Desenvolvimento de aplicações baseadas em IOT com apoio de um middleware multi-protocolo.

John Kennedy Medeiros Alves de Oliveira[1], Paulo Gabriel Gadelha Queiroz[2]

Resumo: O objetivo deste trabalho consistiu em desenvolver um *middleware* multi-protocolo para apoiar o desenvolvimento de aplicações que necessitam garantir a comunicação entre dispositivos diversos. Foram implementados os microsserviços propostos na arquitetura que são responsáveis pelas funções de gerenciamento de dispositivos, gerenciamento de dados, gerenciamento de atuações e comunicação de entrada e saída. As funcionalidades implementadas foram validadas e verificadas por meio de testes automatizados e por meio do desenvolvimento de uma aplicação que utilizou um computador e dois Raspberry Pi. O sistema proposto consistiu em duas aplicações: uma aplicação para enviar mensagens para o *middleware*; e, uma aplicação capaz de receber mensagens do *middleware* e manipular os *LEDs* de acordo com a mensagem recebida. Observa-se que em um dispositivo havia um botão acoplado que quando pressionado enviava mensagens para o *middleware*, que então enviava uma mensagem de atuação para outro dispositivo, este com uma lâmpada de *LED* acoplada, que de acordo com o tipo da mensagem recebida, ligava ou desligava o *LED*.

Palavras-chave: *Middleware*. Internet das coisas. Sistema de Sistemas.

1. INTRODUÇÃO

A definição de Internet das coisas é composta por dois termos, Internet e coisas. Internet é uma infraestrutura global configurável, escalonável e dinamicamente expansível baseada em padrões e protocolos de comunicação. No contexto deste trabalho, coisas são objetos e dispositivos, físicos ou virtuais, com atributos e identidade, são heterogêneos por natureza e perfeitamente integrados à internet [1]. Então, Internet das coisas refere-se a objetos que se comunicam entre si ou com usuários, trocando informações através da internet.

Com a heterogeneidade dos dispositivos, e a existência de uma grande variedade de protocolos de aplicação [2], como também várias opções na camada física [3], para que esses dispositivos heterogêneos possam se comunicar e trocar informações, mais facilmente, é necessário uma plataforma intermediária chamada *middleware*, disponibilizando abstração para aplicações de dispositivos heterogêneos.

Um *middleware* tem o papel de disponibilizar um modelo de programação unificado para desenvolvedores de aplicações e mascarar problemas de heterogeneidade e distribuição existentes entre os dispositivos [8]. Com a ajuda de um *middleware*, possibilita-se a construção de sistemas mais complexos de forma mais simples. Enquanto participava de um projeto de pesquisa científica com tema de Internet das coisas, junto com o aluno de mestrado Rodolfo Felipe Medeiros Alves, o autor deste trabalho colaborou com o desenvolvimento de um *middleware* multi-protocolo para apoiar o desenvolvimento de aplicações para Internet das coisas.

Dessa forma, o objetivo da pesquisa é apresentar a primeira versão do *middleware* desenvolvido e uma aplicação de Internet das coisas para validar o *middleware*. A validação do *middleware* foi realizada através da utilização de testes automatizados e por meio da implementação de de duas aplicações que utilizam o *middleware* por meio de protocolos distintos, uma utilizando MQTT e outra utilizando REST, além disso, uma aplicação semelhante que não utiliza o *middleware* também foi desenvolvida para fins de comparação.

O restante deste artigo está organizado da seguinte maneira: Na Seção 2 são apresentados os trabalhos relacionados; Na Seção 3, apresenta-se como o *middleware* foi desenvolvido; Na Seção 4, apresenta-se o desenvolvimento de uma aplicação para Internet das coisas que utiliza o *middleware*; Na Seção 5, apresenta-se a discussão dos resultados obtidos; Por fim, na Seção 6, são apresentados as conclusões deste trabalho.

2. TRABALHOS RELACIONADOS

O objetivo comum no desenvolvimento de *middlewares* para Internet das coisas é desenvolver uma estrutura de adaptação que atue como interpretador e que forneça abstração e serviços para facilitar no desenvolvimento de aplicações de Internet das coisas [4], [5].

MESMOUDI, Yasser et al(2018) propoem um *middleware* orientado a serviços que atua de forma centralizada, capaz de lidar com o problema da heterogeneidade na comunicação, generalizar o código implementado para as mais usadas plataformas e tratar o problema de circulação de dados heterogêneos.

