

SISTEMA EMBARCADO PARA REGISTRO E TELEMETRIA DE DADOS

Autor: Victor Pierobon F. dos Passos (FHOUniararas) victorpassos10@gmail.com

Orientador: Mauricio Acconcia Dias (FHOUniararas) macdias@fho.edu.br

Resumo

A transferência de informação entre pontos distantes, é um dos desafios das áreas da computação e sofre crescente procura por parte desenvolvedores e empresas, que buscam soluções diversificadas para atender a demanda desse tipo de tecnologia.

Neste artigo, é proposta uma solução de telemetria de dados, provenientes de sensores digitais e analógicos de pressão atmosférica, altitude, temperatura e pressão diferencial. Esses, possibilitam o monitoramento de processos, como o voo de VANTs (Veículos Aéreos Não Tripulados) bem como a coleta de dados para projetistas.

São estabelecidas três etapas para o estudo, onde inicialmente são observadas tecnologias de hardware capazes de suportar diferentes tipos de sensores, para criação de um servidor de dados. Após, é determinado uma solução para a transmissão dos dados, utilizando características de projetos de IoT. A terceira etapa, foi o desenvolvimento da programação dos sensores ligados ao servidor de dados e da comunicação escolhida, para se comunicar com um cliente que seja capaz de controlar as informações recebidas em um computador distante.

Foi possível desenvolver a solução proposta, com alcance aproximado de 150 metros, utilizando uma estrutura de hardware de baixo custo, com efetiva transferência de dados.

Assim, este trabalho contribui para áreas de estudo em telemetria de dados e projetos relacionados a internet das coisas, possibilitando a escalabilidade com acréscimo de sensores que se julgar necessários para cada situação de aplicação.

Palavras-Chaves: (Sistemas embarcados, telemetria de dados, IoT)

1. Introdução

Nas últimas décadas, a tecnologia vem mudando a um ritmo cada vez mais intenso e a pressão para desenvolver novos produtos, melhorar o desempenho de sistemas existentes e criar novos mercados, apenas acelera esse ritmo. Novas maneiras de armazenar informações, construir circuitos integrados e desenvolver hardwares que contenham componentes de software, que possam “pensar” sozinhos com base na entrada de dados, são apenas algumas possibilidades (BOYLESTAD, 2012). Um sistema embarcado, é um sistema no qual o computador é dedicado ao dispositivo ou sistema que ele monitora e ou controla. Diferente de computadores de propósito geral, um sistema embarcado realiza um conjunto de tarefas predefinidas, geralmente com requisitos específicos como: Otimizar o projeto, reduzir tamanho, recursos computacionais e custo do produto (ASSEF, 2017).

A era de tecnologias desenvolvidas com foco em IoT (Internet of Things – Internet das coisas) é assunto presente em áreas ligadas a engenharia, tecnologia e informação, com a procura por implementações de novas soluções para problemas de diferentes áreas. Ao comentar sobre o assunto, Venkatech (2011) diz que é crescente a procura por tecnologias que realizem a capture dados provenientes de sensores distantes e transmita eles para o local de análise, havendo uma tendência crescente em direção ao padrão IEEE 802.11 baseado em TCP/IP. Myers (2013) discorre sobre o assunto, observando que o mercado oferece uma boa oportunidade para criar produtos que beneficiam processos com o uso de tecnologias IoT e que até 2020, serão mais de 30 bilhões de dispositivos conectados a redes de comunicação sem fio.

Plataformas de rápido desenvolvimento de prototipagem, facilitam o início dos projetos neste tema e são fortes aliadas dos projetistas e dos desenvolvedores neste cenário. Módulos de comunicação com microcontrolados pré-certificados são fundamentais e estes devem ser facilmente aplicados ao projeto, configurados e adicionados a rede (MYERS, 2013). Como resultado disso, desenvolvedores não precisam se preocupar com a complexidade de desenvolver seus próprios sistemas de hardware sem fio para aplicações de IoT. Os módulos de Wi-Fi incorporam microcontroladores, que além de suportar os protocolos de comunicação padrão 802.11, também tem conectividade para troca de dados via comunicação UART

(Universal asynchronous receiver/transmitter), SPI (Serial peripheral Interface), I2C (Inter Integrated Circuit) e SDIO (Secure Digital Input Output) (VOKAS, 2015).

Neste cenário e com o propósito da obtenção de uma arquitetura de hardware e software para compor um sistema de telemetria de dados, busca-se em dois primeiros momentos, soluções de hardware e software embarcado que sejam práticas e com bom uso de recursos disponíveis para prototipação. Devem possibilitar a comunicação diferentes tipos de sensores periféricos, com foco para soluções da nova era com dispositivos conectados via Wi-Fi. Em uma terceira etapa, o enfoque é em desenvolver a estrutura de software básica que permita testes e a utilização da arquitetura de maneira que possibilite a escalabilidade do projeto.

2. Metodologia

A pesquisa foi dimensionada em três etapas, que possibilitam uma visão modular da arquitetura de hardware e do software, a fim de proporcionar escalabilidade ao projeto mantendo o baixo custo e fácil reprodução do mesmo. Esses atributos, são altamente buscados no desenvolvimento de projetos de produtos e aplicações em áreas de pesquisas atuais de engenharia, computação e automação; caracterizadas pelos termos: sistemas embarcados, telemetria de dados, indústria 4.0 e IoT (Internet of Things – Internet das coisas).

Cada uma das etapas, possui um objetivo específico e são divididas em:

1. Pesquisa e definição do hardware: Interface microcontrolada para realizar a aquisição de dados, que são provenientes de sensores de baixo custo, porém boa confiabilidade e rápida prototipação.
2. Pesquisa e definição do tipo de comunicação a ser aplicada no projeto: Foco em soluções voltadas para IoT, para desenvolvimento do envio das informações até um receptor distante, que pode ser embarcado em um VANT ou aplicação de sensoriamento remoto.
3. Desenvolvimento e testes das aplicações dos componentes de hardware e software, que contemplam os itens 1 e 2.

