Breve Tutorial em Programação de Microcontroladores ESP8266 com MicroPython

Fernando F. Ramborger, Maria F. Totti, Maysa F. dos Santos, Evandro C. Vilas Boas
Laboratório de Cyber Segurança e Internet das Coisas (CS&I Lab.), Instituto Nacional de Telecomunicações - Inatel
fernandoramborger@gec.inatel.br, mariafernanda.mf@get.inatel.br, maysa.santos@gea.inatel.br, evandro.cesar@inatel.br

***Abstract*—This work presents a brief tutorial on the development of IoT (Internet of Things) applications using the MicroPython programming language and the ESP8266 microcontroller. MicroPython is a Python adaptation, which sets optimized standard libraries. It is used to program the NodeMCU platform, which integrates the ESP8266 microcontroller. This tutorial demonstrates how to install the integrated development environment uPyCraft, update the ESP8266 firmware, manipulate GPIO pins and make an MQTT (Message Queuing Telemetry Transport) connection.**

***Index Terms*—Internet of Things, Python, MicroPython, NodeMCU, ESP8266.**

***Resumo*—Este trabalho apresenta um breve tutorial sobre o desenvolvimento de aplicações IoT (*Internet of Things*) empregando a linguagem de programação MicroPython e o microcontrolador ESP8266. O MicroPython é uma adaptação do Python que integra um conjunto de bibliotecas padrão otimizadas. Emprega-se essa linguagem para programar a plataforma de desenvolvimento NodeMCU, que utiliza o microcontrolador ESP8266. Demonstra-se como instalar o ambiente de desenvolvimento integrado uPyCraft, atualizar o *firmware* do ESP8266, manipular pinos GPIO e realizar uma conexão MQTT (*Message Queuing Telemetry Transport*).**

***Palavras chave*—Internet das Coisas, Python, MicroPython, NodeMCU, ESP8266.**

I. Introdução

A conexão de dispositivos diversos à Internet permite o controle remoto de suas funções e/ou o sensoriamento de aspectos naturais e humanos [1]. Essa abordagem define de modo geral o termo Internet das Coisas ou *Internet of Things* (IoT), inicialmente proposto por Kevin Ashton em 1999. Em uma rede IoT, considera-se a conexão e comunicação entre dispositivos para troca de dados e comandos [2, 3]. Conectam-se aparelhos eletrônicos à Internet para controle de seu funcionamento. Utilizam-se sensores em diversas aplicações para coleta de dados, abrangendo aplicações industriais, residências, agrícolas, etc. Atuadores executam comandos específicos para modificação de um ambiente e/ou interação com pessoas.

Para a conexão à Internet e comunicação com outros dispositivos e servidores, definem-se alguns protocolos de comunicação voltados para atender aos requisitos de dispositivos IoT como consumo de energia, abrangência geográfica, qualidade de serviço e vida útil de baterias [4]. Dentre esses protocolos, cita-se o MQTT (*Message Queuing Telemetry Transport*) amplamente utilizado para desenvolver aplicações IoT [5, 6]. Esse protocolo permite a comunicação entre dispositivos IoT de uma mesma aplicação por meio de uma arquitetura publica/assina. Os dispositivos conectam-se diretamente em um servidor denominado de *broker* que gerencia a troca de informações estruturando-as em tópicos. Para enviar uma informação à rede, um dispositivo troca mensagens com o *broker* para publicá-la em um tópico. Outros dispositivos interessados em acessar esses dados, indicam ao *broker* a assinatura do tópico. Referem-se a esses processos respectivamente como *publish* e *subscribe* [5, 6].

No meio acadêmico, empregam-se algumas plataformas de *hardware* para o aprendizado e desenvolvimento de aplicações IoT. Geralmente, utilizam-se os microcontroladores da família ATmega e o ESP8266 embarcados em plataformas de *hardware* comumente referenciadas como Arduino e NodeMCU [7, 8]. Para ambos os microcontroladores, a linguagem de programação usual é baseada no C/C++. Todavia, pode-se adaptar o microcontrolador ESP8266 para que a programação do *hardware* seja realizada em linguagem Python específica, denominada MicroPython [9, 10, 11]. O MicroPython é uma versão condensada do Python 3 e embarcada em *hardware*.

