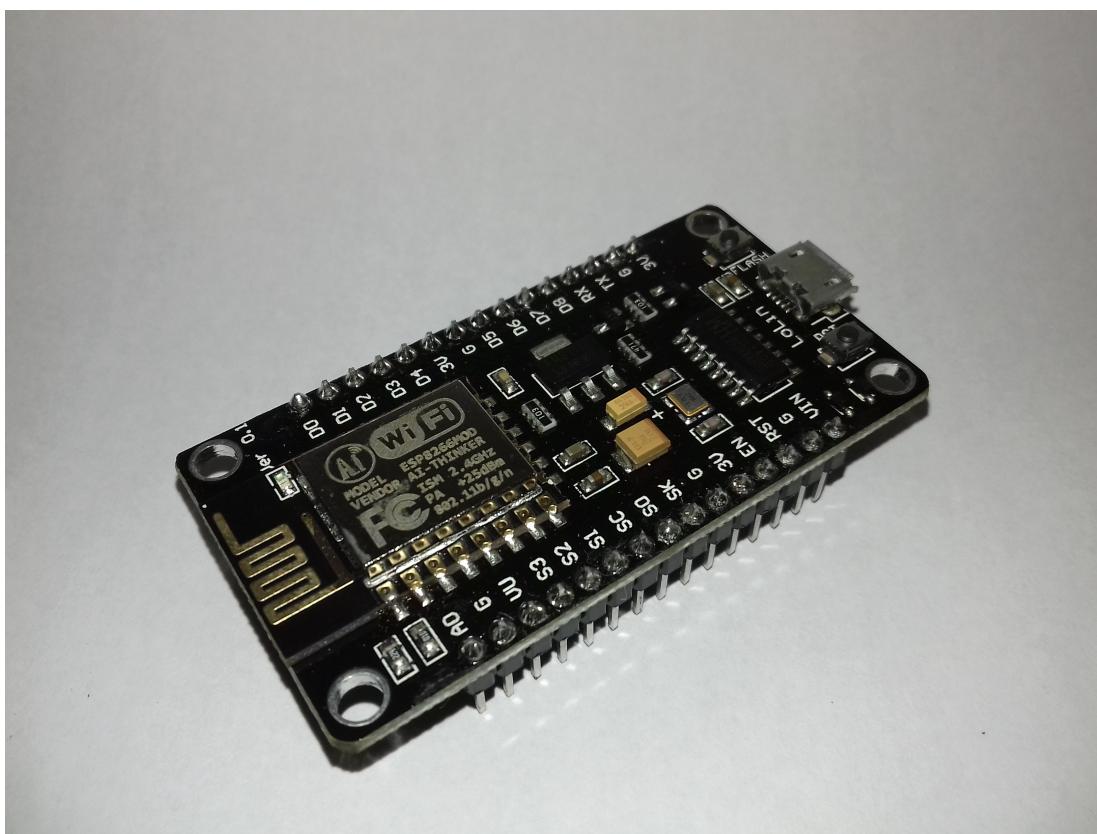
3 Desenvolvimento

Este capítulo apresenta detalhadamente o que será feito para atingir o objetivo do trabalho, também será mostrado os requisitos e ferramentas utilizadas no projeto do sistema de monitoramento residencial.

3.1 ESP8266

Todo o trabalho foi feito tendo a placa de desenvolvimento NodeMCU V1, Figura 15, como ferramenta principal. Esta placa integra o microcontrolador da família ESP8266, o ESP-12E mais precisamente, produzido pela empresa *Espressif Systems*. A placa de desenvolvimento conta com um regulador de tensão e um adaptador serial UART-USB. O μC precisa ser configurado para o modo *flash* antes de carregar um novo *software* para sua memória. Para tal, é necessário que o pino GPIO0 seja forçado para zero antes de reiniciar o dispositivo. Para uma inicialização normal, o pino deve estar em alto ou *floating*. Quando se utiliza a placa de desenvolvimento NodeMCU V1, não é necessário realizar os passos descritos, pois a conexão USB controla o pino GPIO0. Essas características da placa de desenvolvimento facilitam sua utilização.

Figura 15 – Placa de desenvolvimento NodeMCU V1



Fonte: Autor.

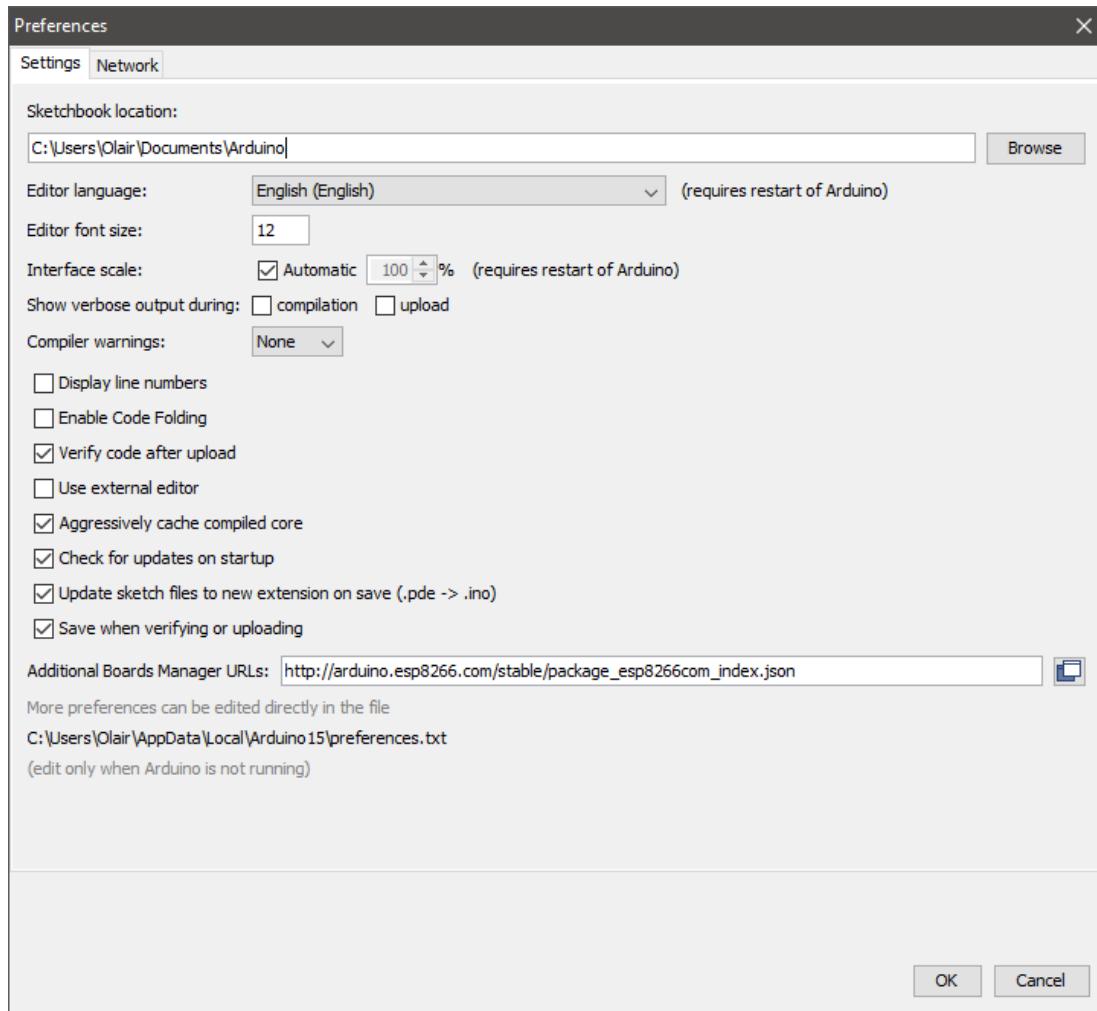
3.2 Testes iniciais

O primeiro passo para programar o NodeMCU V1 utilizando um computador com Windows 10 foi instalar o *driver* CH340G. Este *driver* é o responsável pela comunicação serial USB para UART. Com a comunicação USB-Serial estabelecida foi possível iniciar a programação do microcontrolador utilizados os IDEs citados abaixo.

Os testes iniciais com o NodeMCU V1 foram feitos, simultaneamente, utilizando o Arduino IDE e o ESPPlorer. Na plataforma do Arduino utilizou-se a linguagem de programação C/C++. Já no ESPPlorer foi utilizado a linguagem Lua.

O ambiente de desenvolvimento do Arduino é bem simples e fácil de usar. Para começar a utilizar o Arduino IDE foi necessário instalar a biblioteca do microcontrolador ESP8266. O link da biblioteca é o seguinte: <http://arduino.esp8266.com/stable/package_esp8266com_index.json>. Para instalar basta adicionar este link nas 'Preferências' do IDE, Figura 16.

Figura 16 – Tela de Preferências do Arduino



Fonte: Autor.