ALVARO et al (2019) propuseram um *middleware open source* como um serviço em nuvem para conectar e modelar dispositivos de fusão de dados com base em Internet das coisas, *Big Data* e tecnologias de nuvem, utilizando os protocolos REST, MQTT e CoAP.

Outro trabalho semelhante foi proposto por ESPOSTE et al.,(2019) com o Interscity, que propõe um *middleware open source* para apoiar os desafios na infraestrutura de software de cidades inteligentes, focado em redes e computação distribuída de alto desempenho, modelagem matemática e análise de dados. O *middleware* utiliza os protocolos REST e MQTT e oferece serviços como gerenciamento de serviços e descoberta de recursos da cidade baseados em contexto.

Além desses trabalhos, foram encontrados outros *middlewares* com o propósito de simplificar o desenvolvimento de aplicações IOT por meio da abstração da comunicação entre os dispositivos. O diferencial deste trabalho consiste em dominar essa tecnologia e ser capaz de instanciá-la em redes locais, onde o acesso a internet não existe ou não é bom.

3. DESENVOLVIMENTO

3.1. Middleware

O *middleware* desenvolvido propõe possibilitar o gerenciamento de dispositivos heterogêneos, gerenciamento e persistência de dados, processamento de informações, além de abstrair a comunicação entre dispositivos que utilizem de diferentes tipos de protocolo na camada de aplicação, tais como CoAP, MQTT, AMQP, REST e XMPP. Assim, simplificando o processo de desenvolvimento de aplicações complexas para sistemas de Internet das coisas.

O desenvolvimento do *middleware* iniciou-se a partir de um estudo bibliográfico, utilizado como base, para o planejamento e execução de uma revisão sistemática (RS) com o tema: *middlewares* para apoiar o desenvolvimento de aplicações em Internet das coisas. A partir dos resultados da RS, foi definida uma arquitetura de *middleware* baseada em microsserviços que utiliza os protocolos identificados na RS como os principais mais utilizados no contexto de IOT, que são o MQTT,AMQP, CoAP, HTTP/REST, e LoRaWAN.

Em seguida, realizou-se o estudo das ferramentas necessárias para a implementação da primeira versão do *middleware*, a partir da arquitetura proposta. Adicionalmente, foram selecionadas ferramentas para apoiar os testes automatizados. Foram implementados os microsserviços propostos na arquitetura que são responsáveis pelas funções de gerenciamento de dispositivos, gerenciamento de dados, gerenciamento de atuações e comunicação de entrada e saída. Observa-se que este trabalho foi feito em parceria com um aluno de mestrado do PPgCC.

A arquitetura do *middleware* é orientada a microsserviços, composta por quatro camadas: a camada principal, a camada de comunicação, a camada de protocolos de aplicação e a camada física, onde se encontram os dispositivos. Na camada principal estão microsserviços principais, que não realizam comunicação direta com os dispositivos: Gerenciador de Dispositivos, Descobridor de Serviços, Gerenciador de Atuação e Gerenciador de Dados. Estes microsserviços são responsáveis pelo gerenciamento dos dispositivos, gerenciamento das atuações, descobrimento de serviços e gerenciamento dos dados.

A camada de comunicação é encarregada pela comunicação entre as aplicações externas e os microsserviços da camada principal, nela estão contidos os seguintes microsserviços: Conector de Dispositivos, Comunicador de Entrada, Comunicador de Atuação e Catalogador de Serviços. A camada de protocolos do *middleware* oferece suporte a quatro protocolos para a comunicação: MQTT,AMQP, CoAP, HTTP/REST, e LoRaWAN, e, na camada física, estão os dispositivos, através das aplicações de Internet das coisas. Um esboço geral da arquitetura do *middleware* está ilustrada abaixo na Figura 1, onde estão circulados em vermelho os microsserviços implementados e os protocolos utilizados na primeira versão implementada do middleware.

Nesta primeira versão, foram implementados, da camada principal, os microsserviços: gerenciador de dispositivos, gerenciador de dados e gerenciador de atuação. Da camada de comunicação implementou-se: comunicador de entrada, comunicador de atuação e catalogador de serviços. Essa versão utiliza os protocolos MQTT, AMQP, CoAP e HTTP/REST. A implementação dos microsserviços foi feita com a linguagem de programação Javascript [9], por meio do servidor de aplicação Nodejs [10] e banco de dados não relacional MongoDB [11]. Para os testes automatizados dos microsserviços foi utilizado o framework Jestjs [12]. Na criação da aplicação para o dispositivo Raspberry Pi, utilizada para manipulação de sensores e atuadores e testar o *middleware*, foi utilizada a linguagem de programação Python [13].