Esse trabalho apresenta um breve tutorial para a programação de microcontroladores ESP8266 empregando a linguagem MicroPython. Elencam-se as principais informações para uso da linguagem MicroPython na programação do NodeMCU, representando um resumo de um caderno tutorial desenvolvido pelos autores. Estruturou-se o trabalho em seis seções. Na Seção II, abordam-se alguns aspectos de *hardware* da plataforma NodeMCU como principais características e pinagem. Discute-se a linguagem de programação em MicroPython na Seção III, elencando as principais bibliotecas e comandos. Na Seção IV, introduz-se o ambiente de desenvolvimento integrado uPyCraft, demonstra-se como atualizar o *firmware* da plataforma NodeMCU, manipulam-se pinos GPIO (*General Purpose Input/Output*) digitais e analógico e explora-se a comunicação entre plataformas. Para conectar o NodeMCU à rede em conjunto ao uso do protocolo MQTT, utiliza-se a Plataforma ThingSpeak [12] para desenvolver uma conexão e enviar dados na Seção V. Apresentam-se as conclusões e trabalhos futuros na Seção VI.

II. Módulos ESP-XX e Plataforma NodeMCU

Os módulos ESP permitem estabelecer a conexão entre o microcontrolador e a Internet por meio de uma rede sem fio, utilizando o *chip* ESP8266, mostrado na Figura 1. Existem módulos de diferentes tamanhos e fabricantes. Dentre eles, citam-se os módulos ESP-01 e ESP-12, que se diferenciam em relação ao tamanho e quantidade de entradas e saídas disponíveis para acesso externo [13]. O Módulo WiFi ESP-12E viabiliza a conexão de uma determinada aplicação em uma rede sem fio, possibilitando a troca de informações entre

os microcontroladores com uma rede de computadores ou a Internet. Além disso, esse módulo comunica-se com outros microcontroladores como o Arduino e Raspberry Pi por meio de comunicação serial utilizando os pinos *Tx* e *Rx*, configurados por comandos AT, conhecidos por comandos Hayes [13].

Fig. 1. *Chip* ESP8266.

A programação dos módulos ESP–XX exige adaptadores para conexão via porta seria (USB, *Universal Seria Bus*) e alimentação externa via pino. Portanto, projetaram-se diversas placas de desenvolvimento que integram os módulos ESP–XX como o NodeMCU e os modelos fabricados pela Sparkfun e WeMos, que apresentam adaptadores micro USB para comunicação serial e/ou entradas apropriadas para alimentação externa. A plataforma Node MCU constitui um *firmware* de código aberto para desenvolvimento de aplicações relacionadas à Internet das Coisas (*IoT Internet of Things*). Essa plataforma possui três versões oficiais, como visto na Figura 2. Dentre elas, utiliza-se a versão 2 e, portanto, exploram-se suas características [14]. Esse modelo possui uma arquitetura RISC de 32 bits; processador para operação em 80MHz/160MHz; 4 Mb de memória *flash*; WiFi nativo padrão 802.11b/g/n; operação em modo AP (*Access Point*), *Station* ou ambos; alimentação externa de 5 V através de conector micro USB; onze pinos digitais; um único pino analógico com resolução de 10 bits, operação em nível lógico de 3,3 V; e programação via USB ou WiFi.

(a) (b) (c)

Fig. 2. Versões de NodeMCU (a) Versão 1, (b) versão 2 e (c) Versão 3.

Na Figura 3, indicam-se a pinagem, conexões e componentes da plataforma de desenvolvimento NodeMCU. Destacam-se o módulo ESP-12E, o regulador de tensão de 3,3 V modelo AMS1117 e o conversor USB Serial modelo SILABS CP2102. A placa possui um botão de *Reset* e outro de *flash*, localizados próximo a entrada micro USB. O botão *reset* permite reiniciar a placa, com efeito similar a interromper e reiniciar a alimentação da placa. O botão *flash* provê a gravação de programas no módulo ESP-12E. Realiza-se a alimentação da placa pelo conector micro USB (5 V) ou pelo pino de alimentação externa (V_{in}) com auxílio de um regulador de tensão de 5 V. Deve-se optar por apenas um tipo de alimentação, pois são mutuamente exclusivas. Caso contrário, ocasionam-se danos aos componentes da placa. A placa possui quatro pinos de aterramento (GND) para uso em circuitos externos e também para referência no uso de alimentação externa. Os três pinos de tensão regulada de 3,3 V oferecem alimentação aos circuitos externos e estão conectados à saída do regulador de tensão AMS1117. O pino de *Reset* (RST) permite reiniciar a placa quando se aplica um nível de tensão de 0 V. O pino *Enable* (EN) ativa o módulo ESP-12E quando aplica-se nível de tensão de 3,3 V. O pino identificado por A0 corresponde a entrada do conversor A/D de 10 bits, sendo empregado como entrada analógica com tensão máxima de 1,1 V. Próximo a esse pino, tem-se dois pinos reservados. Os demais pinos correspondem aos pinos digitais programáveis ou GPIO.