O ESPPlorer citado anteriormente é um IDE que usa a linguagem Lua e foi feito para

desenvolvedores que trabalham com o microprocessador ESP8266, o software pode ser encontrado em ESP8266-ru (2014). Para utilizar o ESP8266 neste ambiente de desenvolvimento foi necessário que o *software* NodeMCU estivesse na memória do microprocessador. Tal *software* é baseado em eLua, um software para circuitos embarcados escrito em Lua. A documentação do *software* pode ser encontrada em NodeMCU (2017). Foi utilizado o *software* NodeMCU *flasher* para passar o conteúdo do *software* para a memória do microprocessador. O *software* NodeMCU *flasher* podem ser encontrados no repositório *Github*, (GITHUB, 2017). Em um primeiro momento foi utilizado a versão 0.9.6 – *dev_20150704* devido a facilidade de encontrar o arquivo binário no próprio repositório do *Github*. Com o aumento dos módulos implementados no *software*, um arquivo binário com todos os módulos tornou-se muito grande, desta forma os administradores do NodeMCU decidiram não fornecer mais o arquivo binário. Então cabe ao usuário escolher quais módulos irá utilizar e montar um arquivo binário que contenha esses módulos. Com a evolução deste projeto, a versão 0.9.6-*dev_20150704* não estava mais fornecendo as ferramentas necessárias para cumprir o objetivo do projeto. Decidiu-se utilizar a versão 2.1.0-master_20170824, que é mais recente do *software*. Para gerar o arquivo binário com os módulos necessários foi utilizada a ferramenta disponibilizada por Travis CI (2017).

Ao trocar a versão do *software* NodeMCU deve-se atentar para o *baud rate* utilizado na gravação de dados, pois a versão mais antiga utiliza um *baud rate* de 9600bps, já a versão mais nova tem um *baud rata* de 115200bps. Caso não seja utilizado o *baud rate* correto o IDE apresentará diversos erros ao tentar gravar o código na memória do micro.

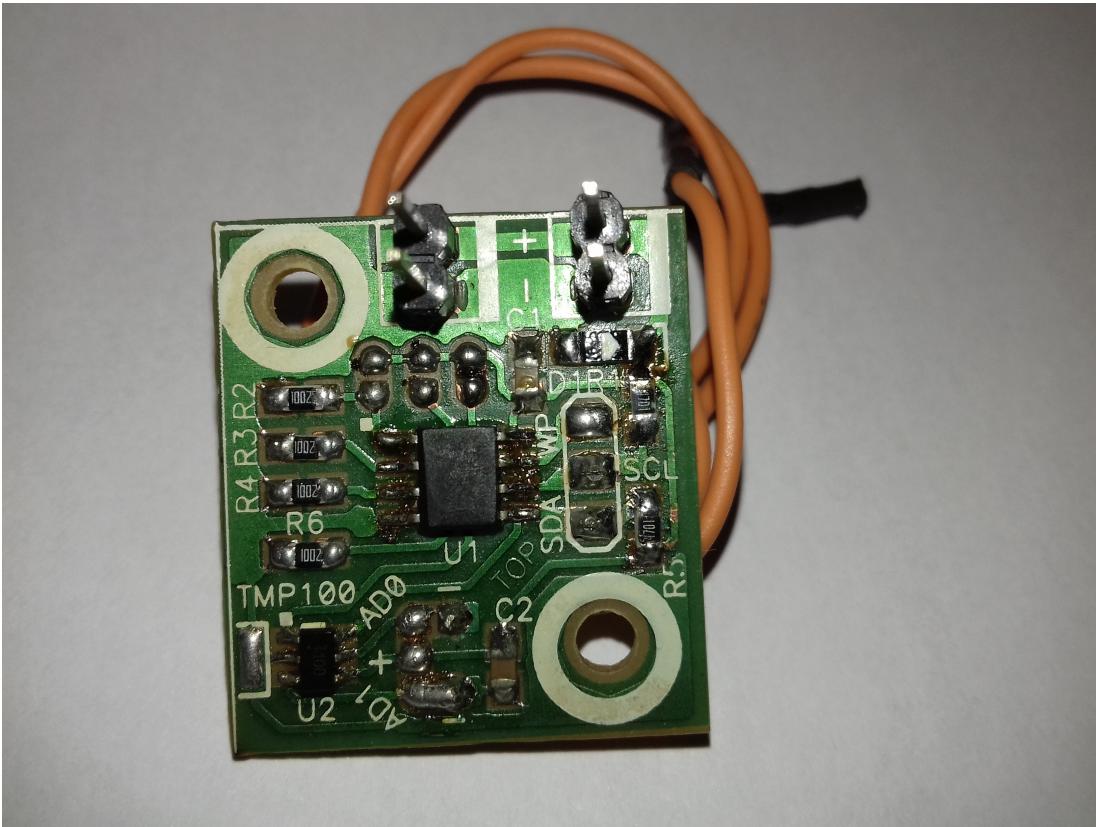
Quando se trata de monitoramento residencial além de detectar eventos é necessário determinar quando esses eventos ocorreram. Para isso foi utilizado o protocolo NTP, *Network Time Protocol*, que é um protocolo que está na camada de aplicação da pilha TCP/IP. O NTP utiliza o UDP como protocolo de transporte.

O *software* NodeMCU possui dois módulos para sincronizar e contar o tempo, o módulo SNTP e o RTC *Time*. O SNTP, *simple network time protocol*, implementa um cliente NTP e faz a sincronia entre o servidor NTP e o relógio do microcontrolador. Ao utilizar as funções do módulo SNTP é possível conectar-se a um servidor NTP e obter o *Unix Timestamp*, que é o valor em segundos contado a partir do dia 1 de janeiro de 1970. Após se conectar com o servidor e obter o *Unix Timestamp* o relógio do microcontrolador é sincronizado com o servidor NTP. Os servidores NTP utilizados neste projeto foram os 0.*br.pool.ntp.org* até o 3.*br.pool.ntp.org*.

Um dos testes iniciais foi adquirir a temperatura ambiente utilizando o sensor TMP100. Este sensor estava soldado em uma PCI, cujo diagrama esquemático se encontra anexado ao final deste documento. A Figura 17 mostra o dispositivo soldado em uma placa de circuito impresso.

A comunicação entre o sensor e o microprocessador é feita através do protocolo I^2C . A configuração do sensor foi feita seguindo as recomendações de Tosin, M.C. (2015). Como

Figura 17 – PCI contendo o TMP100



Fonte: Autor.

o sensor foi soldado em uma placa os pinos de endereçamento também estavam soldados, O AD1 estava em 0 e o AD0 estava em *float*, sendo assim, seu *slave address* é 0b1001001, ou 0x49.

A fim de obter os dados de temperatura do sensor TMP100, este deve ser configurado previamente. Inicialmente o ESP8266, mestre, deve enviar *start* no barramento I^2C seguido de uma palavra de controle para escrita, esta palavra deve conter o endereço do sensor 0x48. O TMP100, escravo, deve responder com um ACK. A seguir é enviado ao *pointer register* o endereço do *config register* (0x01). O sensor deve responder um ACK. O próximo byte enviado será escrito no *config register*, pois o *pointer register* contém seu endereço. O próximo comando configura a resolução do sensor, neste caso foi utilizado a resolução máxima do TMP100, ou seja, 12 bits. Para conseguir tal resolução os bits R1 e R0 do *config register* devem ser (1,1). Portanto o *Byte* enviado será (0x60). O escravo deve responder um ACK. Finalmente, o mestre deve enviar um STOP.

Com esta configuração feita é possível ler os valores de temperatura do sensor. De acordo com Texas Instruments (2015), para uma resolução de 12 bits o intervalo mínimo entre as leituras deve ser de 320ms. Sabendo disso, para iniciar o processo de aquisição da temperatura o mestre deve enviar um *start* seguido por uma palavra de controle de escrita. Então o escravo deve responder com ACK. O endereço do registrador de temperatura deve

ser enviado, portanto o *Byte* a ser enviado deve ser (0x00). Novamente, o escravo deve responder um ACK. Um segundo *start* seguido de uma palavra de controle de leitura deve ser enviada para o barramento. O escravo deve responder um ACK. Esta palavra de controle sinaliza para o sensor que ele deve responder com os *Bytes* de dados referentes à temperatura. Então o dispositivo TMP100 deve enviar 2 *Bytes*, 12 primeiros bits de dados e os 4 últimos são zero. Com estes *Bytes* recebidos é necessário convertê-los em um valor de temperatura graus Celsius. O primeiro *Byte* recebido contém os bits mais significativos e o segundo *Byte* contém os bits menos significativos. Para fazer a conversão deve-se fazer uma divisão por 16 no segundo *Byte*, isso significa fazer um deslocamento para a direita, posteriormente deve-se multiplicar o primeiro *Byte* por 16, deslocá-lo para a esquerda. Após o deslocamento é feita uma concatenação entre os *Bytes* e é feita uma multiplicação por 0.0625. O resultado é a temperatura em graus Celsius.

