CoAP based IoT data transfer from a Raspberry Pi to Cloud

Thomas Lee Scott
School of Computing, Mathematics & Digital Technology
Manchester Metropolitan University
Manchester, United Kingdom
tomscott292@gmail.com

Amna Eleyan
School of Computing, Mathematics & Digital Technology
Manchester Metropolitan University
Manchester, United Kingdom
A.Eleyan@mmu.ac.uk

***Abstract*—This research investigates the use of The Constrained Application Protocol (CoAP) in transmitting sensor data to the cloud. It aims to explore how CoAP fits into the Internet of Things (IoT) ecosystem and what advantages it offers over other IoT protocols. A framework is proposed using a Raspberry Pi (RPi) and sensor acting as an IoT endpoint. This endpoint will poll the sensor and using CoAP will send the latest data formatted as JavaScript Object Notation (JSON) to a CoAP cloud endpoint at regular intervals. The cloud service will receive, format and display the data from the RPi to the user.**

***Index Terms*—Internet of Things, CoAP, M2M, constrained devices, Raspberry Pi board.**

I. Introduction

The reduced cost of low powered small devices, such as the RPi, has made it more accessible to create bespoke systems. This combined with the increasing popularity of home automation allows for these devices to be used in the IoT.

The IoT can be viewed as a large distributed network comprising of highly dynamic devices [1]. Small low powered "smart" devices can connect and communicate with one another. Some of these devices can contain or communicate with sensors that record real world data. This data can then be transmitted to other devices allowing them to trigger actions. In this way groups of smart devices can be used to improve day to day situations such as automated houses (thermostats and heating etc.), security and improved monitoring.

The Raspberry Pi [2] is a credit card sized computer developed by the Raspberry Pi Foundation. The RPi's ability to act as a GNU/Linux server and the interfacing services provided by its general purpose I/O pins make it a popular choice of hardware for IoT applications. [3]

With 48% of the UK market considering their smartphone as the most important device for Internet access [4], allowing users to use their handheld devices to view and manage their data has become increasingly necessary. Cloud platforms that allow access from any device go a long way to solving this problem. Storing sensor data in the cloud allows for easy access to users from any device as well as allowing for scalable storage.

As these devices are limited in computing power, it is important that the devices communicate efficiently. This paper explores the use of CoAP as a protocol to transmit sensor data from a small, low powered device (RPi) to send sensor data to the cloud.

In this system, a sensor will be attached to the RPi; the RPi will be responsible for taking the data from the sensor and then using the CoAP protocol to transmit this data to the cloud platform.

A. Motivation

With the increasing adoption of IoT systems and technology in day to day life, it becomes increasingly important for the devices to be able to operate at peak efficiency. As there are numerous technologies and standards that allow different IoT devices to communicate, this paper investigates the use of CoAP in order to determine: 1) how CoAP transfers data from a IoT device to another CoAP node. 2) how a RPi can be used as a flexible platform to provide sensor data. 3) how this data can be sent to a cloud platform.

B. Related Work and Contribution

Rode, Sagrolikar, and Kulkarni [5] carries out a similar investigation, using a RPi device as a IoT node connected to sensors. The RPi collects the data from the sensors and then transmits the data to a cloud platform. The cloud platform in this instance is a HyperText Transfer Protocol (HTTP) server which will receive the sensor data and display it to the user. Rode, Sagrolikar, and Kulkarni [5] proposes using the Message Queuing Telemetry Transport (MQTT) protocol to transmit the sensor data from the RPi to the HTTP server. MQTT is a popular IoT protocol developed to specialise in the transfer of data from Wireless Sensor Networks (WSNs) [6]. MQTT works on a publish/subscribe model; in Rode, Sagrolikar, and Kulkarni [5] the RPi acts a *publisher*, publishing the sensor data to the broker and the HTTP server acts as a *subscriber*. The MQTT broker is responsible for coordinating subscribers to the data and subscribers will usually have to contact the broker explicitly in order to subscribe [6]. This contrasts to the approach taken in this proposal using CoAP, where each node in the CoAP network acts as both a server and a client in a more traditional HTTP model and nodes within the infrastructure will communicate with one another directly.

Jassas, Qasem, and Mahmoud [7] used a RPi connected to sensors to measure patients' body temperature and transmit

this data wirelessly to the cloud. In that paper, the data was transmitted to an Amazon Web Services (AWS) cloud computing platform. There the data was stored, mined in order to make decisions, and displayed to the user allowing the data to be updated and reviewed. The data was transmitted from the RPi to the AWS server using Secure Socket Layer (SSL). The development of specialised protocols for constrained devices, such as CoAP could allow these health monitoring RPis to save power, save network bandwidth and potentially receive more readings to process.

Lee, Park, Jung, *et al.* [8] used a RPi combined with a DHT22 sensor to measure the indoor temperature in real time. This data was transmitted using HTTP to a Representational State Transfer (REST) Application programming interface (API), where the temperature was stored in a database. These temperatures were then used to inform an application replicating the actions of an air conditioner. The use of a RESTful API in this paper would allow the project to easily be adapted to using CoAP to replace HTTP.

In comparison to the research examined, this paper proposes using CoAP to implement Machine-to-Machine (M2M) communication. The research undertaken in the study will benefit future research into IoT devices and networks and provide knowledge of how CoAP can be implemented on a constrained device and how this device can then transmit data to the cloud.