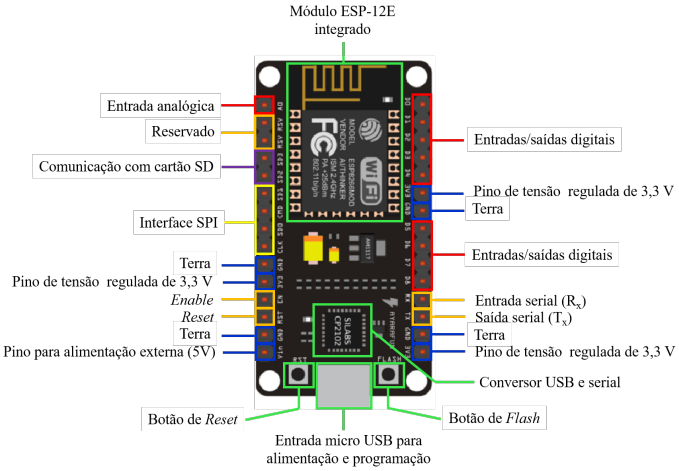

Fig. 3. *Layout* da NodeMCU e indicação de pinagem, conectores e componentes.

Empregam-se os pinos GPIO rotulados D0 a D8 como entradas e saídas digitais. Na Figura 4, relacionam-se os pinos GPIO com os rótulos D0 à D8. Esses pinos, com exceção do D0, possuem interrupção e modulação por largura de pulso (PWM, *pulse width modulation*). O pino digital D0 também permite controlar os modos *sleep's* da placa, relacionados ao consumo de bateria. O pino digital D3 encontra-se conectado ao botão *flash*, sendo utilizado para controle de *upload* de programas na memória *flash*. Utilizam-se os pinos digitais de D5 a D8 em comunicações SPI de alta velocidade.

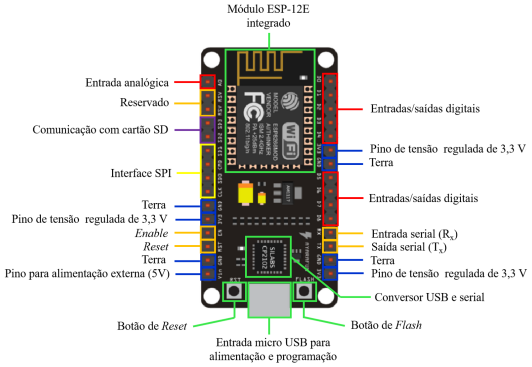

Fig. 4. Identificação dos pinos programáveis.

III. Linguagem MicroPython

A programação em Python surgiu em 1990, sendo desenvolvida por Guido Van Rossum no Centro de Matemática