3.3 Utilização do protocolo MQTT

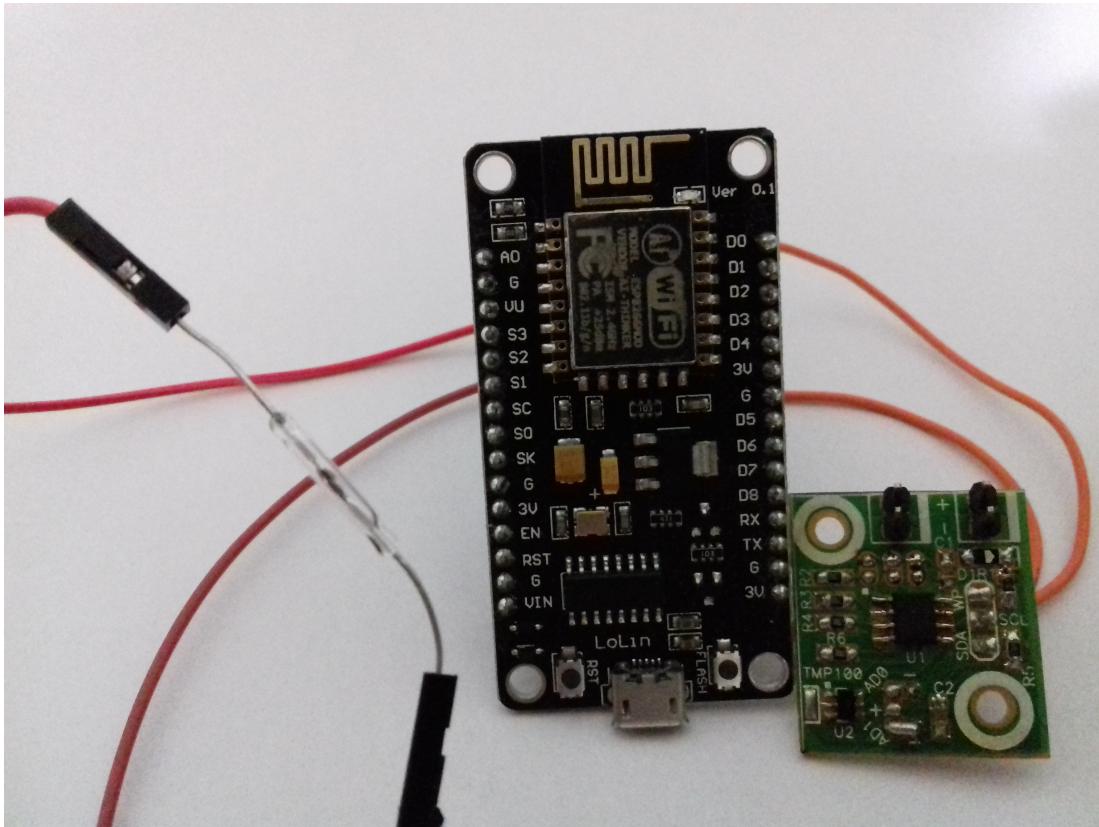
Após os testes de aquisição de temperatura foi utilizado o protocolo MQTT para transmitir dados dos sensores via internet. Nesta etapa de desenvolvimento do projeto foram utilizados apenas dois tipos de sensores, o de temperatura e o magnético, a Figura 18 apresenta os sensores conectados à placa de desenvolvimento. Dispositivos como sensor de nível de água, sensor de presença e vários outros poderiam ser utilizados.

Como dito anteriormente, optou-se por escrever o código utilizando a linguagem Lua e o IDE ESPloerer. O código foi dividido em 5 módulos para melhor organização e depuração. Sempre que o dispositivo é reinicializado o módulo init.lua é o primeiro código a ser executado. Então o módulo realinit.lua é inicializado, sua função é inicializar os outros 3 módulos e executar a função que inicia o módulo steup.lua.

O módulo config.lua contém apenas dados, nenhuma função é executada aqui. Neste módulo pode ser encontrado a tabela de configuração do Wi-Fi, esta tabela contém o SSID e a senha da rede, e duas *flags*; station_cfg.auto, que quando ativada o módulo tenta se reconectar automaticamente quando a conexão é perdida; station_cfg.save quando for verdadeira o SSID e o PW da rede serão armazenados na *flash*. O config.lua também contém as informações do *Broker* a ser utilizado, bem como os pinos e endereços da configuração do barramento I^2C e os pinos do sensor. Foi utilizado o *Broker* público da HiveMQ neste testes. O nome do tópico a ser utilizado na comunicação MQTT também está no config.lua, que é JRNodemcu/.

Em setup.lua o código contém a configuração do TMP100, que foi descrita anteriormente, a configuração do sensor magnético (função RS_config), é selecionado a opção de interrupção e é habilitado um resistor de *pullup*. O setup.lua também faz a configuração e conexão Wi-Fi. Após obter um endereço de IP válido são utilizadas as funções do módulo STNP para sincronizar o relógio do microcontrolador. Por fim, o módulo setup.lua

Figura 18 – Protótipo de testes



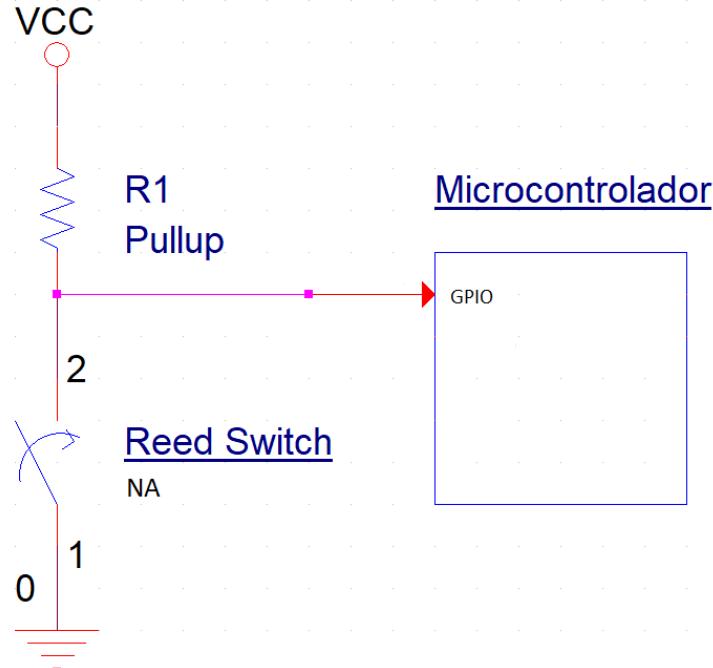
Fonte: Autor.

executa uma função que inicia o módulo application.lua.

O módulo application.lua contém as funções que habilitam a comunicação MQTT, as funções de leitura dos sensores. Este módulo começa com a inicialização de um cliente MQTT, então uma tentativa de conexão com o *Broker* é feita. Se a conexão for efetuada, inicializa-se a interrupção do sensor magnético e dois *timers*, um que controla a leitura de temperatura e outro que envia o ID da placa do ESP8266. O ID é enviado para que a conexão com o *Broker* não seja interrompida devido a um *keepalive timeout*. Caso haja uma falha na comunicação com o *Broker* uma função tenta a reconexão a cada 10 segundos. A cada 1 minuto o *timer* da leitura de temperatura inicia uma função que lê a temperatura do sensor como descrito anteriormente e transmite o valor da temperatura para o *Broker*. A interrupção do pino do *reed switch* é ativada tanto em borda de subida (porta aberta) como em borda de descida (porta fechada). Após uma borda ser detectada um *timer* é iniciado, sua função é providenciar um *debounce*, então após o período de 100ms é feita a leitura do sensor, se o pino estiver em baixo significa que a porta está fechada, se o pino estiver em alto a porta está aberta, figura 19. O código descrito acima, que foi escrito em Lua, pode ser encontrado no Apêndice A.