2.1 Pesquisa e definição do hardware

Para compor os elementos hardware, foram observadas as plataformas de prototipação microcontroladas, que permitem um alto grau de escalabilidade, a fim de proporcionar aplicabilidade em diversas áreas como: em VANTs (Veículos Aéreos Não Tripulados) e projetos voltados para indústria 4.0 e IoT (Internet of Things – Internet das coisas).

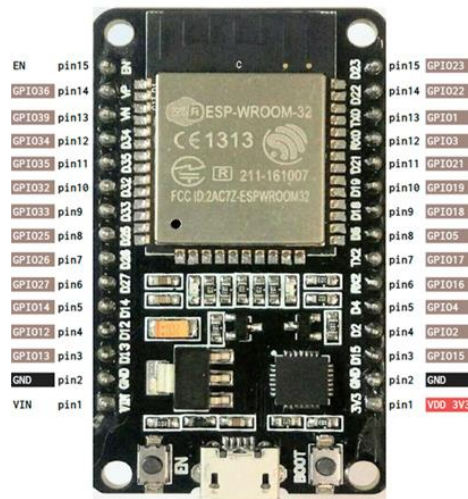
Aplicações deste tipo, tem em comum a necessidade requerer baixo consumo de energia e peso reduzido. São desejáveis plataformas com microcontroladores e ou microprocessadores versáteis, com entradas e saídas de sinais analógicos e digitais, e barramentos de comunicação com funcionalidades voltadas para diferentes tipos de protocolos. Foram consideradas e comparadas, diferentes plataformas para a finalidade proposta, sendo elas:

Quadro 1 – Comparação do hardware embarcado

Plataforma	Microncontrolador ou Microprocessador	I/O digital	I/O analógico	Comunicação nativa	Alimentação mínima externa	Dimensão e peso
Raspberry Pi B	ARM Cortex-A53, 64bits, 1.2GHz – 4 núcleos.	40 pinos, 5volts.	Não possui.	USB, I2C, SPI, UART, Ethernet, WiFi e bluetooth.	5V – 2.5A	85x56x17mm. Aprox. 45g
Arduino uno	ATMEGA328, 8bits, 20MHz.	14 pinos, 5volts.	6 pinos para 5volts, conversor A/D de 10bits.	USB, I2C, SPI e UART.	9V – 600mA	68x53x10mm. Aprox.40g
ESP32 devkit v1	TENSILICA XTENSA LX6, 32bits, 240MHz – 2 núcleos.	25 pinos, 3.3volts.	6 pinos para 3.3volts, conversor A/D de 12bits.	USB, I2C, SPI, UART, WiFi, Bluetooth.	5V – 600mA	52x28x5mm. Aprox.20g

Foi escolhida a plataforma ESP32 Devkit v1, pois: Em relação as demais plataformas observadas para aplicações e estudos na área, ela se mostra versátil por ter uma boa gama protocolos de comunicações nativa e não dependendo de módulos adicionais para compor um sistema amplo. Também por possuir boa resolução na conversão A/D e grande disponibilidade de I/O, além de ter tamanho e peso reduzidos.

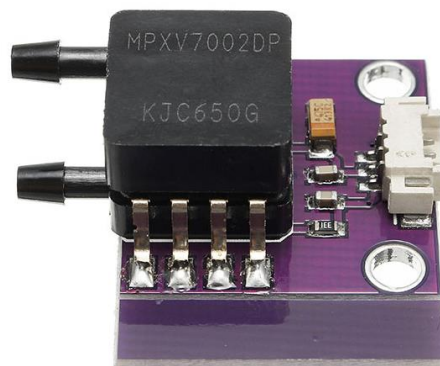
Figura 1 – ESP32 DEVKIT V1



Fonte 1 – Adaptado de: playelek.com

Utilizando programação em linguagem C/C++, foi desenvolvido um código teste inicial para conhecer a plataforma de hardware, realizando a leitura de um sensor de pressão diferencial analógico, modelo MPXV7002DP conectado ao GPIO 36 da placa de desenvolvimento, exibindo os resultados via comunicação serial com o computador.

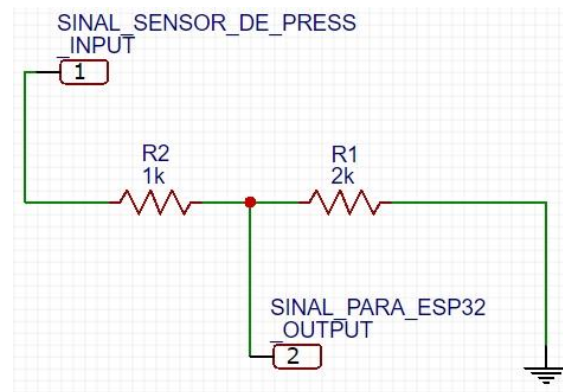
Figura 2 – Sensor analógico, de pressão diferencial



Por ser analógico e alimentado com 5vcc, a tensão máxima que o sensor emite quando submetido a sua pressão máxima (2KPa) é próximo a tensão de alimentação. Foi

necessário realizar uma associação de resistores, para compor um divisor de tensão. Desta maneira, quando o sensor for submetido a pressões próximas a máxima, sua tensão de resposta não ultrapassa os 3.3volts de operação, sendo a tensão máxima suportada pela plataforma ESP32 Devkit v1.

Figura 3 – Associação de resistores para conversão de tensão



O sensor BMP280 apresenta boa confiabilidade e estabilidade de resultados para ensaios de pressão atmosférica, altitude e temperatura. Sua comunicação com o ESP32 Devkit v1, é realizada via padrão I2C utilizando pinos de GPIO 22 e 21 (para SCL e SDA). A resposta do sensor para os dados propostos, é apresentado nas escalas: Pascal, metros e graus Celsius, para os itens mencionados respectivamente.