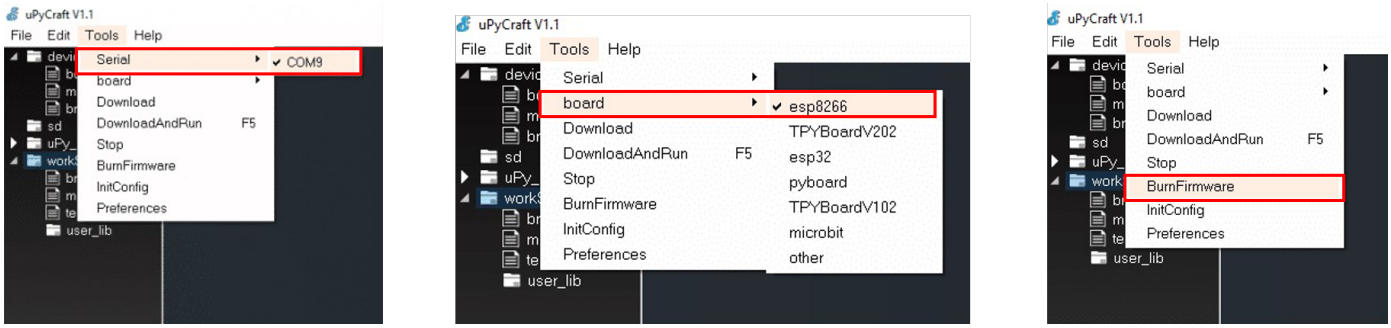

Fig. 5. Primeiros passos para atualização do *firmware* da plataforma NodeMCU para programação em MicroPython por meio da IDE uPyCraft.

Stichting, na Holanda [9]. É uma linguagem de alto nível e orientada a objetos com semântica dinâmica. Popularizou-se devido à sua compatibilidade com a maioria dos sistemas operacionais. Já o MicroPython originou-se dos trabalhos do programador australiano e físico Damien George [10, 15]. Essa linguagem é uma implementação eficiente do Python 3, que inclui um pequeno conjunto da biblioteca padrão otimizadas para programação de microcontroladores.

Na construção da linguagem MicroPython, simplificaram-se as bibliotecas padrão do Python para adequar a linguagem ao propósito. Logo, as novas bibliotecas passaram a conter apenas um subconjunto de funcionalidades quando comparadas à original. Para a nomenclatura, considerou-se o acréscimo da letra 'u' ao início da biblioteca padrão, permitindo manter a similaridade com a linguagem Python e ao mesmo tempo identificar as bibliotecas do MicroPython. Devido à correlação entre as linguagens, restringe-se a abordar apenas as bibliotecas específicas para programação do ESP8266. Caso o leitor não tenha conhecimentos amplos sobre a linguagem Python, indica-se consultar [15]. A linguagem MicroPython possui oito bibliotecas especificas, cujas funcionalidades são:

- **btree**: implementação de bancos de dado usando armazenamento externo e o modelo chave-valor, se assemelhando ao funcionamento do tipo *dict*;
- **framebuf**: geração de um *buffer* de quadro geral para criar e enviar imagens do tipo bitmap para um monitor;
- **machine**: manipulação do *hardware*, controlando diversas partes como pinos GPIO e modos de operação;
- **micropython**: acesso e controle de funcionalidades internas do MicroPython;
- **network**: configuração de conexão da plataforma de *hardware* com à Internet por meio de *drivers* e funções de roteamento;
- **ubluetooth**: controle de interfaces Bluetooth disponíveis na placa e com suporte ao Bluetooth *Low Energy* (BLE);
- **ucryptolib**: implementação de mecanismos de criptografia com três possíveis modos: *electronic code book* (ECB), *cipher block chaining* (CBC) e *counter mode* (CTR);
- **uctypes**: Manipulação de dados binários de maneira estruturada;

Dentre as bibliotecas supracitadas, exploram-se principalmente os métodos e funções da biblioteca **machine** para manipular os pinos de GPIO digitais e analógico e da biblioteca **network** para conectar o NodeMCU em uma rede WiFi. Além disso, empregam-se alguns métodos de bibliotecas padrão como a **time** para prover pausas na execução do código e a **umqtt.simple** para implementar o protocolo MQTT.

IV. Primeiros passos na programação do NodeMCU em MicroPython

Nessa seção, aborda-se a preparação do ambiente de desenvolvimento para programar o NodeMCU em MicroPyhton e também algumas programações básicas como a manipulação de pinos digitais e pino analógico.

A. Preparação do ambiente de desenvolvimento

Para programar o NodeMCU em linguagem MicroPython, torna-se necessário selecionar um ambiente de desenvolvimento integrado (IDE, *integrated development environment*) que implemente o MicroPython, assim como realizar a atualização do *firmware* para interpretar os comandos dessa linguagem. Logo, selecionou-se o IDE uPyCraft que é especificamente projetado para trabalhar com a linguagem MicroPython. Esse IDE é gratuito e de fácil instalação e uso. O uPyCraft oferece suporte para atualizações *on-line*, compatibilidade com os sistemas operacionais Windows, Mac, Linux, suporte *on-line* para atualizações de placas e suportes técnicos pelo Fórum ou Github. Para instalar o uPyCraft, deve-se baixar o instalador (link de sugestão: https://www.embarcados.com.br/micropython-no-esp8266/) correspondente ao sistema operacional da máquina que receberá a instalação.