Para realizar testes o *reed switch* foi posicionado no batente de uma porta e um imã foi posicionado na porta, ao lado do sensor. O diagrama de blocos da figura 20 mostra

Figura 19 – Circuito esquemático da conexão do *reed switch* ao microcontrolador



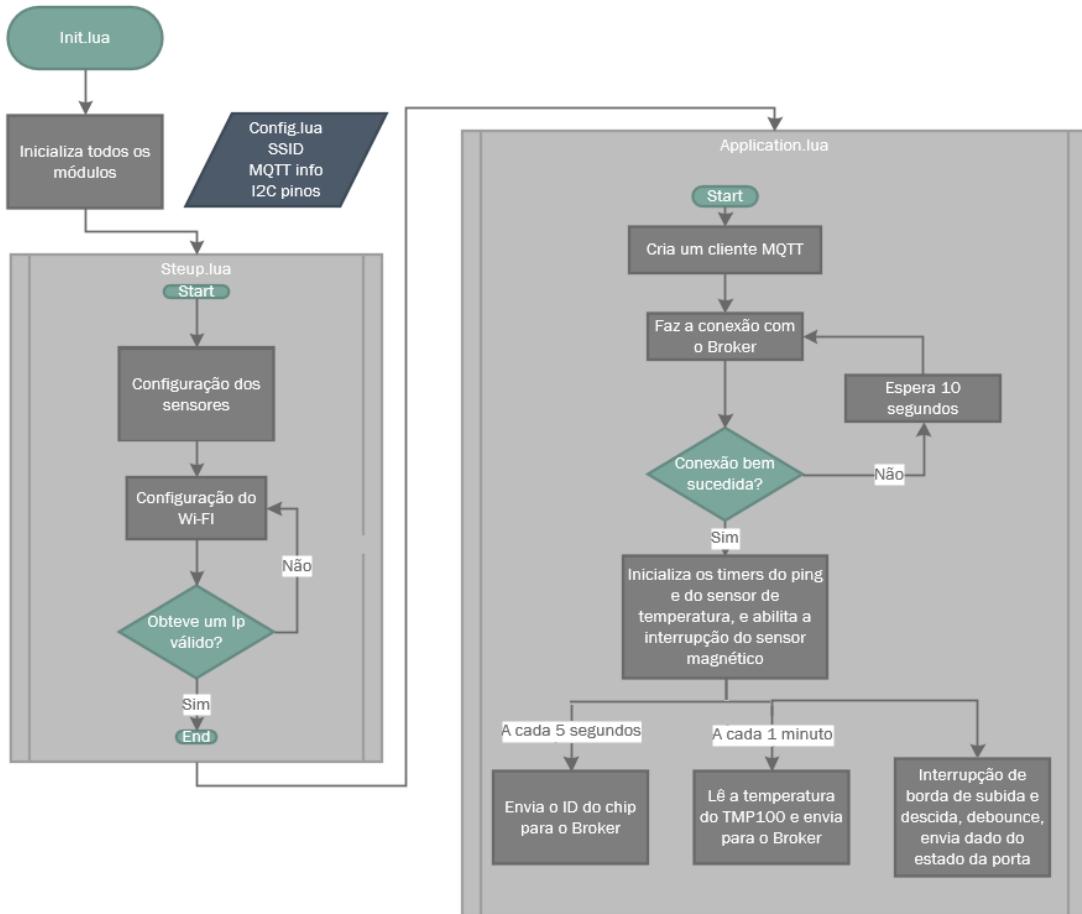
Fonte: Autor.

de forma simplificada o código descrito acima.

Para receber os dados utilizou-se dois métodos. O primeiro foi utilizando o programa MQTT Spy, ele deve funcionar em qualquer sistema operacional que tenha a versão apropriada do Java 8 instalada. Este programa está disponível em *MQTT-Spy (2017)*. O segundo método foi utilizando o aplicativo para celulares *Android IoT MQTT Dashboard* da empresa Nghia TH. Este aplicativo é facilmente encontrado na *Google Play*. O celular utilizado para os teste foi um Samsung J3 com a versão 5.1.1 do Android.

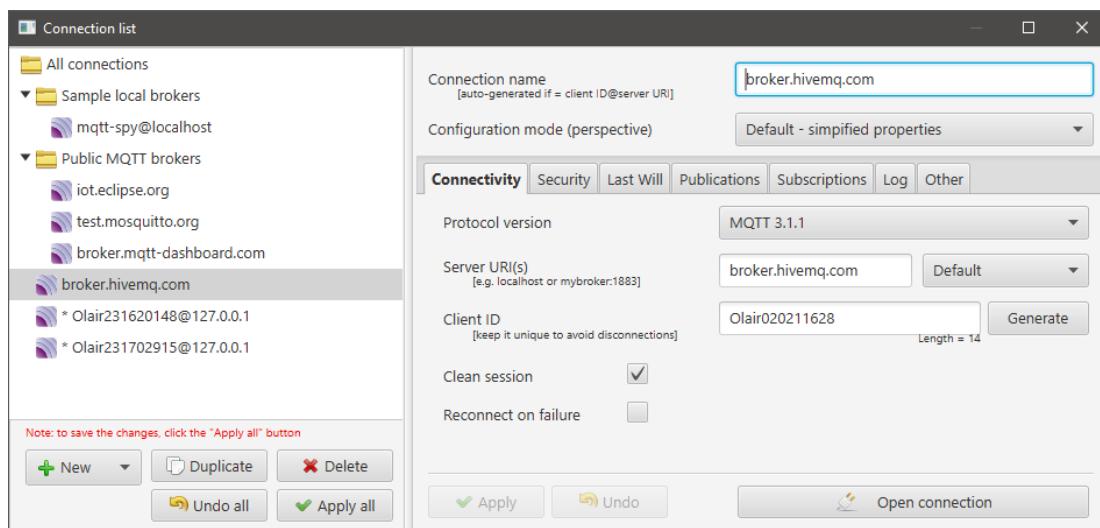
Para utilizar o MQTT Spy foi necessário configurar uma nova conexão de acordo com a figura 21. Efetuado a conexão com sucesso bastou apenas criar uma nova inscrição e inserir o nome do tópico desejado.

Figura 20 – Diagrama de blocos do código que envia dados dos sensores através de uma comunicação MQTT



Fonte: Autor.

Figura 21 – Conexão criada no MQTT Spy



Fonte: Autor.

O aplicativo IoT MQTT Dashboard é muito simples de usar, basta escolher um ID

personalizado, inserir o nome do *server*, e utilizar a porta 1883, que é o número padrão do MQTT.

Posteriormente foi utilizado o *Mosquitto*, um *Broker* que pode ser instalado e utilizado localmente. Os arquivos de instalação podem ser encontrados em Mosquitto (2017). As configurações podem ser alteradas no arquivo mosquito.conf, é possível determinar quantas mensagens de QoS 1 e 2 por cliente o servidor armazena, quanto tempo a sessão permanente dura, a porta do servidor (padrão 1883), o tempo de *keepalive* e muitas outras configurações podem ser customizadas.

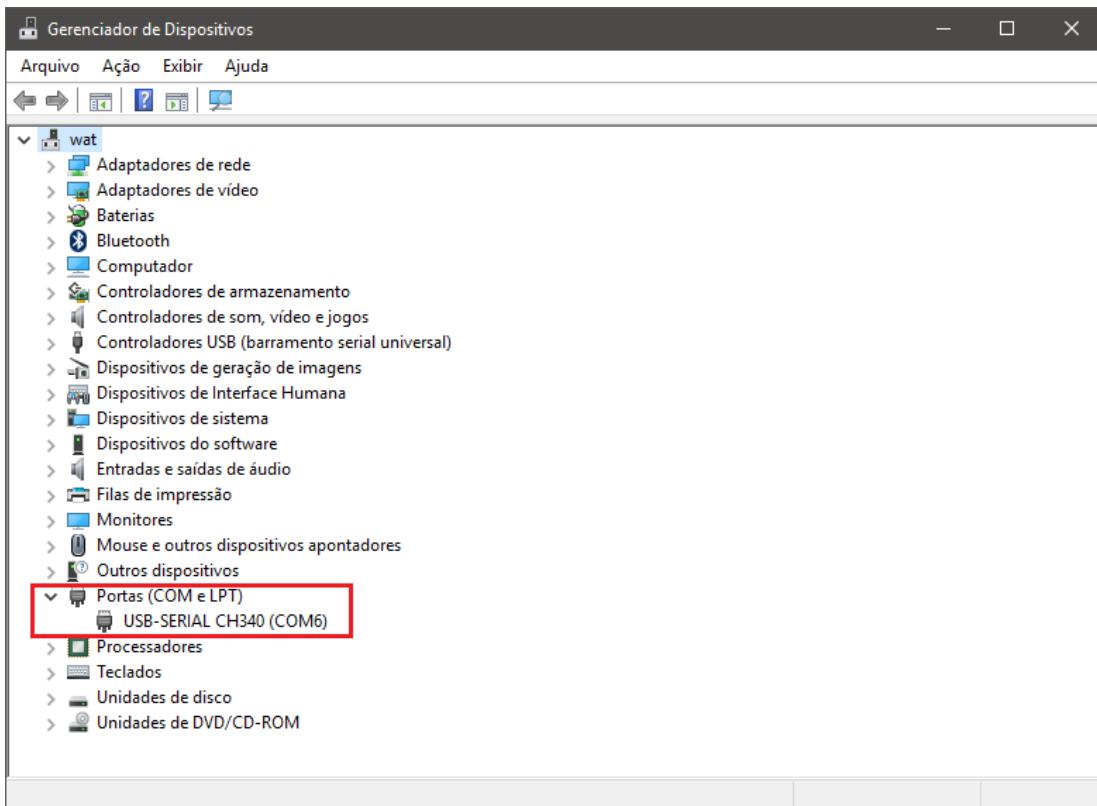
Para o desenvolvimento do aplicativo de celular foi utilizado o projeto fornecido por Eclipse Foundation (2017). Tal projeto implementa um cliente MQTT escrito em Java, que é compatível com dispositivos que rodam a JVM.

4 Resultados e Discussões

Neste capítulo serão apresentados e discutidos os principais resultados obtidos utilizando a placa de desenvolvimento NodeMCU V1, desde os resultados preliminares até os resultados que atingiram os objetivos propostos.