Figura 4 – Sensor BMP280

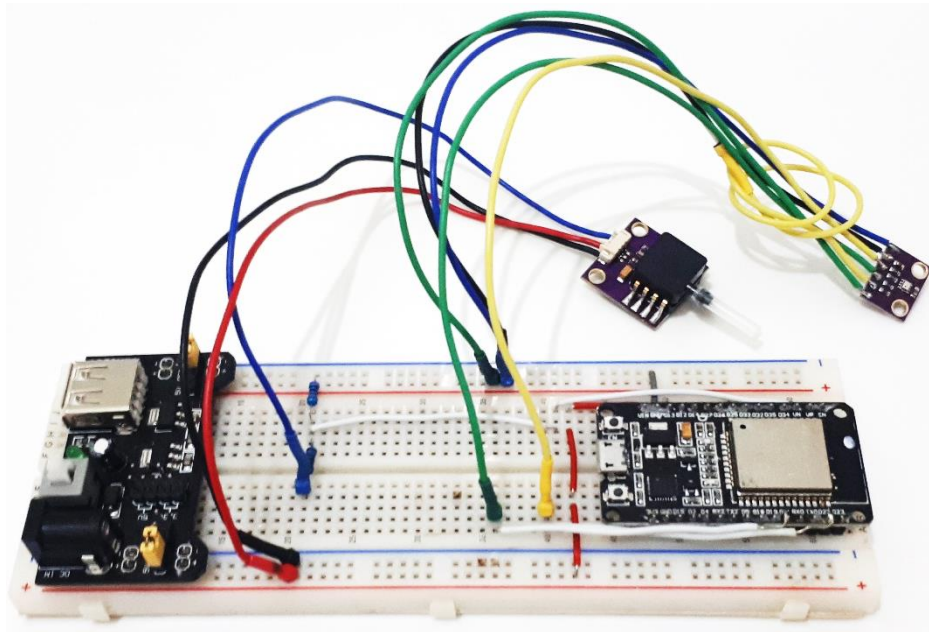


A implementação final, conta com a utilização de uma protoboard e de uma fonte de alimentação externa de 9vcc – 800mA, que alimenta todo o conjunto de componentes, formado por:

- 1 módulo de reguladores de tensão, responsável por fornecer a tensões de 3.3Vcc e 5Vcc;
- 1 módulo sensor de pressão diferencial MPXV7002DP;
- 1 módulo de pressão atmosférica, altitude e temperatura BMP280;

- associação de resistores, para queda de tensão do sinal do sensor MPXV7002DP;
- módulo ESP32 Devkit v1.

Figura 5 – Implementação final do conjunto de componentes

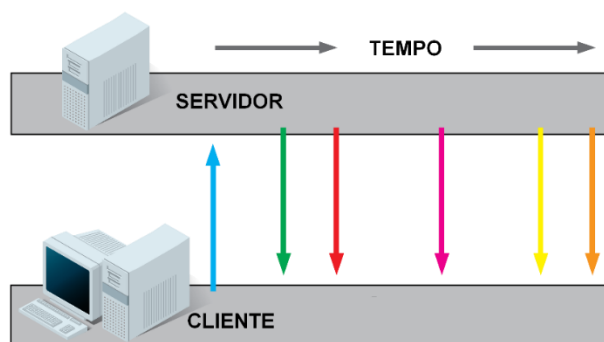


2.2 Pesquisa e definição do tipo de comunicação

A escolha da plataforma do hardware, permite a implementação de uma comunicação utilizando redes WiFi e protocolos para esse tipo de aplicação. Além de ser tendência observada na área da computação, para troca de dados e informações, essa abordagem permite uma solução alternativa para o monitoramento de sensores de um VANT ou demais aplicações de monitoramento, assim como a escalabilidade do projeto.

Para prover as informações de sensores por meio do hardware escolhido, é necessária uma aplicação de servidor web com comunicação utilizando a tecnologia de websocket, que é implementado sobre sockets de TCP/IP. A aplicação de um servidor com websocket, possibilita a comunicação full-duplex entre o hardware servidor e o cliente, que acessa ou recebe as informações em tempo real sempre que um novo dado estiver disponível no servidor.

Figura 6 – Websocket entre servidor e cliente



Fonte 2 – Adaptado de: os.alfajango.com

2.3 Desenvolvimento e testes das aplicações

Definidos o hardware e a comunicação a serem utilizados, foi programado utilizando a linguagem C/C++, um servidor web com acesso via endereço IP, para obtenção das informações do ESP32 via comunicação websocket. Bibliotecas de funções proprietárias de WiFi e websocket que suportam o hardware escolhido, ainda não existem por completo; assim, são utilizadas versões anteriores disponíveis para outras plataformas de funcionamento semelhante e são realizadas modificações para não ocorrer conflito entre elas, no mento da compilação e upload para o ESP32 devkit v1.

Para criar um cliente que acessa o servidor via websocket, foram escolhidos ambientes de desenvolvimento de livre distribuição e forte uso por parte da comunidade de desenvolvedores da mesma área, garantindo a disponibilidade ao acesso de atualizações e novos incrementos. Assim, a linguagem Python 3 foi utilizada para o a programação do cliente receptor, pois possui recursos objetivos para o desenvolvimento da comunicação utilizando websocket, manipulação de arquivos e criação de interface gráfica para uso da aplicação.

Testes iniciais foram realizados, para que os dados obtidos pelo sensor de pressão diferencial, fossem enviados via rede WiFi até um computador distante, que armazena as informações em um arquivo tabelado. Duas diferentes arquiteturas de hardware e software foram estabelecidas, para testes de alcance do sinal e obtenção das informações, sendo elas:

1. Na Figura 7, encontra-se a estrutura utilizada para compor a primeira etapa de testes, que utiliza o hardware ESP32 devkit v1 como servidor de dados web em modo AP (Access Point). Neste, a própria plataforma oferece uma rede de conexão, para estabelecer a comunicação via websocket entre o próprio servidor e o computador cliente.

Figura 7 – Utilização do ESP32 como Access Point



2. A Figura 8, exemplifica a estrutura da segunda etapa de testes. A plataforma ESP32 devkit v1 atua como servidor dos dados, conectado em uma rede wireless fornecida por um roteador wireless de uso comumente residencial. O cliente acessa essa mesma rede, para iniciar a comunicação com o websocket do servidor e assim receber as informações que requisita.