A atualização do *firmware* da plataforma NodeMCU requer a instalação prévia do uPyCraft e o *download* do arquivo, que pode ser encontrado no site do MicroPython (https://micropython.org/download/esp8266/). Nesse endereço eletrônico, encontram-se diversas opções de placas e opta-se por '*Generic ESP8266 module*', baixando a versão mais recente do *firmware*. Para a atualização, conecta-se a placa à porta serial de um computador que contenha a IDE e o arquivo de *firmware*. No uPyCraft e em '*Tools*'>'*Serial*', seleciona-se a porta serial em que o NodeMCU conectou-se. Em seguida, deve-se optar pelo modelo de placa esp8266 em '*Tools*'>'*board*'. Acessa-se a área de inclusão do *firmware* em '*Tools*'>'*BurnFirmware*'. Indicam-se esses passos na Figura 5.

Na janela de diálogo que se abre (Figura 6), deve-se selecionar a plataforma em '*board*', habilitar a reescrita do *firmware* da placa em '*erase_flash*', selecionar a porta de comunicação serial em que o NodeMCU conectou-se em

'*com*'. No campo '*Firmware Choose*', opta-se por '*Users*' e seleciona-se o arquivo de *firmware* baixado anteriormente. Ao confirma a ação em 'ok', inicia-se o processo de reescrita do *firmware*, exibindo-se o avanço em uma segunda janela de diálogo. Ao finalizar a operação, o NodeMCU está preparado para programação.

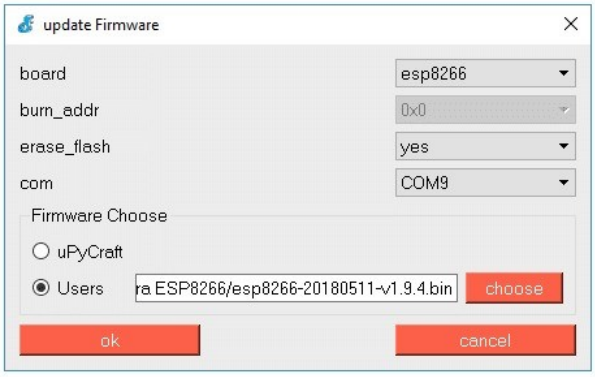

Fig. 6. Configurações para atualizar o *firmware* da plataforma NodeMCU.

B. Manipulando portas digitais

Para utilizar as portas digitais do NodeMCU pela programação em MicroPython, deve-se importar a biblioteca **machine**. Essa biblioteca permite definir o pino ao qual deseja-se trabalhar como entrada ou saída, indicando o pino GPIO e o estado da porta (entrada 'IN' e saída 'OUT'). Ressalta-se que o MicroPython trabalha com a nomenclatura de GPIO apresentada na Figura 4. Para exemplificar, considerou-se o piscar de um LED conectado ao GPIO 0 (correspondente a porta digital D3), como mostrado na Figura 7. Utilizou-se uma função temporizadora *sleep*( ) para prover o efeito de piscar. Definiu-se o estado do LED por meio do método *value*( ).

```
# Manipulando portas digitais

#importando bibliotecas:
from machine import Pin
from time import sleep

led = Pin(0, Pin.OUT) # definindo o pino

led.value(1) # Estado do led como aceso
sleep(3.0)   # paralisa a execucao por 3 segundo

led.value(0) # Estado do led como apagado
sleep(3.0)   # paralisa a execucao por 3 segundo

led.value(1) # Estado do led como aceso
sleep(3.0)   # paralisa a execucao por 3 segundo

led.value(0) # Estado do led como apagado
sleep(3.0)   # paralisa a execucao por 3 segundo
```

Nota-se que a uPyCraft não possui as funções *setup*( ) e *loop*( ), características da IDE do Arduíno. Entretanto, pode-se implementar a função *loop*( ) por meio de uma estrutura de repetição *while* com laço infinito.