A fim de iniciar os trabalhos com o microcontrolador foi necessário instalar o *driver* CH340G. Após ser instalado com sucesso foi possível verificar que o Windows reconheceu o microprocessador , Figura 22.

Figura 22 – Placa de desenvolvimento reconhecida pelo Windows através da comunicação USB.



Fonte: Autor.

Ao longo do trabalho foi decidido não utilizar o Arduino IDE e focar todo o desenvolvimento do projeto utilizando o ESPplorer, pois os primeiros resultados significativos foram alcançados no ESPplorer. A documentação bem organizada e clara do *software* do NodeMCU, foi um dos principais motivos para seguir o projeto programando em LUA e utilizando o ESPplorer IDE.

Com a comunicação USB estabelecida foram iniciados os trabalhos com a plataforma ESPplorer. Utilizando os métodos descritos acima foi possível fazer leituras de temperatura com o sensor TMP100. A Figura 23 mostra o código utilizado para estabelecer a

comunicação I^2C e obter os valores de temperatura, que podem ser vistos no console do ESPloerer.

Figura 23 – Console do ESPloerer mostrando a temperatura lida do TMP100

A screenshot of the ESPloerer v0.2.0-r5 software interface. The top menu bar includes File, Edit, ESP, View, Links, NodeMCU & MicroPython, AT-based, RN2483. The main window has tabs for Script, Commands, Snippets, Settings. The left pane displays a script named setup.lua with code for initializing the I²C bus and reading from a TMP100 sensor. The right pane shows the serial terminal output, which includes configuration commands like I²C settings and read functions, followed by temperature values (e.g., 25.0875, 38, 29.5425, 29.45). A vertical scroll bar on the right pane indicates many more lines of data.

Fonte: Autor.

A seguir, tem-se a Figura 24 que mostra o funcionamento do NodeMCU V1 juntamente com o sensor TMP100 enviando dados de temperatura para o *Broker* da HiveMQ.

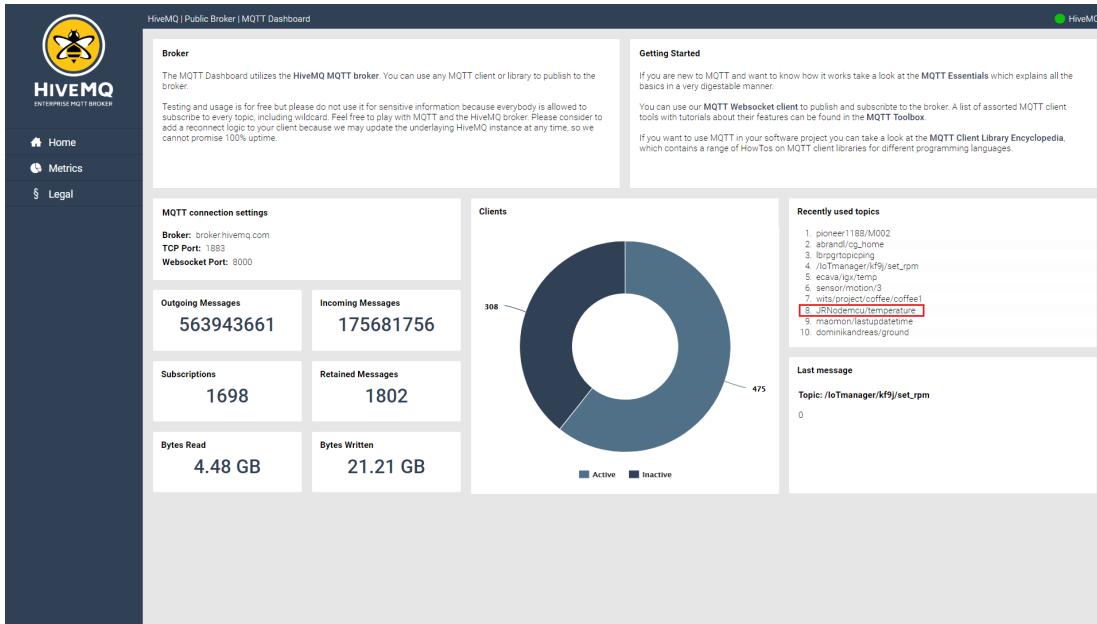
Figura 24 – Console do ESPloerer mostrando a comunicação MQTT sendo estabelecida

A screenshot of the ESPloerer v0.2.0-r5 software interface. The top menu bar includes File, Edit, ESP, View, Links, NodeMCU & MicroPython, AT-based, RN2483. The main window has tabs for Script, Commands, Snippets, Settings. The left pane displays a script named setup.lua with code for establishing an MQTT connection to a broker. The right pane shows the serial terminal output, which includes messages like 'ESP8266 mode is: 1', 'MAC address is: 5e:cf:7f:b0:21:66', 'IP is 192.168.137.65', and 'Client created'. It also shows the process of connecting to the broker and sending data, including file.close(), dofile("realmqtt.lua"), and various MQTT-related messages.

Fonte: Autor.

O painel da HiveMQ, figura 25, que pode ser encontrado em <<http://www.mqtt-dashboard.com/>>, registra o momento em que o tópico utilizado pelo autor apareceu nos tópicos usados recentemente.

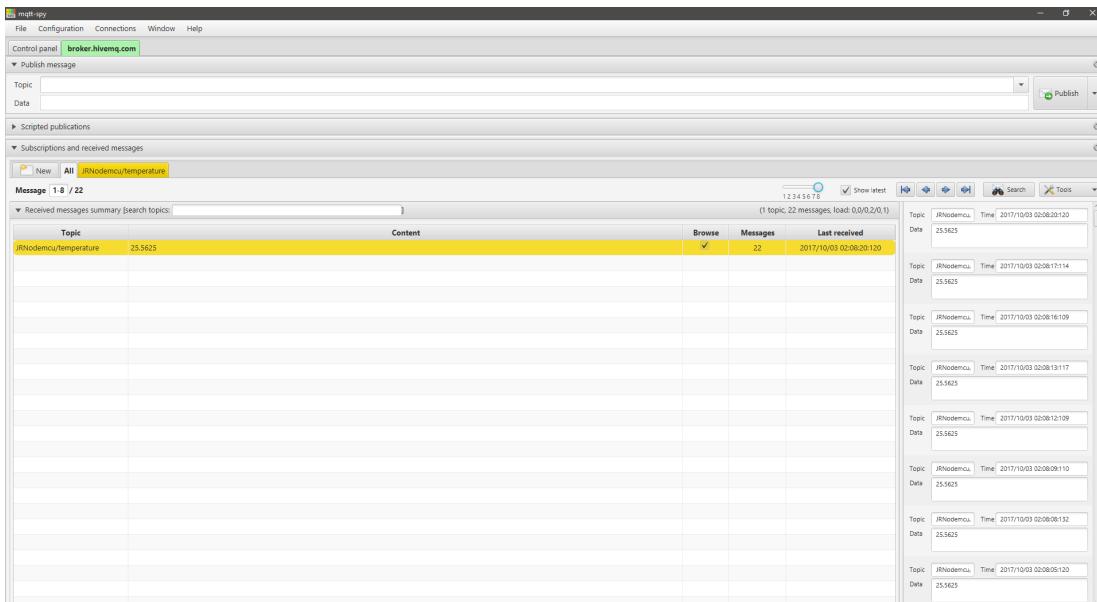
Figura 25 – Dashboard do HiveMQTT mostrando o tópico utilizado pelo autor



Fonte: Autor.

A Figura 26 mostra os dados de temperatura recebidos utilizando o programa para Windows MQTT Spy.

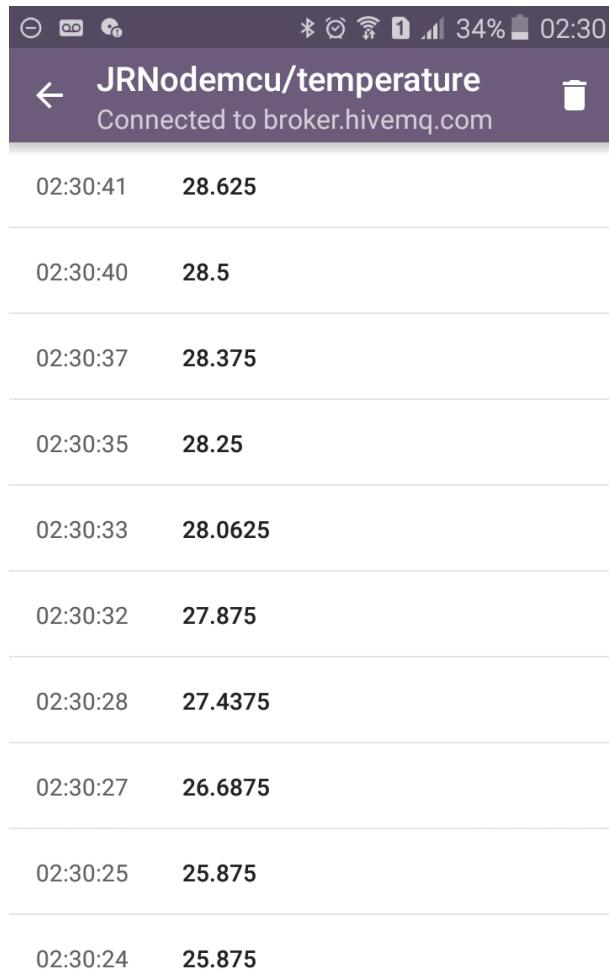
Figura 26 – MQTT Spy mostrando dados recebidos



Fonte: Autor.