Figura 8 – Utilização de roteador para compor a rede



O desenvolvimento da programação que compõem a o servidor de dados e comunicação, é de fácil implementação, modificação e reprodução, com funcionamento em repetições, sendo exemplificado no algoritmo da Figura 9. O

algoritmo utilizado no cliente da aplicação, que recebe e armazena as informações dos sensores, é representado pela Figura 10.

Figura 9 – Algoritmo utilizado no servidor

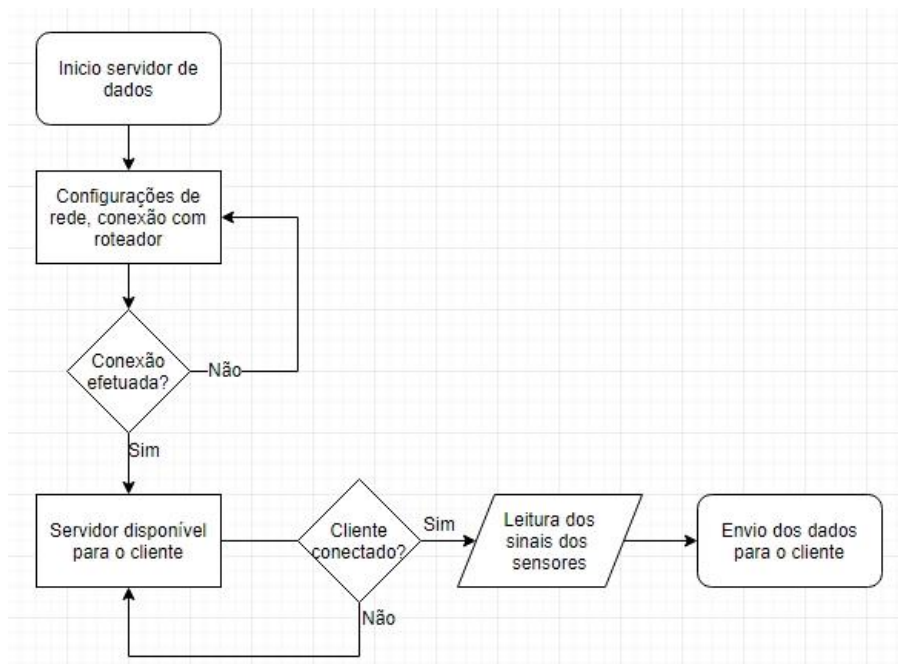
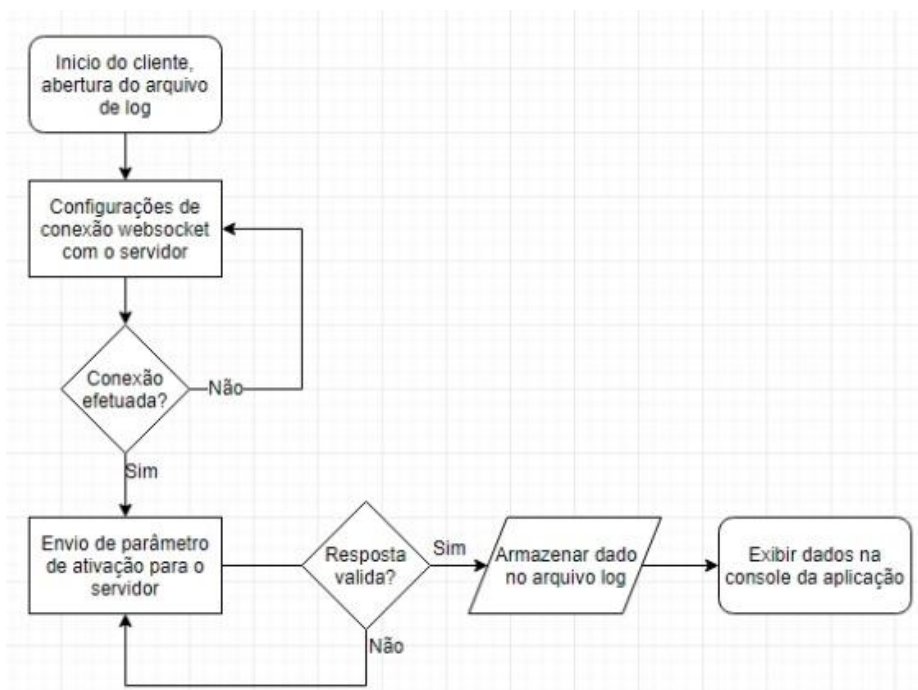


Figura 10 – Algoritmo utilizado no computador cliente



4. Resultados

Utilizando a arquitetura propostas para as duas etapas de testes, foi possível observar que a comunicação se mantém estável em distâncias de variam de 50 até 120 metros. Na segunda arquitetura de testes, foi observado que: com a utilização de um roteador wireless, é possível cobrir uma distância maior entre o ponto em que o servidor se localiza com os sensores em relação ao computador cliente. Um aumentando em 30% (aproximadamente 36 metros) da distância de cobertura do sinal, em relação à primeira etapa.