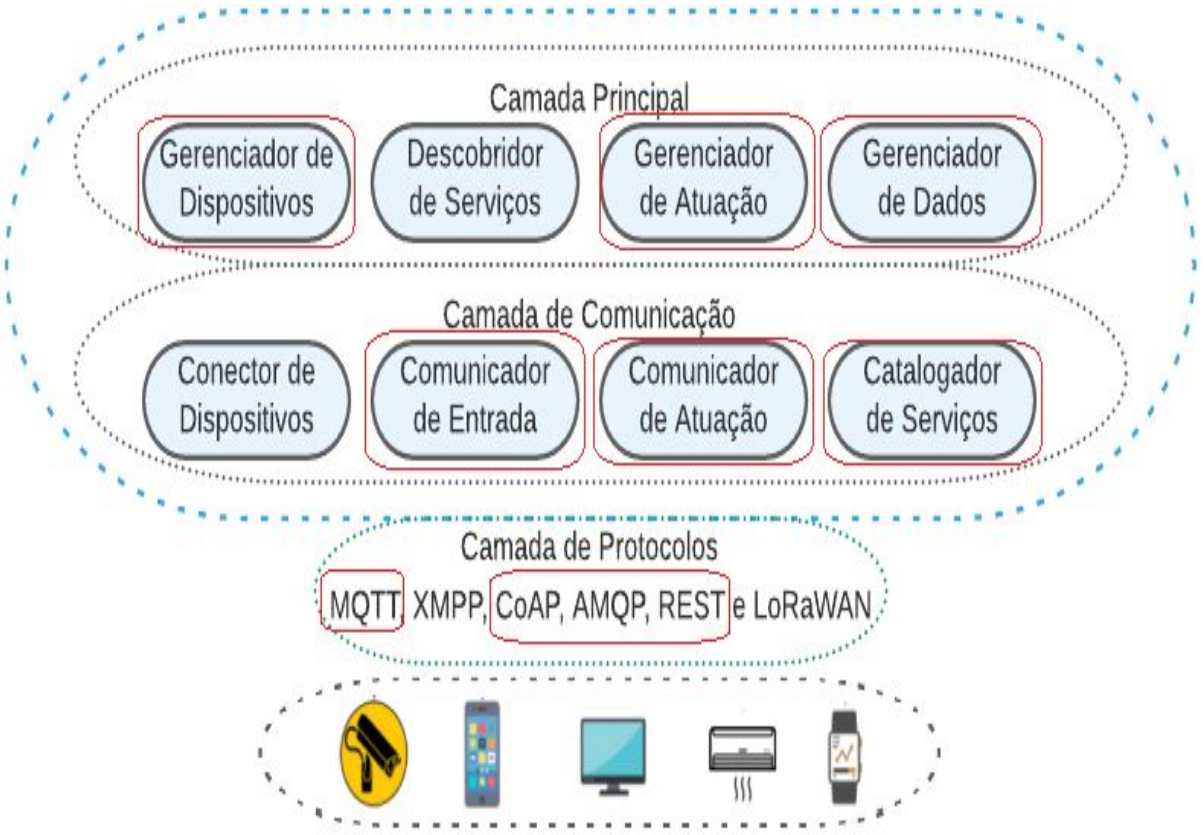

Figura 1. Esboço da arquitetura do *middleware*. (Autoria Própria)

4. VALIDAÇÃO DO MIDDLEWARE

As funcionalidades implementadas na primeira versão do *middleware* foram validadas e verificadas por meio de testes automatizados, realizados utilizando o framework Jestjs [12], e por meio do desenvolvimento de um sistema utilizando dois dispositivos Raspberry Pi, cada um conectado a uma placa *protoboard*, uma com um botão acoplado e outra com uma lâmpada de *LED* acoplada, visto que o Raspberry Pi oferece suporte para entrada de botões e leds além de existir bibliotecas que ajudam a gerenciar essa utilização, como também o fato desse dispositivo ser capaz de enviar mensagens através de mais de um protocolo, como o REST e MQTT, podendo realizar essa comunicação entre os dispositivos utilizando a infraestrutura do middleware.