A Figura 27 exibe um *log* de dados de temperatura recebidos utilizando um celular como *Subscriber*, o aplicativo utilizado foi o MQTT *Dashboard*.

Figura 27 – Aplicativo IoT MQTT Dashboard mostrando os dados de temperatura



Fonte: Autor.

As leituras de temperatura utilizando o sensor TMP100 foram feitas com sucesso, embora alguns problemas tenham sido encontrados, como o *slave address* do sensor. Observando os pinos AD0 e AD1 nota-se que eles foram soldados na posição *float* e 0, respectivamente, então de acordo com o *datasheet* do dispositivo o *slave address* seria 0b1001001, porém o endereço utilizado foi 0b1001000, que é o endereço de AD0 e AD1 em 0 e 0.

Inicialmente houveram dificuldades em trabalhar com estes módulos, pois em Lua os módulos são armazenados em *cache* na tabela *package.loaded*, para que não haja a necessidade de reexecutar o código do módulo toda vez que ele for utilizado por uma parte do programa. Então uma vez que o módulo foi armazenado na *cache* ele não é modificado, mesmo que o código do módulo seja alterado e enviado para o microcontrolador. Para recarregar de fato o módulo e salvar as alterações feitas, primeiramente é necessário remover o módulo da *cache* com o comando *package.loaded.NomeDoModulo = nil*. Feito isso, pode-se utilizar a função *require* para salvar o módulo na *cache*.

Como já foi dito, o código principal deveria ser dividido em quatro módulos, porém foi

incluído mais um módulo, totalizando 5 módulos, pois quando o init.lua faz a inicialização dos outros módulos e executa a função setup.start() o código apresenta um erro de memória insuficiente, o que não faz sentido, pois o código todo não ocupa mais de 10% da memória do μC . Para contornar este problema foi adicionado o módulo realinit.lua, ele faz o que o init.lua deveria fazer, e o init.lua contém apenas um comando, dofile("realinit.lua"). Ou seja, na inicialização do microcontrolador, primeiramente é executado o módulo init.lua, que por sua vez executa o realinit.lua.

Com a implementação do SNTP foi possível obter o horário em que as leituras dos sensores foram feitas. A figura 28 mostra o aplicativo MQTT Dashboard exibindo os dados de temperatura, data e horário da leitura.

Figura 28 – Aplicativo Iot MQTT Dashboard mostrando os dados de temperatura e quando estes dados foram lidos



Fonte: Autor.

A figura 29 mostra o funcionamento do *reed switch*. Nestes dois testes foi utilizado o Broker da HiveMQ

Figura 29 – Aplicativo IoT MQTT Dashboard mostrando dados do *reed switch*



Fonte: Autor.

Os resultados a seguir foram obtidos utilizando o *Broker Mosquitto*. Após sua instalação foi aberto o *prompt* de comando e inicializado o servidor com o comando "mosquitto". Para a verificação do funcionamento do *Broker* foi utilizado o comando "netstat -a", que mostra todas portas TCP e UDP que o computador está escutando. Como pode-se notar na figura 30, o servidor foi instalado com sucesso, pois a porta 1883 (padrão do MQTT) está em uso.

Figura 30 – *Prompt* de comando do Windows exibindo as conexões ativas

Proto	Endereço local	Endereço externo	Estado
TCP	0.0.0.0:135	wat:0	LISTENING
TCP	0.0.0.0:445	wat:0	LISTENING
TCP	0.0.0.0:554	wat:0	LISTENING
TCP	0.0.0.0:1883	wat:0	LISTENING
TCP	0.0.0.0:2869	wat:0	LISTENING
TCP	0.0.0.0:5357	wat:0	LISTENING
TCP	0.0.0.0:8884	wat:0	LISTENING
TCP	0.0.0.0:10243	wat:0	LISTENING
TCP	0.0.0.0:49664	wat:0	LISTENING
TCP	0.0.0.0:49665	wat:0	LISTENING
TCP	0.0.0.0:49666	wat:0	LISTENING
TCP	0.0.0.0:49667	wat:0	LISTENING
TCP	0.0.0.0:49668	wat:0	LISTENING
TCP	0.0.0.0:49689	wat:0	LISTENING
TCP	127.0.0.1:4380	wat:0	LISTENING
TCP	192.168.0.107:139	wat:0	LISTENING
TCP	192.168.0.107:5040	wat:0	LISTENING
TCP	192.168.0.107:53348	msnbot-65-52-108-195:https	ESTABLISHED
TCP	192.168.0.107:53351	ce-in-f188:5228	ESTABLISHED
TCP	192.168.0.107:53379	edge-star-shv-01-gru2:https	ESTABLISHED
TCP	192.168.0.107:53520	a173-222-198-73:https	CLOSE_WAIT
TCP	192.168.0.107:53521	a173-222-198-73:https	CLOSE_WAIT
TCP	192.168.0.107:53530	192.16.48.200:https	CLOSE_WAIT
TCP	192.168.0.107:53566	ec2-54-175-150-162:https	ESTABLISHED
TCP	192.168.0.107:54782	28:https	TIME_WAIT
TCP	192.168.0.107:54787	28:https	TIME_WAIT

Fonte: Autor.

Com o servidor pronto para ser utilizado, bastou alterar o nome do *host* no módulo config.lua para o IP local do computador rodando o servidor, que no caso era "192.168.0.107". Posteriormente foi criado um cliente utilizando o Mosquitto, para isso foi digitado o seguinte comando no prompt: mosquitto_sub -h 192.168.0.107 -t "#" -v. Onde mosquitto_sub é para criar um *Subscriber*, -h é o endereço do *host*, -t "#" é para que a assinatura seja feita em todos os tópicos do servidos, -v para mostrar o tópico juntamente com a mensagem quando algum dado é recebido.

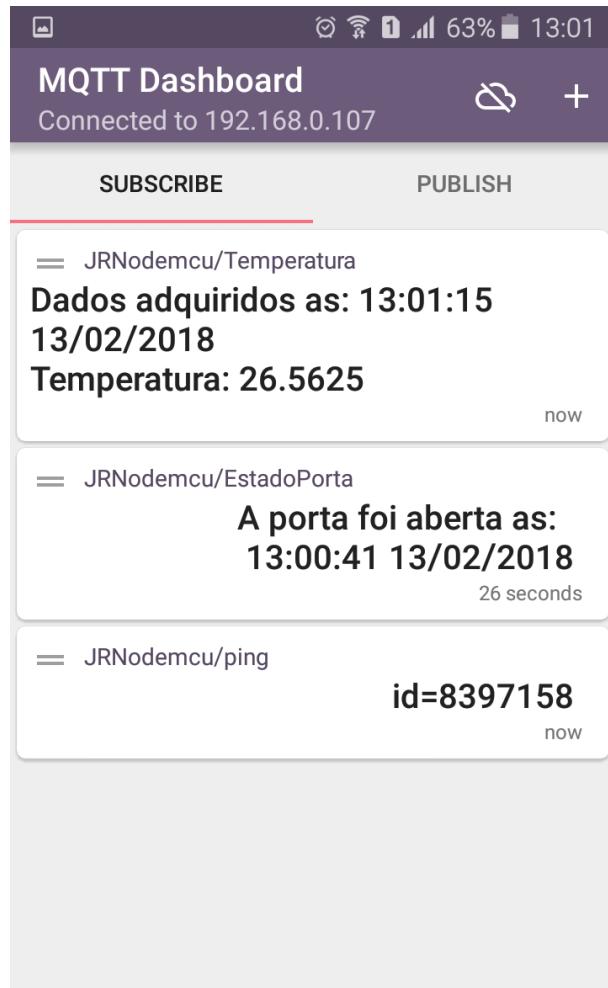
As figuras 31 e 32 mostram o cliente criado utilizando a plataforma Mosquitto e o aplicativo MQTT Dashboard recebendo dados do servidor local.