C. Comunicação UART

A IDE uPyCraft não possui um espaço para interação entre a plataforma e o usuário como ocorre na IDE do Arduíno

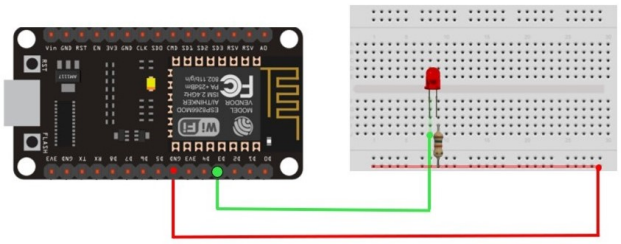

Fig. 7. Circuito para exemplificar a manipulação de portas digitais do NodeMCU pela programação em MicroPython.

por meio da função 'Monitor Serial'. Entretanto, pode-se programar o NodeMCU para estabelecer comunicação com uma plataforma Arduino para utilizar essa funcionalidade. Explora-se a comunicação universal assíncrona transmissor-receptor ou *universal asynchronous receiver-transmitte* (UART) entre as plataformas por meio de comandos uart inclusos na biblioteca **machine**. Utiliza-se o mesmo exemplo da Figura 7 e comandos enviados pelo 'Monitor Serial' da IDE do Arduino para comandar o estado do LED:

```
# Comunicacao entre NodeMCU e Arduino

#importando bibliotecas:
from machine import Pin, UART

uart = UART (0, 115200) #definindo a taxa de
    comunicacao
led = Pin(0, Pin.OUT)   # definindo o pino


ch = b'' #declaracao da variavel em bits

#loop infinito:
while True:

    if uart.any() > 0:   #presenca de dados
        ch = uart.readline() # Le o comando

        #Compara o comando para controlar o LED
        if ch == b'on':
            uart.write('Led On! \n')
            led.off()
        elif ch == b'off':
            uart.write('Led Off! \n')
            led.on()
        else:
            uart.write('COmando inv lido')
```

Para que a comunicação funcione corretamente, deve-se habilitar uma linha de código de "boot" e fazer o *download*: uos.dupterm(None, 1). Configura-se um segundo arquivo para o "boot":

```
# Configuracao de boot

import uos,machine

uos.dupterm(None, 1)

import gc
```

Na IDE uPyCraft, executa-se os códigos e desconecta-se a placa da IDE. Já na IDE do Arduíno, seleciona-se a porta em que a placa está conectada. Em seguida, abre-se o monitor serial e interage-se com o NodeMCU.

D. Uso da porta analógica

Como discutido na Seção II, o NodeMCU possui apenas uma única porta analógica. Emprega-se essa porta para a conexão de sensores diversos como sensor de luminosidade, sensor de umidade, etc. Os comandos necessários para manipular essa porta também se encontram na biblioteca **machine**. Para exemplificar o uso da porta analógica, considerou-se um sensor de luminosidade do tipo LDR (*light dependent resistor*), visto na Figura 8.

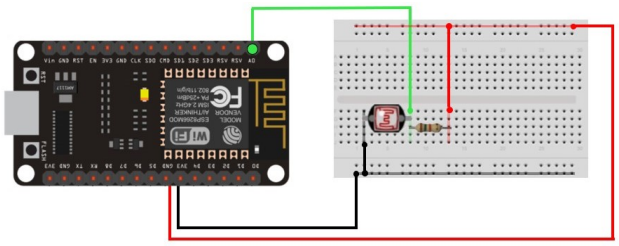

Fig. 8. Circuito para exemplificar o uso da porta analógica do NodeMCU pela programação em MicroPython.

Implementou-se o código a seguir para ler o valor de luminosidade captado pelo sensor e imprimi-lo em intervalos de 5 segundos:

```
# Manipulando a porta analogica

#importando bibliotecas:
from machine import ADC
import time

adc = ADC(0) #define o pino analogico

#loop infinito:

while True:
    print('Valor de luminosidade:', adc.read())
    time.sleep(0.5) #atraso de 5 segundos
```

Com o intuito de prover amplo conhecimento sobre os tipos de sensores. Explorou-se uma gama de sensores e os incluiu no caderno desenvolvido pelos autores. Nesse trabalho, restringiu-se apenas à apresentação de um exemplo com o objetivo de demonstrar a manipulação da porta analógica.