O sistema proposto consistiu em duas aplicações: uma aplicação para enviar mensagens para o *middleware*; e, uma aplicação capaz de receber mensagens do *middleware* e manipular o *LED* de acordo com a mensagem recebida. Observa-se que, em um dos dispositivos havia um botão acoplado que, ao ser pressionado, enviava mensagens para o *middleware*. Em seguida, o *middleware* enviava uma mensagem de atuação para o outro dispositivo, este com uma lâmpada de *LED* acoplada, que de acordo com o tipo de mensagem recebida, ligava ou desligava o *LED*.

Os microsserviços implementados tiveram suas funções verificadas através de testes de unidade automatizados, utilizando o framework Jestjs [12], cada função foi testada com parâmetros de entrada corretos e também com parâmetros de entrada incorretos, verificando se o comportamento em ambos os casos estava de acordo com o esperado. Ilustra-se na Figura 2, parte da implementação de um arquivo de testes automatizados do módulo gerenciador de dispositivos. Entre as linhas 1 e 10 estão as importações necessárias e as configurações iniciais. Entre as linhas 12 e 23 é construído um objeto de um dispositivo para servir de semente e ser inicialmente gravado no banco de dados de teste. Entre as linhas 25 e 40, foi implementada uma função do Jestjs [12] que utiliza a função do gerenciador de dispositivos para atualizar um dispositivo já existente, passando parâmetros de entrada corretos. Na linha 42, é definido que comportamento é esperado do módulo ao utilizar a função anterior, verificando seu retorno.

```
const server          = require('../server')
const supertest       = require('supertest')
const request         = supertest(server)
const Device          = require('../Model/device')

const databaseName    = 'test05'
setupDB(databaseName)

jest.setTimeout(30000);

const seedDevice      = {
  "uuid": "007",
  "latDefault": 0,
  "lonDefault": 0,
  "resource": [
    "Teste"
  ],
  "uri": "testeUri",
  "protocol": "MQTT",
  "describe": "Example",
  "typeDevice": "Sensor"
}

it("Should successfully update a device's 'description' attribute", async done => {
  const seededDevice = new Device.db(seedDevice)
  await seededDevice.save()

  const res = await request.post(`/devices/update`).send({
    "uuid": "007",
    "latDefault": 0,
    "lonDefault": 0,
    "resource": [
      "Teste"
    ],
    "uri": "testeUri",
    "protocol": "MQTT",
    "describe": "ExampleUpdated",
    "typeDevice": "Sensor"
  })

  expect(res.body.describe).toBe("ExampleUpdated")

  done()
})
```

Figura 2. Parte da implementação de um arquivo de teste automatizado para o módulo gerenciador de dispositivo. (Autoria Própria)

Na aplicação desenvolvida para validar o *middleware*, um dos dispositivos Raspberry Pi estava conectado a uma placa *protoboard* com um *LED* acoplado, conforme pode ser observado na Figura 3. Conectado a Internet, o dispositivo se conecta a um *broker* MQTT e é inscrito em um tópico no qual, conforme recebe mensagens, acende ou apaga o *LED*. O segundo dispositivo também estava conectado a uma placa *protoboard*, por sua vez com um botão acoplado que ao ser pressionado enviava mensagens utilizando o protocolo de aplicação MQTT ou REST.