Figura 31 – *Broker* local sendo utilizado

```
JRNodemcu/temperature Data acquired at: 2018/01/14 21:25:05
Temperature: 27.25
JRNodemcu/temperature Data acquired at: 2018/01/14 21:25:07
Temperature: 27.25
JRNodemcu/ping id=8397158
JRNodemcu/temperature Data acquired at: 2018/01/14 21:25:09
Temperature: 27.25
JRNodemcu/temperature Data acquired at: 2018/01/14 21:25:11
Temperature: 27.25
JRNodemcu/temperature Data acquired at: 2018/01/14 21:25:13
Temperature: 27.25
JRNodemcu/DoorStatus The door has CLOSEd at:
2018/01/14 21:25:13
JRNodemcu/ping id=8397158
JRNodemcu/DoorStatus The door has OPENed at:
2018/01/14 21:25:15
JRNodemcu/temperature Data acquired at: 2018/01/14 21:25:15
Temperature: 27.25
JRNodemcu/temperature Data acquired at: 2018/01/14 21:25:17
Temperature: 27.25
JRNodemcu/DoorStatus The door has CLOSEd at:
2018/01/14 21:25:17
JRNodemcu/DoorStatus The door has OPENed at:
2018/01/14 21:25:19
JRNodemcu/ping id=8397158
JRNodemcu/temperature Data acquired at: 2018/01/14 21:25:19
Temperature: 27.25
JRNodemcu/temperature Data acquired at: 2018/01/14 21:25:21
Temperature: 27.25
```

Fonte: Autor.

Figura 32 – Cliente recebendo dados do *Broker* local



Fonte: Autor.

Os resultados da comunicação MQTT foram satisfatórios quando se utilizou uma rede Wi-Fi estável, nesses casos a comunicação entre a placa NodeMCU e o celular pode durar por horas. Em casos onde a rede Wi-Fi não era estável o microcontrolador se desconectava facilmente do *Broker*.

5 Conclusões

Este trabalho apresentou um projeto de monitoramento residencial de baixo custo, as ferramentas utilizadas no protótipo eram preferencialmente de *software* livre. Com o sistema proposto o usuário tem a possibilidade de verificar as condições de sua residência de forma remota e em tempo real.

Os resultados obtidos mostraram o funcionamento do sensor de temperatura TMP100 e do *reed switch* conectados a uma placa NodeMCU V1. Utilizando o protocolo MQTT foi possível transmitir os dados dos sensores pela *internet* e lê-los em um celular. Com o protocolo NTP foi possível saber a data e hora exata em que os sensores adquirem os dados. Os resultados exibem o potencial da placa de desenvolvimento NodeMCU V1 em projetos de IoT. O sistema desenvolvido pode operar com várias placas NodeMCU V1 e diversos sensores para monitorar diversas características de um ambiente. Tais resultados mostram que o projeto proposto é estável, sendo viável sua instalação em uma residência.

O código pode ser customizado para atender as mais diversas necessidades apresentadas pelos usuários. Um dos pontos negativos do sistema implementado é o fato do servidor privado funcionar apenas em rede local. Para contornar este problema é possível utilizar servidores públicos, que têm uma segurança questionável, ou utilizar servidores pagos, o que sairia do escopo de projeto de baixo custo. Também não foi possível criar um registro de dados no computador onde o servidor local foi instalado.

Em trabalho futuros, uma possível solução para que o servidor funcionasse na *internet* seria fazer o redirecionamento da porta 1883 no roteador e utilizar IP fixo. Com essas medidas seria possível acessar os dados em qualquer parte do mundo. Uma possível solução para acessar os dados em um *data log*, como foi proposto inicialmente, seria criar um banco de dados utilizando *Structured Query Language*, que é uma ferramenta elaborada justamente para projetar banco de dados. Ainda falando sobre trabalhos futuros, seria interessante testar outros tipos de sensores, como sensores de presença e movimento. Também seria viável organizar uma rede de vários microcontroladores NodeMCU para monitorar uma residência ou até mesmo complexo residencial.

Referências

- ALZAHANI, S. M. Sensing for the internet of things and its applications. *5th International Conference on Future Internet of Things and Cloud Workshops*, 2017. 23
- ASHTON, K. That internet of things in the real world, things matter more than ideas. *RFID Journal*, 2009. 21
- ATZORI ANTONIO IERA, G. M. L. The internet of things: A survey. *Computer Networks, Elsevier*, 2010. 21
- BRAGA, T.; RUIZ, L.; NOGUEIRA, J. Tecnologia de nós sensores sem fio. *DCC/UFMG*, 2003. 37
- Comer, D. *Interligação de redes com TCP/IP*. [S.l.]: Rio de Janeiro, Elsevier, 2006. 25
- Ebay. *ESP8266 price*. 2017. Acesso em: 22 de novembro de 2017. Disponível em: <<https://www.ebay.com/itm/Esp-12E-ESP8266-Serial-Port-WIFI-Transceiver-Wireless-Module-AP-STA/381374534484?epid=1279738338&hash=item58cbb19354:g:gOgAAOSweW5VY887>>. 35
- Eclipse Foundation. *Eclipse Paho Java Client*. 2017. Acesso em: 03 de janeiro de 2018. Disponível em: <<https://github.com/eclipse/paho.mqtt.java>>. 47
- ESP8266-ru. *Integrated Development Environment (IDE) for ESP8266 developers*. 2014. Acesso em: 10 de outubro de 2017. Disponível em: <<https://esp8266.ru/esplorer/>>. 41
- Espressif. *ESP8266EX Datasheet*. 2017. Acesso em: 22 de novembro de 2017. Disponível em: <http://espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf>. 35
- Evans, D. *The Internet of Things: How the Next Evolution of the Internet Is Changing Everything*. 2011. Acesso em: 03 de janeiro de 2018. Disponível em: <https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf>. 21, 22
- GITHUB. 2017. Acesso em: 30 de setembro de 2017. Disponível em: <<https://github.com/nodemcu>>. 41
- KANG, D.; PARK, M.; KIM, H. Room temperature control and fire alarm/suppression iot service using mqtt on aws. *International Conference on Computing, Communication and Automation*, 2017. 23
- KAUR, A.; JASUJA, A. Health monitoring based on iot using raspberry pi. *International Conference on Computing, Communication and Automation*, 2017. 23
- KODALI, R.; MAHESH, K. Low cost ambient monitoring using esp8266. *2nd International Conference on Contemporary Computing and Informatics*, 2016. 23
- KODALI, R. K.; MANDAL, S. Iot based weather station. *International Conference on Control Instrumentation Communication and Computational Technologies*, 2016. 24

LUA. *Lua documentation*. 2017. Acesso em: 22 de outubro de 2017. Disponível em: <<http://www.lua.org/docs.html>>. 36

MENS, J.-P. Using the mqtt iot protocol for unusual but useful purposes. *Admin network and security magazine*, 2015. 33

Michael Yuan. *Getting to know MQTT*. 2017. Acesso em: 03 de janeiro de 2018. Disponível em: <<https://www.ibm.com/developerworks/library/iot-mqtt-why-good-for-iot/index.html>>. 27

Mosquitto. *An Open Source MQTT v3.1/v3.1.1 Broker*. 2017. Acesso em: 22 de novembro de 2017. Disponível em: <<https://mosquitto.org>>. 47

MQTT. *MQTT documentation*. 2017. Acesso em: 30 de setembro de 2017. Disponível em: <<http://mqtt.org/documentation>>. 27

MQTT-SPY. 2017. Acesso em: 30 de setembro de 2017. Disponível em: <<https://github.com/eclipse/paho.mqtt-spy>>. 45

NIST. *What is a sensor?* 2009. Acesso em: 03 de janeiro de 2018. Disponível em: <<https://www.nist.gov/el/intelligent-systems-division-73500/definitions>>. 37

NodeMCU. *NodeMCU documentation*. 2017. Acesso em: 30 de setembro de 2017. Disponível em: <<https://nodemcu.readthedocs.io/en/master/>>. 37, 41

PINTO, F. D. M. Desenvolvimento de um protótipo de um sistema domótico. *Dissertação de Mestrado, Instituto Superior Técnico - Universidade Técnica de Lisboa*, 2010. 21

SAHA, S.; MAJUMDAR, A. Data centre temperature monitoring with esp8266 based wireless sensor network and cloud based dashboard with real time alert system. *Devices for Integrated Circuit (DevIC)*, 2017. 23

Texas Instruments. *TMP10x Temperature Sensor With I₂C and SMBus Interface with Alert Function in SOT-23 Package*. 2015. Acesso em: 9 de outubro de 2017. Disponível em: <<http://www.ti.com/lit/ds/symlink/tmp100.pdf>>. 42

Tosin, M.C. *Laboratório 14 I₂C*. 2015. Roteiro de laboratório da disciplina de Microprocessadores do curso de Engenharia Elétrica da UEL. 41

Travis CI. *NodeMCU custom builds*. 2017. Acesso em: 03 de janeiro de 2018. Disponível em: <<https://nodemcu-build.com/>>. 41

Valdez, J. and Beck J. *Understanding the I₂C Bus*. 2015. Acesso em: 22 de outubro de 2017. Disponível em: <<http://www.ti.com/lit/an/slva704/slva704.pdf>>. 33, 34

Apêndice A

CÓDIGO

```

1 -----file : init.lua
2
3 dofile ("realinit.lua")