V. Conexão com a Internet e Implementação do Protocolo MQTT

Nessa seção, demonstra-se como empregar a biblioteca **network** para conectar o NodeMCU em uma rede WiFi para estabelecer comunicação com a plataforma ThingSpeak por meio do protocolo MQTT implementado pela biblioteca **umqtt.simple**.

A. Conectando o NodeMCU à uma rede WiFi

Por meio da biblioteca **network**, pode-se configurar o NodeMCU para atuar como um ponto de acesso (AP), como uma estação (STA) ou como ambos (AP/STA). Nesse trabalho, explora-se a programação do NodeMCU para operar como uma estação que se conecta em um AP para conexão com à Internet. Inicialmente, importa-se a biblioteca e instancia-se um objeto para acesso a seus métodos e funções. Em seguida, implementa-se uma estrutura de decisão para verificar o estado da conexão. Se não conectado, o NodeMCU tenta estabelecer a conexão com a rede WiFi desejada. A seguir, tem-se as linhas de códigos para realizar o processo. Estruturou-se o código em uma função por praticidade:

```
# Conexao com rede WiFi

#importando bibliotecas:
import network

#funcao para conexao

def conectar():
    #instnciando o objeto
    sta_if = network.WLAN(network.STA_IF)
    #verifica a conexao
    if not sta_if.isconnected():
        print('Conectando...')
        sta_if.active(True)
        #conecta com a rede especificada:
        sta_if.connect('<Nome da rede>', '<Senha>')

        while not sta_if.isconnected():
            pass

#loop infinito:
while True:
    conectar()
    #demais comandos
```

B. Comunicação com o protocolo MQTT

Para exemplificar uma comunicação por meio do protocolo MQTT, conectou-se o NodeMCU em uma rede WiFi e estabeleceu-se comunicação com a API do ThingSPeak para enviar o estado de um LED a cada intervalo de tempo. O Thingspeak é uma aplicação utilizada para armazenar e recuperar dados por meio de APIs REST e MQTT [12]. Para utilizar a plataforma Thingspeak, deve-se criar um usuário e um canal. No canal, recuperam-se o ID e a API de gravação para uso na programação do NodeMCU, como indicado na Figura 9.

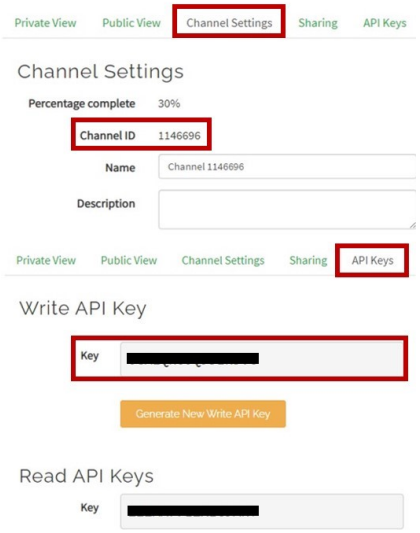

Fig. 9. ID e API de gravação do canal criado no ThingSpeak.

No uPyCraft, importou-se as bibliotecas **machine** para manipular portas digitais, **time** para impor atrasos na execução do código e **umqtt.simple** para a comunicação com ThingSpeak. Em seguida, configurou-se o servidor, instanciou-se um objeto

cliente e explicitaram-se o ID e chave do canal (veja Figura 9). Estruturou-se o tópico para publicação na linha 23 e a mensagem na linha 25. Conectou-se ao ThingSpeak por meio do método *connect*( ), publicou-se a mensagem no devido tópico usando o método *publish*( ) e finalizou-se a conexão com o método *disconnect*( ).

```
# Conexao com ThinSpeak pelo protocolo MQTT

#importando bibliotecas:
from machine import Pin
from time import sleep
from umqtt.simple import MQTTClient

SERVER = 'mqtt.thingspeak.com'
CHANNEL_ID = '<ID do canal>'
WRITE_API_KEY = '<Chave>'

#instanciando o objeto cliente:
cliente = MQTTClient('umqtt_client', SERVER)
led = Pin(16, Pin.OUT)
atraso = 1
inverte = 0

while True:

    led.value(atraso)
    sleep(2.0)
    #definindo o topico:
    topic = 'channels/' + CHANNEL_ID + 'publish/' +
    WRITE_API_KEY
    #definindo o conteudo:
    payload = 'field1 =' +str(led.value())

    client.connect()
    cleint.publish(topic, payload)
    client.disconnect()
    atraso, inverte = inverte, atraso
```

Na Figura 10, verificou-se o estado do LED (Aceso '1' e apagado '0') em um *dashborad* da plataforma ThingSpeak em tempo real.