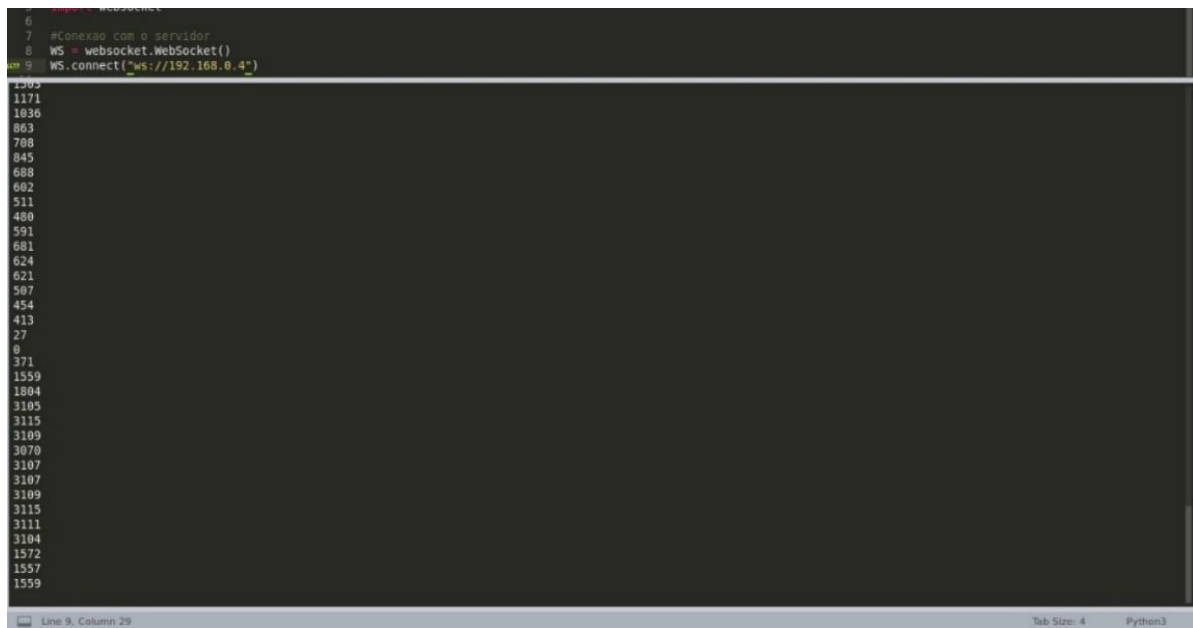
A utilização de bibliotecas não específicas para o hardware e comunicação em questão, não apresentam muitas restrições, problemas ou problemas de comunicação wireless visto em teste na Figura 11. As modificações realizadas em bibliotecas utilizadas para a aplicação que resultam na criação do servidor de dados web e utilização de websocket para comunicação, permitem construir um código prático e com bom funcionamento, mesmo que seu suporte seja para um hardware anterior, ao utilizado neste artigo.

Ocorre uma diferença de 50 milissegundos entre: requisitar, receber e armazenar uma nova informação no cliente, utilizando apenas o sensor analógico (MPXV7002DP) como fonte de dados, teste realizado na Figura 12. Quando inseridos a troca de dados do sensor BMP280, configurando então mais 3 tipos de informações diferentes para a troca de mensagens apresenta, totalizando 4 tipos diferentes dados; foi observado um atraso em média de 200 milissegundos entre o ato de requisitar, receber e armazenar as informações no cliente. O arquivo de saída após utilização da arquitetura contendo os 4 tipos de dados diferentes, armazenou os dados brutos enviados pelos sensores, de maneira como visto nas Figuras 13, 14, 15 e 16.

Figura 11 – Sucesso na conexão com roteador wireless e IP disponível para comunicação websocket

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1496
load:0x40078000,len:8596
load:0x40080400,len:6980
entry 0x400806f4
.....
Servidor OK. Roteador OK. ON!
IP:
192.168.0.4
```

Figura 12 – Conexão websocket realizada pelo cliente e dados brutos analógicos em teste