```

```

1 ----- file : realinit.lua
2
3 app = require "application"
4 --package.loaded.app = nil
5 config = require "config"
6 --package.loaded.config = nil
7 setup = require "setup"
8 --package.loaded.setup = nil
9
10 setup.start()

```

```

1 -----file : config.lua
2
3 local module = {}
4
5 module.station_cfg = {}
6 module.station_cfg.ssid = "jrwifi"
7 module.station_cfg.pwd = "senha321"
8 module.station_cfg.auto = true
9 module.station_cfg.save = true
10
11 module.HOST = "192.168.0.107"
12 module.PORT = 1883
13 module.ID = node.chipid()
14
15 module.scl = 1 --Pino d2
16 module.sda = 2 --Pino d1
17 module.dev_addr = 0x48
18 module.config_addr = 0x01 --config register
19 module.temp_addr = 0x00 --temp register
20 module.resolution = 0x60 --resolucao 12 bits
21
22 module.ENDPOINT = "JRNodeMCU/"
23
24 module.rSwitchPin = 3

```

```

26 return module

1 —— file: setup.lua
2
3 local module = {}
4
5 function time_sync()
6     sntp.sync("a.st1.ntp.br")
7     sec = rtctime.get()
8     if sec ~= 0 then
9         tmr.stop(1)
10    tm = rtctime.epoch2cal(rtctime.get() -3600*2)
11    print("Connection established at:")
12    print(string.format("%04d/%02d/%02d %02d:%02d:%02d",
13        tm["year"], tm["mon"], tm["day"], tm["hour"],
14        tm["min"], tm["sec"]))
15    app.start()
16 else
17     print("No time acquired")
18     tmr.alarm(1,5000, 1, time_sync)
19 end
20 end
21
22 local function wifi_wait_ip()
23     if wifi.sta.getip() == nil then
24         print("IP unavailable, Waiting...")
25     else
26         tmr.stop(1)
27         print("IP is ".. wifi.sta.getip())
28         time_sync()
29
30     end
31 end
32
33 —Reed switch config
34 local function RS_config()
35     gpio.mode(config.rSwitchPin, gpio.INT, gpio.PULLUP)
36 end
37
38 ———TMP 100 config
39 local function TMP100()
40     i2c.setup(0, config.sda, config.scl, i2c.SLOW)
41     i2c.start(0)
42     i2c.address(0, config.dev_addr, i2c.TRANSMITTER)
43     i2c.write(0, config.config_addr)
44     i2c.write(0, config.resolution) —configura o config
45     register para 12bits de resol

```

```

46     i2c.stop(0)
47 end
48
49 function module.start()
50     RS_config()
51     TMP100()
52     print("Configuring Wifi ...")
53     wifi.setmode(wifi.STATION);
54     wifi.sta.config(config.station_cfg)
55     tmr.alarm(1, 1000, 1, wifi_wait_ip)
56 end
57
58 return module

```

```

1 —— file : application.lua
2
3 local module = {}
4 m = nil
5
6 local function RS_int()
7
8     tmr.alarm(4, 100, tmr.ALARM_SINGLE, function()
9         if gpio.read(3) == 0 then
10             print("abriu")
11             tm = rtctime.epoch2cal(rtctime.get() - 3600*2)
12             str_tm = string.format("%04d/%02d/%02d
13 %02d:%02d:%02d", tm["year"], tm["mon"], tm["day"],
14 tm["hour"], tm["min"], tm["sec"])
15             m:publish(config.ENDPOINT .. "DoorStatus", "The
16 door has CLOSEed at:\n"..str_tm, 2, 0)
17         else
18             print("fechou")
19             tm = rtctime.epoch2cal(rtctime.get() - 3600*2)
20             str_tm = string.format("%04d/%02d/%02d
21 %02d:%02d:%02d", tm["year"], tm["mon"], tm["day"],
22 tm["hour"], tm["min"], tm["sec"])
23             m:publish(config.ENDPOINT .. "DoorStatus", "The
24 door has OPENed at:\n"..str_tm, 2, 0)
25     end
26 end)
27 end
28
29
30 — Sends a simple ping to the broker
31 local function send_ping()
32     print("Sending data")
33     m:publish(config.ENDPOINT .. "ping", "id=" .. config.ID, 0

```

```

34     , 0)
35 end
36
37 — Reads and Sends temp data to the broker
38 local function RaS_temp()
39     i2c.start(0)
40     i2c.address(0, config.dev_addr, i2c.TRANSMITTER)
41     i2c.write(0, config.temp_addr)
42     i2c.start(0)
43     i2c.address(0, config.dev_addr, i2c.RECEIVER)
44     local temp = i2c.read(0, 2)
45     i2c.stop(0)
46     —faz a convercao do dado
47     local nbl = (string.byte(temp,2))/16
48     local bh = (string.byte(temp,1))*16
49     local conc = bh+nbl
50     local temperatura = 0.0625*conc
51     print(temperatura)
52     tm = rtctime.epoch2cal(rtctime.get() -3600*2)
53     str_tm = string.format("%04d/%02d/%02d %02d:%02d:%02d",
54     tm["year"], tm["mon"], tm["day"], tm["hour"], tm["min"],
55     tm["sec"])
56     print(str_tm)
57     m:publish(config.ENDPOINT .. "temperature", "Data acquired
58     at: "..str_tm.." \nTemperature: "..temperatura, 0, 0 )
59     —m:publish(config.ENDPOINT .. "temperature", temperatura,
60     0, 0)
61 end
62
63 — Sends my id to the broker for registration
64 local function register_myself()
65     m:subscribe(config.ENDPOINT .. config.ID,0,function(conn)
66         print("Successfully subscribed to data endpoint")
67     end)
68 end
69
70 local function con_success (con)
71     print("Connected to the broker!")
72     —register_myself()
73     — And then pings each 4000 milliseconds
74     tmr.alarm(6, 5000, tmr.ALARM_AUTO, send_ping)
75     tmr.alarm(5, 2000, tmr.ALARM_AUTO, RaS_temp)
76     gpio.trig(3, "both", RS_int)
77 end
78
79 function handle_mqtt_error(client, reason)
80     print("FAIL!")

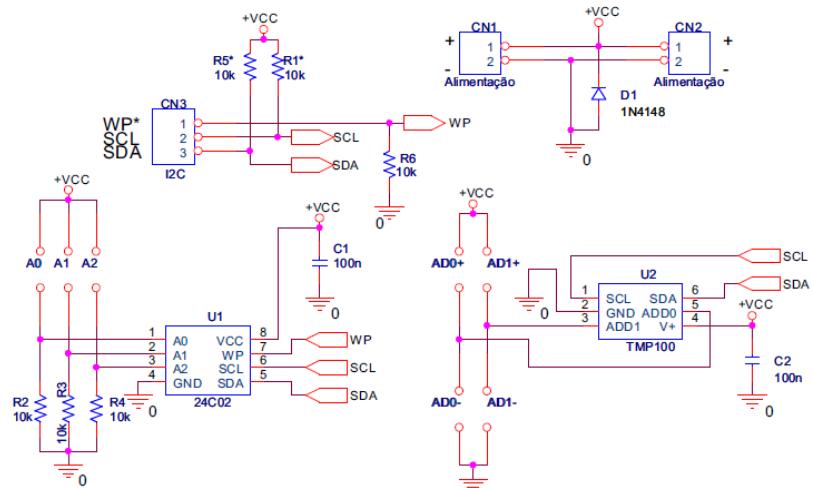
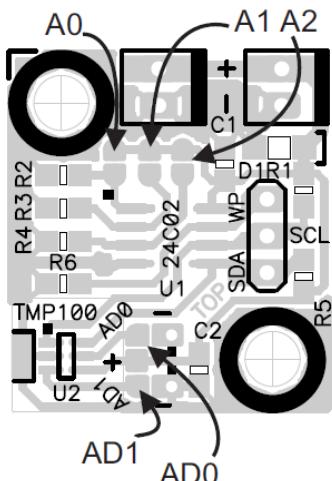
```

```
81     tmr.alarm(3, 1000, tmr.ALARM_SEMI, do_mqtt_connect)
82 end
83
84
85 function do_mqtt_connect()
86     -- Connect to broker
87     m:connect(config.HOST, config.PORT, 0, 0, con_success(m),
88     handle_mqtt_error)
89 end
90
91
92 function module.start()
93     m = mqtt.Client(config.ID, 120)
94     m:on("offline", handle_mqtt_error)
95     print("Client created")
96     do_mqtt_connect()
97     --m:on("offline", handle_mqtt_error)
98 end
99
100
101
102 return module
```


Anexo

Diagrama TMP100

I2C Shield



* Ítems opcionais
A0, A1, A2, AD0 e AD1 são espaços para jumper por solda

A memória 24C02 obedece o formato de endereçamento binário, já os pinos de endereçamento e os endereços I2C do TMP100 obedecem a seguinte tabela:

ADD1	ADD0	SLAVE ADDRESS
0	0	1001000
0	Float	1001001
0	1	1001010
1	0	1001100
1	Float	1001101
1	1	1001110
Float	0	1001011
Float	1	1001111