Fig. 10. Gráfico gerado pelo ThingSpeak.

VI. Conclusão

Esse trabalho apresentou um breve tutorial sobre o uso da linguagem MicroPython para a programação de microcontroladores ESP8266, representando um resumo de um caderno tutorial desenvolvido pelos autores. A linguagem MicroPython pode colaborar para o fácil aprendizado de conceitos básicos em eletrônica devido à simplicidade da linguagem, característica da linguagem Pyhton. Contudo, concluiu-se que o emprego do MicroPython para programação de plataformas *maker* no meio acadêmico é algo recente e pouco difundido devido à consolidação de outras linguagens como o C/C++. Isso está aliado à qualidade dos IDE empregados para a programação. Ao comparar o uPyCraft com o IDE do Arduino, nota-se a ausência de algumas funcionalidades e facilidades que tornam o aprendizado fácil. Encontraram-se referências à outros IDE's para programação em MicroPython tanto do NodeMCu quanto de outras plataformas. Portanto, propõem-se como trabalhos futuros explorar esses IDE's e compará-los em termos de suas funcionalidades, assim como explorar outras plataformas com a programação em MicroPython.

Referências

[1] Simone Cirani, Marco Picone, Luca Veltri e Gianluigi Ferrari. *Internet of Things: Architectures, Protocols and Standards*. 1th. John Wiley Sons Ltd, 2019.

[2] Ammar Rayes e Samer Salam. *Internet of Things from Hype to Reality: The Road to Digitization*. 2th. Springer, 2019.

[3] B. K. Tripathy e J. Anuradha. *Internet of Things (IoT): Technologies, Applications, Challenges, and Solutions*. 1th. CRC Press, 2018.

[4] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari e M. Ayyash. "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications". Em: *IEEE Communications Surveys Tutorials* 17.4 (2015), pp. 2347–2376.

[5] Gaston C. Hillar. *MQTT essentials : a lightweight IoT protocol : the preferred IoT publish-subscribe lightweight messaging protocol*. 1th. Packt, 2017.

[6] Gaston C. Hillar. *Hands-On MQTT Programming with Python: Work with the lightweight IoT protocol in Python*. 1th. Packt, 2018.

[7] Arduino. *What is Arduino?* URL: https://www.arduino.cc/en/Guide/Introduction (acesso em 03/01/2021).

[8] G. Bauermeister. *Guia do Usuário do ESP8266*. URL: https://www.filipeflop.com/blog/guia-do-usuario-do-esp8266/ (acesso em 03/01/2021).

[9] M. L. Hetland. *Python Algorithms Mastering Basic Algorithms in the Python Language*. Apress, 2010.

[10] C. Bell. *MicroPython for the Internet of Things: A Beginner's Guide to Programming with Python on Microcontrollers*. Apress, 2017.

[11] A. Kurniawan. *MicroPython for ESP8266 Development Workshop*. 2016.

[12] ThingSpeak. *ThingSpeak for IoT Projects*. URL: https://thingspeak.com/ (acesso em 03/01/2021).

[13] Vida de Silicio. *Módulo Wifi ESP8266 12E*. URL: https://www.vidadesilicio.com.br/modulo-wifi-esp8266-12e (acesso em 23/11/2020).

[14] Greici Oliveira. *NodeMCU – Uma plataforma com características singulares para o seu projeto ioT*. URL: https://blogmasterwalkershop.com.br/embarcados/nodemcu/nodemcu-uma-plataforma-com-caracteristicas-singulares-para-o-seu-projeto-iot/ (acesso em 23/11/2020).

[15] MicroPython. *MicroPython*. URL: https://micropython.org/ (acesso em 23/11/2020).