```
6 #Conexao com o servidor
7 WS = websocket.WebSocket()
8 WS.connect("ws://192.168.0.4")
9
1202
1171
1036
863
708
845
688
602
511
480
591
681
624
621
507
454
413
27
0
371
1559
1004
3105
3115
3109
3070
3107
3107
3109
3115
3111
3104
1572
1557
1559
```

Line 9, Column 29 Tab Size: 4 Python3

Figura 13 – Dados armazenados na saída de arquivos tabelados

	A	B	C	D	E	F
1	ANLG PITOT	ALTITUDE	PRESS	TEMPERATURE	TIME	
2	1671	566.04	94707.94	23.91	PyQt5.QtCore.QTime(23, 34, 37, 76)	
3	1660	565.91	94709.89	23.90	PyQt5.QtCore.QTime(23, 34, 37, 283)	
4	1662	566.17	94708.27	23.91	PyQt5.QtCore.QTime(23, 34, 37, 491)	
5	1659	566.01	94711.33	23.90	PyQt5.QtCore.QTime(23, 34, 37, 699)	
6	1657	566.05	94708.92	23.91	PyQt5.QtCore.QTime(23, 34, 37, 907)	
7	1663	565.85	94710.38	23.90	PyQt5.QtCore.QTime(23, 34, 38, 115)	
8	1674	565.89	94711.36	23.90	PyQt5.QtCore.QTime(23, 34, 38, 323)	
9	1668	565.98	94708.27	23.91	PyQt5.QtCore.QTime(23, 34, 38, 531)	
10	1667	565.76	94710.05	23.91	PyQt5.QtCore.QTime(23, 34, 38, 739)	
11	1661	565.90	94711.00	23.90	PyQt5.QtCore.QTime(23, 34, 38, 947)	
12	1648	566.21	94709.70	23.90	PyQt5.QtCore.QTime(23, 34, 39, 155)	
13	1663	565.87	94710.87	23.90	PyQt5.QtCore.QTime(23, 34, 39, 363)	
14	1653	565.87	94710.55	23.90	PyQt5.QtCore.QTime(23, 34, 39, 576)	
15	1650	566.01	94708.72	23.90	PyQt5.QtCore.QTime(23, 34, 39, 784)	
16	1647	565.94	94709.53	23.91	PyQt5.QtCore.QTime(23, 34, 39, 991)	
17	1644	566.10	94707.45	23.90	PyQt5.QtCore.QTime(23, 34, 40, 199)	
18	1635	565.73	94706.64	23.91	PyQt5.QtCore.QTime(23, 34, 40, 407)	
19	1648	566.09	94710.34	23.91	PyQt5.QtCore.QTime(23, 34, 40, 615)	
20	1648	566.24	94709.98	23.92	PyQt5.QtCore.QTime(23, 34, 40, 823)	
21	1653	565.96	94709.01	23.91	PyQt5.QtCore.QTime(23, 34, 41, 31)	
22	1648	565.96	94709.01	23.92	PyQt5.QtCore.QTime(23, 34, 41, 239)	
23	1851	565.93	94709.82	23.92	PyQt5.QtCore.QTime(23, 34, 41, 447)	
24	2363	565.80	94709.82	23.92	PyQt5.QtCore.QTime(23, 34, 41, 655)	
25	1693	565.77	94709.45	23.94	PyQt5.QtCore.QTime(23, 34, 41, 863)	
26	1647	565.73	94709.63	23.93	PyQt5.QtCore.QTime(23, 34, 42, 71)	
27	2288	566.01	94709.91	23.94	PyQt5.QtCore.QTime(23, 34, 42, 279)	
28	2831	565.78	94712.19	23.95	PyQt5.QtCore.QTime(23, 34, 42, 487)	
29	3184	565.89	94710.41	23.94	PyQt5.QtCore.QTime(23, 34, 42, 695)	
30	1648	565.93	94708.97	23.94	PyQt5.QtCore.QTime(23, 34, 42, 903)	

Figura 14 – Dados armazenados na saída de arquivos tabelados

	A	B	C	D	E	F
31	1649	565.99	94710.72	23.94	PyQt5.QtCore.QTime(23, 34, 43, 111)	
32	1649	565.88	94711.37	23.95	PyQt5.QtCore.QTime(23, 34, 43, 327)	
33	1659	565.83	94712.64	23.94	PyQt5.QtCore.QTime(23, 34, 43, 535)	
34	3070	565.84	94709.42	23.95	PyQt5.QtCore.QTime(23, 34, 43, 743)	
35	2935	565.91	94711.02	23.94	PyQt5.QtCore.QTime(23, 34, 43, 951)	
36	1651	565.64	94711.50	23.95	PyQt5.QtCore.QTime(23, 34, 44, 159)	
37	1647	565.94	94711.34	23.95	PyQt5.QtCore.QTime(23, 34, 44, 367)	
38	2455	565.83	94707.44	23.95	PyQt5.QtCore.QTime(23, 34, 44, 575)	
39	3168	565.71	94709.88	23.95	PyQt5.QtCore.QTime(23, 34, 44, 783)	
40	1710	565.98	94711.34	23.95	PyQt5.QtCore.QTime(23, 34, 44, 991)	
41	1211	565.64	94715.20	23.96	PyQt5.QtCore.QTime(23, 34, 45, 199)	
42	1380	565.73	94710.69	23.96	PyQt5.QtCore.QTime(23, 34, 45, 407)	
43	1682	565.64	94714.36	23.96	PyQt5.QtCore.QTime(23, 34, 45, 615)	
44	2363	565.74	94714.23	23.96	PyQt5.QtCore.QTime(23, 34, 45, 825)	
45	1075	565.69	94710.45	23.96	PyQt5.QtCore.QTime(23, 34, 46, 31)	
46	684	565.79	94709.16	23.97	PyQt5.QtCore.QTime(23, 34, 46, 239)	
47	627	565.82	94710.94	23.97	PyQt5.QtCore.QTime(23, 34, 46, 447)	
48	2203	565.78	94711.76	23.97	PyQt5.QtCore.QTime(23, 34, 46, 655)	
49	2652	565.64	94710.62	23.98	PyQt5.QtCore.QTime(23, 34, 46, 863)	
50	2855	565.76	94709.97	23.98	PyQt5.QtCore.QTime(23, 34, 47, 71)	
51	926	565.85	94712.53	23.98	PyQt5.QtCore.QTime(23, 34, 47, 275)	