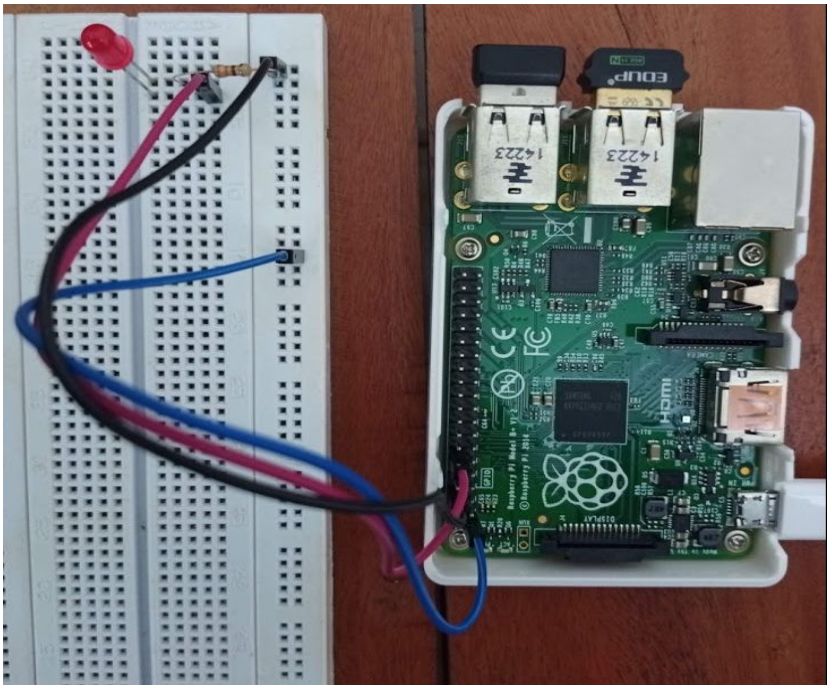

Figura 3. Raspberry Pi conectado a placa *protoboard* com *LED* acoplado. (Autoria Própria)

Para construção desta aplicação, a infraestrutura do middleware rodava na nuvem e os dispositivos foram registrados na infraestrutura do *middleware* como também foi registrada a atuação de trocar o estado da lâmpada. Foi implementado um programa para o dispositivo 1, conectado a rede e que ao ter o botão acoplado pressionado, envia mensagens para o *middleware* através do protocolo MQTT, o middleware então recebe esta mensagem através de seu micro serviço comunicador de entrada e verifica se há alguma atuação registrada para esse dispositivo, para que essa atuação possa ser enviada através do micro serviço comunicador de atuação para o dispositivo destino através de seu protocolo de aplicação. Ilustra-se na Figura 4, nas linhas 7, 8 e 9, o código responsável pela configuração *protoboard* e em qual slot o botão está conectado. Na linha 11, é instanciado o cliente mqtt e na linha 14 a conexão com o *broker* é estabelecida. Observe que na linha 18 é verificado se o botão está pressionado, e caso esteja, nas linha 20 e 21 é definida e publicada a mensagem para o tópico no qual o *middleware* está inscrito, em específico na linha 20 está descrito o tópico e na linha 21 está descrita a mensagem, contendo o id do dispositivo, sua localização e o corpo da atuação.

```
1  import paho.mqtt.client as mqtt
2  import RPi.GPIO as GPIO
3  import time
4  import os
5
6  # setup the protoboard button
7  GPIO.setmode(GPIO.BCM)
8  GPIO.setwarnings(False)
9  GPIO.setup(24,GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
10
11 client = mqtt.Client()
12
13 # Broker address and port to connect
14 client.connect(os.getenv('MIDDLEWARE_BROKER_URL'), 1883)
15
16 while(1):
17     # If the button is pressed
18     if GPIO.input(24) == True:
19         # Broker topic and message to send
20         client.publish("ufersa/middleware/iot",
21          '{"deviceId": "8", "lon": -5.0, "lat": -6.0, "object": {"message": "switch-lamp"}}')
22         time.sleep(1)
23
```

Figura 4. Implementação do programa utilizado no dispositivo 1 que ao botão ser pressionado, envia mensagens para o *middleware* utilizando o protocolo MQTT. (Autoria Própria)

O mesmo programa citado anteriormente para o dispositivo 1, que contém o botão acoplado foi também implementado utilizando um protocolo de aplicação diferente do anterior, o protocolo REST, que como o programa anterior, também envia mensagens para o middleware, que recebe essas mensagens através de seu comunicador de entrada e verifica se há alguma atuação registrada para esse dispositivo para que essa atuação possa ser enviada através do comunicador de atuação utilizando o protocolo de aplicação registrado para o dispositivo destino. Esta implementação está ilustrada abaixo na Figura 5. Da linha 7 a linha 9 está o código responsável pela configuração da placa *protoboard* e o botão acoplado. Na linha 12 é definido o endereço que *middleware* recebe requisições REST, na linha 14 é montado o corpo da mensagem a ser enviada, contendo o id único do dispositivo, sua latitude e longitude e a descrição da atuação. Na linha 23, é verificado se o botão está pressionado, e caso esteja, e então na linha 25 é realizada uma requisição REST para o endereço do *middleware* enviando o objeto da mensagem definida na linha 14.

```
1   import time
2   import requests
3   import json
4   import os
5
6   # setup the protoboard button
7   GPIO.setmode(GPIO.BCM)
8   GPIO.setwarnings(False)
9   GPIO.setup(24,GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
10
11  # Middleware input communicator url
12  url     = os.getenv('MIDDLEWARE_REST_ENDPOINT')
13  # Message payload
14  payload = {
15   "deviceId": "8",
16   "lon": -5.0, "lat": -6.0,
17   "object": {"message": "switch-lamp"}
18   }
19
20
21  while(1):
22      # If the button is pressed
23      if GPIO.input(24) == True:
24          # POST with JSON
25          response = requests.post(url, data=json.dumps(payload))
26          print(response.status_code)
27          time.sleep(1)
```

Figura 5. Implementação do programa utilizado no dispositivo 1, que ao botão ser pressionado, envia mensagens para o *middleware* utilizando o protocolo REST. (Autoria Própria)

Na Figura 6 apresenta-se a implementação do programa utilizado no dispositivo 2, que está com o *LED* acoplado e, ao receber mensagens enviadas pelo comunicador de atuação do *middleware* através do protocolo MQTT, acende ou apaga o *LED*. Entre as linhas 8 e 11 encontra-se o código responsável pela configuração da *protoboard* que contém o *LED*. Entre as linhas 13 e 21 define-se que, no momento em que a conexão for estabelecida com o *broker*, uma variável que representa o estado inicial do *LED* é criada, e, é realizada a inscrição em um tópico deste *broker*. Observa-se que o tópico de inscrição é aquele em que o *middleware* terá que enviar mensagens para se comunicar com o dispositivo. Entre as linhas 23 e 41, é definido que, ao chegar uma mensagem no tópico do *broker* em que o dispositivo está inscrito, essa mensagem é tratada e, caso essa mensagem corresponda a palavra esperada, como verificado na linha 29, é trocado o estado da lâmpada, tanto na variável de controle de estado, quanto na *protoboard*, apagando-a ou acendendo-a. Observe que o código das linhas 43, 44 e 45 é responsável por instanciar um cliente MQTT e definir que esse cliente utilizará as definições citadas anteriormente, e então, na linha 48 é estabelecida a conexão com o *broker* MQTT.

Observa-se que foram desenvolvidas duas aplicações para o dispositivo do botão, uma em MQTT e outra em REST. Entretanto, visto que a comunicação é intermediada pelo *middleware*, não há problemas se a mensagem for enviada via REST ou MQTT, pois o *middleware* consegue repassar para o protocolo MQTT que é utilizado aqui.

```python
import paho.mqtt.client as mqtt
import RPi.GPIO as GPIO
import time
import json
import os

# Setup the protoboard led
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(18,GPIO.OUT)
GPIO.output(18, GPIO.LOW)

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))

    # Leds current state, 0 = off, 1 = on
    global led_status
    led_status = 0

    # Broker topic to subscribe on
    client.subscribe("middleware/ufersa/iot")

def on_message(client, userdata, msg):
    data_payload = json.loads(str(msg.payload))

    # Prints ( Topic - Message )
    print(msg.topic + " - " + str(data_payload["command"]["message"]))

    if data_payload["command"]["message"] == "switch":
        global led_status

        # If off, switchs to on
        if led_status == 0:
            led_status = 1
            GPIO.output(18, 1)
            time.wait(1)
        # If on, switchs to off
        else:
            led_status = 0
            GPIO.output(18, 0)
            time.wait(1)

client             = mqtt.Client()
client.on_connect  = on_connect
client.on_message  = on_message

# Broker address and port to connect
client.connect(os.getenv('MIDDLEWARE_BROKER_URL'),1883)
client.loop_forever()
```

Figura 6. Implementação do programa utilizado no dispositivo 2 que recebe mensagens do *middleware*, via protocolo MQTT, e muda o estado da lâmpada baseado na mensagem recebida.(Autoria Própria)

As mesmas aplicações utilizando o protocolo MQTT foram também implementadas sem a utilização do *middleware*, a fim de comparar as dificuldades no processo de desenvolvimento dessa aplicação com e sem o apoio da infraestrutura do *middleware*. Na Figura 7, está a implementação do programa para o dispositivo 1, que está com o botão acoplado, que ao ser pressionado, envia uma mensagem, dessa vez diretamente para o outro dispositivo que contém o *LED*, para que este troque o estado do *LED*. Nas linhas 7, 8 e 9, está o código responsável pela configuração da placa *protoboard* com o botão, na linha 11 é instanciado um cliente MQTT, na linha 14 é estabelecida a conexão MQTT com o *broker*, recebendo como parâmetros seu endereço e sua porta. Observe que na linha 18 é verificado se o botão está pressionado, e caso esteja, então na linha 12 é publicada uma mensagem no tópico do *broker* MQTT , no qual o dispositivo que contém o *LED* está inscrito. Os parâmetros para essa publicação são: o tópico do *broker* e a mensagem que deseja enviar. Neste exemplo, é simplesmente, a mensagem que o outro dispositivo irá receber diretamente sem passar pelo *middleware*.

```
1  import paho.mqtt.client as mqtt
2  import RPi.GPIO as GPIO
3  import time
4  import os
5
6  # setup the protoboard button
7  GPIO.setmode(GPIO.BCM)
8  GPIO.setwarnings(False)
9  GPIO.setup(24,GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
10
11 client = mqtt.Client()
12
13 # Broker address and port to connect
14 client.connect(os.getenv('BROKER_URL'), 1883)
15
16 while(1):
17     # If the button is pressed
18     if GPIO.input(24) == True:
19         # Broker topic and message to send
20         client.publish("lamproom30", "switch")
21         time.sleep(1)
22
```