52	257	565.78	94711.43	23.98	PyQt5.QtCore.QTime(23, 34, 47, 483)	
53	162	565.86	94709.64	23.98	PyQt5.QtCore.QTime(23, 34, 47, 691)	
54	2480	565.71	94710.91	23.98	PyQt5.QtCore.QTime(23, 34, 47, 899)	
55	2915	565.75	94711.39	23.98	PyQt5.QtCore.QTime(23, 34, 48, 107)	
56	1008	565.85	94712.37	23.98	PyQt5.QtCore.QTime(23, 34, 48, 315)	
57	250	565.68	94713.96	23.98	PyQt5.QtCore.QTime(23, 34, 48, 523)	
58	384	565.89	94711.52	23.99	PyQt5.QtCore.QTime(23, 34, 48, 731)	
59	2789	565.95	94710.42	23.99	PyQt5.QtCore.QTime(23, 34, 48, 939)	
60	3195	565.79	94710.91	23.98	PyQt5.QtCore.QTime(23, 34, 49, 147)	

Figura 15 – Dados armazenados na saída de arquivos tabelados

	A	B	C	D	E	F
61	3197	565.91	94708.11	23.99	PyQt5.QtCore.QTime(23, 34, 49, 355)	
62	979	566.05	94712.30	23.99	PyQt5.QtCore.QTime(23, 34, 49, 564)	
63	554	565.76	94714.09	23.99	PyQt5.QtCore.QTime(23, 34, 49, 771)	
64	2386	565.90	94712.14	24.00	PyQt5.QtCore.QTime(23, 34, 49, 979)	
65	3193	565.83	94712.30	24.00	PyQt5.QtCore.QTime(23, 34, 50, 187)	
66	3147	566.13	94706.44	24.00	PyQt5.QtCore.QTime(23, 34, 50, 395)	
67	1045	565.91	94709.41	23.99	PyQt5.QtCore.QTime(23, 34, 50, 603)	
68	368	566.03	94709.24	23.99	PyQt5.QtCore.QTime(23, 34, 50, 811)	
69	1957	565.90	94708.72	24.01	PyQt5.QtCore.QTime(23, 34, 51, 19)	
70	3201	566.01	94710.52	24.00	PyQt5.QtCore.QTime(23, 34, 51, 227)	
71	3001	565.74	94711.65	24.01	PyQt5.QtCore.QTime(23, 34, 51, 435)	
72	1153	565.87	94708.06	24.01	PyQt5.QtCore.QTime(23, 34, 51, 643)	
73	1337	565.94	94709.69	24.01	PyQt5.QtCore.QTime(23, 34, 51, 851)	
74	2930	565.96	94709.86	24.01	PyQt5.QtCore.QTime(23, 34, 52, 59)	
75	3200	566.00	94710.27	24.01	PyQt5.QtCore.QTime(23, 34, 52, 267)	
76	3088	565.91	94708.36	24.01	PyQt5.QtCore.QTime(23, 34, 52, 475)	
77	1210	566.03	94711.41	24.02	PyQt5.QtCore.QTime(23, 34, 52, 684)	
78	787	565.85	94709.95	24.02	PyQt5.QtCore.QTime(23, 34, 52, 891)	
79	2077	565.62	94711.90	24.02	PyQt5.QtCore.QTime(23, 34, 53, 99)	
80	3200	565.73	94711.22	24.03	PyQt5.QtCore.QTime(23, 34, 53, 307)	
81	3205	565.77	94712.03	24.03	PyQt5.QtCore.QTime(23, 34, 53, 515)	
82	2730	565.69	94714.30	24.03	PyQt5.QtCore.QTime(23, 34, 53, 723)	
83	1159	566.18	94711.09	24.03	PyQt5.QtCore.QTime(23, 34, 53, 931)	
84	1280	565.84	94710.27	24.04	PyQt5.QtCore.QTime(23, 34, 54, 139)	
85	3194	566.15	94710.11	24.03	PyQt5.QtCore.QTime(23, 34, 54, 347)	
86	3195	566.21	94709.10	24.03	PyQt5.QtCore.QTime(23, 34, 54, 555)	
87	3200	566.12	94707.80	24.03	PyQt5.QtCore.QTime(23, 34, 54, 763)	
88	865	566.16	94709.10	24.03	PyQt5.QtCore.QTime(23, 34, 54, 971)	
89	79	566.40	94710.23	24.04	PyQt5.QtCore.QTime(23, 34, 55, 179)	
90	225	566.12	94708.12	24.03	PyQt5.QtCore.QTime(23, 34, 55, 387)	

Figura 16 – Dados armazenados na saída de arquivos tabelados

	A	B	C	D	E	F
91	858	566.11	94709.06	24.04	PyQt5.QtCore.QTime(23, 34, 55, 595)	
92	2655	565.84	94710.07	24.04	PyQt5.QtCore.QTime(23, 34, 55, 803)	
93	2968	566.11	94710.69	24.04	PyQt5.QtCore.QTime(23, 34, 56, 11)	
94	3200	565.60	94707.77	24.04	PyQt5.QtCore.QTime(23, 34, 56, 220)	
95	2058	565.71	94708.41	24.05	PyQt5.QtCore.QTime(23, 34, 56, 427)	
96	352	565.97	94710.52	24.05	PyQt5.QtCore.QTime(23, 34, 56, 635)	
97	223	565.86	94711.02	24.05	PyQt5.QtCore.QTime(23, 34, 56, 843)	
98	432	565.88	94708.41	24.04	PyQt5.QtCore.QTime(23, 34, 57, 52)	
99	2442	565.90	94707.92	24.05	PyQt5.QtCore.QTime(23, 34, 57, 259)	
100	3200	566.10	94703.86	24.05	PyQt5.QtCore.QTime(23, 34, 57, 467)	
101	3204	566.06	94709.84	24.06	PyQt5.QtCore.QTime(23, 34, 57, 675)	
102						

Para desenvolvimento da interface de uso da aplicação, que permita a conexão entre o servidor de informações com o computador cliente, foi utilizado a biblioteca de desenvolvimento PYQT5. A Figura 17 exibe a tela inicial da aplicação, onde um campo deve ser preenchido com o IP de conexão do servidor de dados. Após preencher o campo como na Figura 18, é possível iniciar a conexão ao clicar no botão “IP to WBS”, onde uma confirmação surge para o usuário, como visto na Figura 19.