Figura 7. Programa utilizado no dispositivo 1, que ao botão ser pressionado, envia mensagens diretamente para o tópico em que o dispositivo que contém o *LED* acoplado está inscrito.(Autoria Própria)

Para o dispositivo 2, que está com o *LED* acoplado, a implementação do programa para receber mensagens e acender ou apagar o *LED* está ilustrada abaixo na Figura 8. Da linha 8 a linha 11, está o código responsável pela configuração da placa *protoboard* e o *LED* acoplado, da linha 13 a linha 19, é definido que, no momento em que a conexão for estabelecida com o *broker*, uma variável que representa o estado inicial do *LED* é criada, e, é realizada a inscrição em um tópico deste *broker*, tópico no qual o dispositivo que contém o botão terá que enviar mensagens para se comunicar com esse dispositivo. Da linha 21 a linha 38, é definido que, sempre que chegar uma mensagem no tópico em que está inscrito, caso essa mensagem corresponda a palavra esperada, como verificado na linha 26, em que a variável contém diretamente a palavra a ser verificada, é trocado o estado da lâmpada, tanto na variável de controle de estado, quanto na *protoboard*, apagando ou a acendendo o *LED*. Observe que entre as linhas 40 e 42 um cliente MQTT é instanciado e é definido que ele utilizará as definições citadas anteriormente. Por fim, na linha 44 uma conexão com o *broker* MQTT é estabelecida.

```python
import paho.mqtt.client as mqtt
import RPi.GPIO as GPIO
import time
import json
import os

# Setup the protoboard led
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(18,GPIO.OUT)
GPIO.output(18, GPIO.LOW)

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    global led_status
    led_status = 0

    # Broker topic to subscribe on
    client.subscribe("lamproom30")

def on_message(client, userdata, msg):

    # Prints ( Topic - Message )
    print(msg.topic + " - " + msg.payload)

    if msg.payload == "switch":
        global led_status

        # If off, switchs to on
        if led_status == 0:
            led_status = 1
            GPIO.output(18, 1)
            time.wait(1)
        # If on, switchs to off
        else:
            led_status = 0
            GPIO.output(18, 0)
            time.wait(1)

client             = mqtt.Client()
client.on_connect  = on_connect
client.on_message  = on_message

# Broker address and port to connect
client.connect(os.getenv('BROKER_URL'),1883)
client.loop_forever()
```

Figura 8. Programa utilizado no dispositivo 2, que recebe mensagens via MQTT e acende ou apaga o *LED*, sem a utilização do *middleware*. (Autoria Própria)

5. DISCUSSÃO DOS RESULTADOS

Observando-se o grau de dificuldade durante processo do desenvolvimento da aplicação para a validação do *middleware*, identificou-se que em aplicações simples, onde há pouca quantidade de dispositivos, sem a necessidade então de cadastrar e gerenciar os dispositivos, como também o fato dos dispositivos não utilizarem protocolos diferentes para comunicação, e, sem a necessidade de persistir os dados da comunicação entre os dispositivos, a complexidade para o desenvolvimento com e sem o *middleware* é semelhante. Entretanto, em sistemas de Internet das coisas mais complexos, com grande quantidade de dispositivos, havendo dispositivos que utilizam protocolos diferentes um dos outros, onde haja necessidade de persistência dos dados, seja para controle ou tratamento e estatísticas, o apoio da infraestrutura do *middleware* seria essencial, pois o *middleware* oferece gerenciamento de dispositivos, com identificação única para cada um deles, como também banco de dados para persistir os dados coletados, gerenciamento de atuações para suprir as demandas conforme necessário pelo sistema.