Ao confirmar o endereço IP para conexão websocket entre o servidor de dados desenvolvido com uso da plataforma ESP32 Devkit v1 e o computador cliente, a troca de dados se inicia e as informações são exibidas alternadamente na barra de status, como visto nas Figuras 20, 21, 22 e 23. Ao finalizar o loop programado, para aquisição

de 100 conjunto de dados do servidor, a aplicação encerra com a opção de encerrar a comunicação ou não a comunicação websocket, como demonstrado na Figura 24.

Figura 17 – Tela inicial da interface da aplicação



Figura 18 – Inserindo o endereço IP para conexão websocket



Figura 19 – Confirmação do endereço IP para conexão websocket

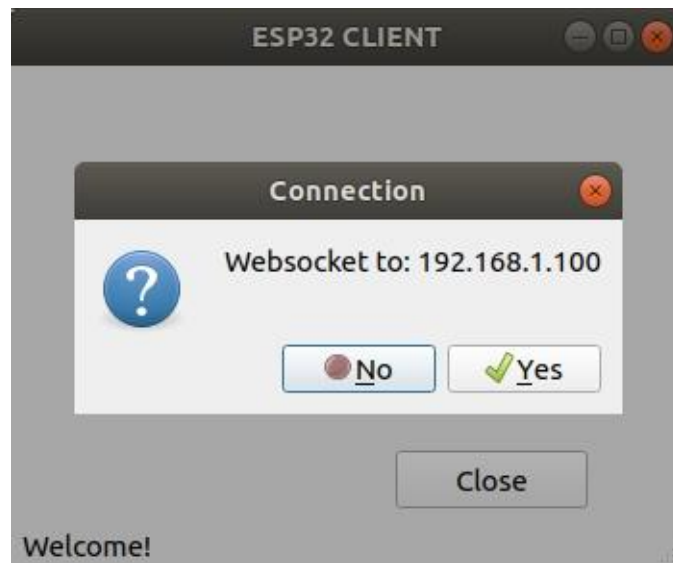


Figura 20 – Dado do sensor analógico MPXV7002DP



Figura 21 – Dado da altitude em metros, fornecido pelo sensor BMP280

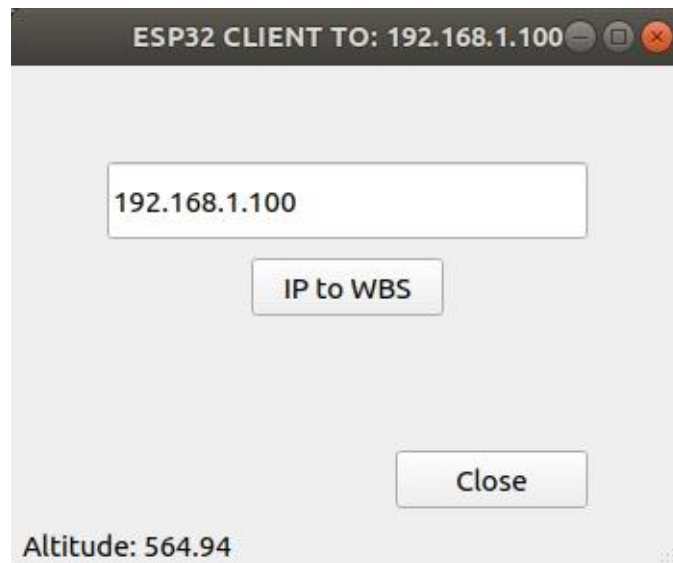


Figura 22 – Dado da pressão atmosférica em Pascal, fornecida pelo sensor BMP280



Figura 23 – Dado da temperatura em graus Celsius, fornecida pelo sensor BMP280

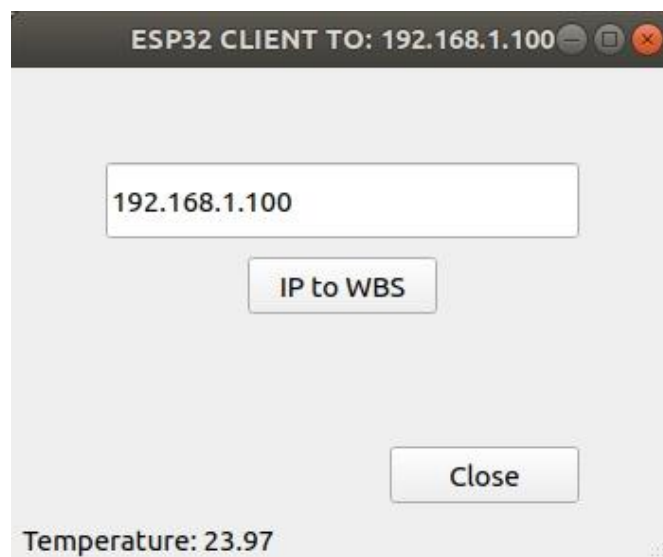
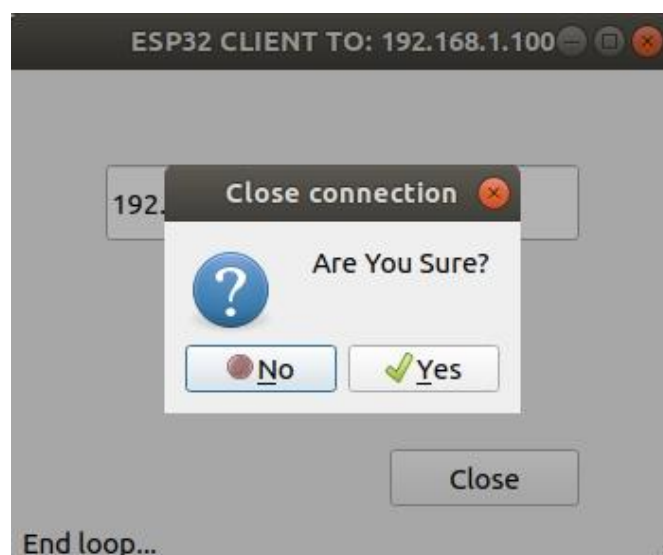


Figura 24 – Término do loop programado, para aquisição de 100 conjunto de dados do servidor



5. Conclusão

Este estudo desenvolveu a solução proposta, que contempla a arquitetura de hardware e software para a transmissão e armazenamento de dados, provenientes de sensores analógicos e digitais com foco em soluções no cenário IoT. Se mostra possível para aplicações em VANTs (com a utilização de sensores que favorecem projetos na área) além de possibilitar sensoriamento remoto de atividades e de outros tipos de aplicações.

Ao implementar uma rede WiFi associada ao uso de websocket para obtenção de monitoramento de dados de sensores, nota-se que é uma abordagem para resolução da proposta, de forma viável e prática. O acesso remoto as informações e a escalabilidade do projeto, são pontos interessantes de se notar pois foram utilizados componentes de hardware com baixo custo, que permitiram o desenvolvimento e possibilitam a reprodução dos experimentos com sucesso.

A interface com o usuário é um dos pontos que é de interesse para esse tipo de aplicação, a fim de facilitar o uso por demais pessoa, que não tem familiaridades com o desenvolvimento de projetos na área. Espera-se que demais desenvolvedores utilizem o projeto desenvolvido, como uma solução eficiente para a telemetria de seus projetos.

6. Referências bibliográficas

BOYLESTAD, Robert L. **Introdução à análise de circuitos**. 12. Ed. São Paulo, SP: Pearson Prentice Hall, 2012. 959 p. il. Tradução de: Introductory circuit analysis. ISBN 97885645574205

ASSEF, A. Amauri. **Sistemas Embarcados**. Universidade Tecnológica do Paraná. Disponível em <<http://paginapessoal.utfpr.edu.br/amauriassef/disciplinas/sistemas-embarcados-ppgse/aulas-e-roteiros>> Acesso em 14 jan. 2018

VENKATECH, Narasimhan. **Design Wi-Fi connectivity into your embedded internet thing**: Embedded, 11 jul. 2011. Disponível em <<https://www.embedded.com/design/connectivity/4217738/Designing-Wi-Fi-connectivity-into-your-embedded--Internet-thing->>> Acesso em: 9 out. 2018.

MYERS, Dana. **Wi-Fi for embedded Internet of Things applications**. Ecnmag: 30 jul. 2013. Disponível em <<https://www.ecnmag.com/article/2013/07/wi-fi-embedded-internet-things-applications>> Acesso em: 9 out. 2018.

VOKAS, Nikos. **Connecting devices to the Internet of Things with Wi-Fi**. Embedded Computing: 5 jan. 2015. Disponível em <<http://www.embedded-computing.com/embedded-computing-design/connecting-devices-to-the-internet-of-things-with-wi-fi#>> Acesso em: 9 out. 2018.