Como dispositivos que utilizam protocolos de aplicação do tipo que necessitam de um *broker*, como por exemplo o protocolo MQTT, e estes necessitam estar conectados a um mesmo *broker* em que o *middleware*, uma possível melhoria futura é adicionar um *broker* próprio a infraestrutura do middleware.

6. CONCLUSÃO

O objetivo deste trabalho foi apresentar a arquitetura de um *middleware* multiprotocolo construído para facilitar a criação e integração de dispositivos no contexto de aplicações IOT. O middleware continua em evolução, com novas funcionalidades em desenvolvimento. Apesar de existirem outros middlewares semelhantes, estes não dão suporte a múltiplos protocolos ou estão disponíveis apenas na nuvem, enquanto o middleware implementado, além de dar suporte a múltiplos protocolos, pode ser instanciado e configurado em uma infraestrutura de rede local, em fazendas, por exemplo, que não tenham acesso a internet, sendo útil no apoio de desenvolvimento de aplicações de Internet das coisas também nesses lugares.

Para validar o uso da infraestrutura do *middleware* desenvolvido foram implementadas algumas aplicações baseadas em Internet das coisas. Devido ao grau de complexidade no processo de desenvolvimento de sistemas de Internet das coisas. Diante da simplicidade do sistema, as diferenças no processo do desenvolvimento utilizando ou não o *middleware* são poucas, com isso ficou clara a necessidade de validar o *middleware* com aplicações mais complexas. Diante disso, neste momento está sendo projetada uma aplicação para auxiliar na agricultura de precisão.

Por fim, vale destacar que o desenvolvimento deste trabalho trouxe grandes aprendizados, tais como:

- novas linguagens programação, como Javascript [9] para a criação dos serviços e Python [13] para a criação das aplicações de Internet das coisas;
- estudo e utilização de testes automatizados utilizando a ferramenta Jestjs [12];
- montagem e conexão de dispositivos utilizando diferentes protocolos de rede.

7. REFERÊNCIAS BIBLIOGRÁFICAS

[1] BANDYOPADHYAY, Soma et al. Role of middleware for internet of things: A study. International Journal of Computer Science and Engineering Survey, v. 2, n. 3, p. 94-105, 2011.

[2] SZTAJNBERG, A.; STUTZEL, M.; MACEDO, R. Protocolos de aplicação para a Internet das coisas: conceitos e aspectos práticos. In: CSBC. Jornadas de Atualização em Informática. Natal, RN: SBC, 2018.

[3] WEISER, M. The computer for the 21st century. Scientific American, p. 94–104, September 1991. Disponível em: <https://www.lri.fr/~mbl/Stanford/CS477/papers/Weiser-SciAm.pdf>.

[4] RAZZAQUE, Mohammad Abdur et al. Middleware for internet of things: a survey. IEEE Internet of things journal, v. 3, n. 1, p. 70-95, 2015.

[5] MESMOUDI, Yasser et al. A Middleware based on Service Oriented Architecture for Heterogeneity Issues within the Internet of Things (MSOAH-IoT). Journal of King Saud University-Computer and Information Sciences, 2018.

[6] LUIS BUSTAMANTE, Alvaro; PATRICIO, Miguel A.; MOLINA, José M. Thinger. io: An Open Source Platform for Deploying Data Fusion Applications in IoT Environments. Sensors, v. 19, n. 5, p. 1044, 2019.

[7] DEL ESPOSTE, Arthur M. et al. InterSCity: A Scalable Microservice-based Open Source Platform for Smart Cities. In: SMARTGREENS. 2017. p. 35-46.

[8] BLAIR, Gordon S. et al. An architecture for next generation middleware. In: Middleware'98. Springer, London, 1998. p. 191-206.

[9] FLANAGAN, David; LIKE, Will Sell. JavaScript: The Definitive Guide, 5th. 2006.

[10] TILKOV, Stefan; VINOSKI, Steve. Node. js: Using JavaScript to build high-performance network programs. IEEE Internet Computing, v. 14, n. 6, p. 80-83, 2010.

[11] CHODOROW, Kristina. MongoDB: the definitive guide: powerful and scalable data storage. " O'Reilly Media, Inc.", 2013.
[12] Jest. 2020. Página inicial. Disponível em: <https://jestjs.io//>. Acesso em: 10 de jun. de 2020
[13] VAN ROSSUM, G.; DRAKE, F. L. Python 3 Reference Manual; CreateSpace. Scotts Valley, CA, 2009.