II. Background

A. Internet of Things

The Internet of Things (IoT) is an umbrella term used to describe physical 'smart' devices equipped with telecommunication interfaces, connected to one another via the Internet [9]. Whereas the Internet traditionally connected computers, the embedding of electronics into physical objects has allowed the Internet to expand [1]. These devices can contain sensors which will produce data and in some cases these devices can be controlled remotely. The combination of these devices in a network, especially when the actions of one devices are informed by the data from another device, is the foundation of IoT [10]. IoT systems can impact in many areas such as home automation, where devices can work together to automate heating and security aspects of the home [8], and medicine where devices can be used as monitors to provide real time information about patient health [3].

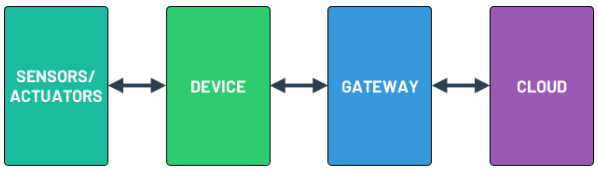

Figure 1: Basic IoT Architecture.

Figure 1 illustrates the various components present within any given IoT architecture. The sensor/actuator is the component that will receive data from the real world or perform a physical action. This data is then transmitted to a device which is responsible for the sensor/actuator. This device may be responsible for multiple sensors, as in Jassas, Qasem, and Mahmoud [7] which connected multiple e-health sensors to a RPi device. This device will then communicate with a gateway. The gateway will handle the destination of data passed to it. Within CoAP the gateway would be a router ensuring messages are sent to the correct endpoints. The cloud is final destination for the sensor data where the data will be stored and can be further used for analysis.

B. The Constrained Application Protocol

The Constrained Application Protocol (CoAP) is a transfer protocol specialised for use with the web, constrained nodes and constrained networks [11]. The protocol is designed for M2M applications and is ideally suited for use within the IoT ecosystem. CoAPs features of observable resources, multicasting, M2M discovery make it a better fit for IoT applications than HTTP [12].

CoAP recognises that web services have become dependant in REST architecture and works to implement a subset of REST common with HTTP while optimising for M2M applications [11]. It achieves this by offering built-in discovery, multicast support and asynchronous message exchanges [11].

CoAP uses a compact binary format with a fixed header size of 4 bytes, exchanging messages over User Datagram Protocol (UDP) or Datagram Transport Layer Security (DTLS) to send messages securely. CoAP resources are addressable by Uniform Resource Identifiers (URIs) and can be interacted with through the same methods as HTTP: GET, PUT, POST and DELETE.

With regards to reliability, CoAP offers four types of messages: Confirmable, Non-Confirmable, Acknowledgement and Reset [13]. After a Confirmable request is sent to a CoAP endpoint, the endpoint will respond with an Acknowledgement message. This message can contain the requested data in a 'piggybacked' response. Otherwise, an empty Acknowledgement message is sent and a Confirmable message will be sent once the data is ready. The original requester will then respond with an empty Acknowledgement message to confirm receipt of the data [11].

III. Designed CoAP-based IoT Architecture

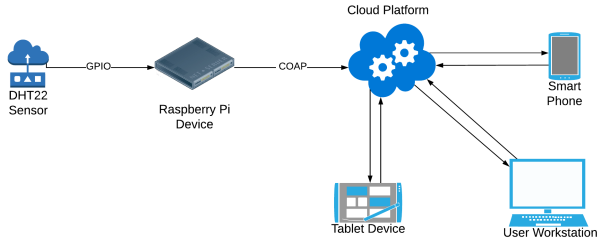

Figure 2: CoAP-based IoT Architecture.

The system shall consist of four main elements: the sensor, the RPi, CoAP and the cloud platform. The sensor will collect

the data and pass this to the RPi. The RPi will then be responsible for manipulating the data into a suitable format for transmission via CoAP. The implementation of CoAP will communicate with the cloud platform. The cloud platform will store the data, allowing access to users. The parts of the system are explained below:

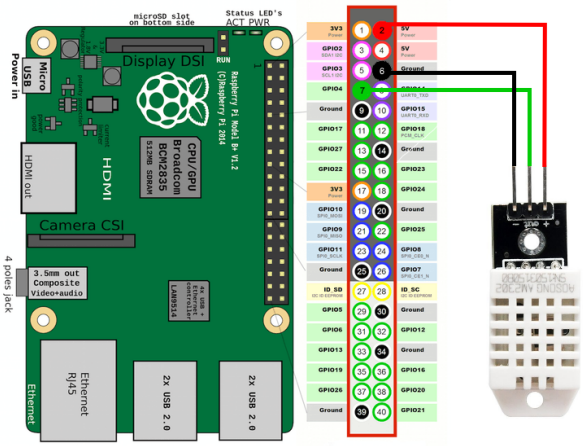

Figure 3: Wiring diagram for connecting the DHT22 sensor to the RPi.

Sensors The DHT22/AM2302 digital relative temperature and humidity sensor will be used to collect temperature and humidity data from the environment. The DHT22 has three pins: One for the 3.3–5.5V power, one for data output and one pin for neutral. The RPi can connect to the DHT22 using the RPi's General Purpose Input Output (GPIO) ports to provide power, a ground connection and an output for the DHT22's data, this connection is shown in Figure 3. The DHT is capable of measuring temperatures in the range of -40 to 80 degrees celsius and reporting a relative humidity of between 0 to 100% to an accuracy of $\pm 2\%$. The largest negative to the DHT22 sensor is that it will only report new data once every two seconds, leading to reading being up to two seconds old.

Device The Raspberry Pi 3 Model B is a Linux-based microcomputer. It includes built in WiFi, GPIO ports, a 1.2GHz Quad-Core processor, MicroSD card slot, memory, video/audio outputs, Ethernet port, and power source [2]. It uses a MicroSD memory card as a boot drive and runs a specialised GNU/Linux distribution named Raspbian [14]. Although the RPi can be attached to a monitor using the built in HDMI port and controlled with a USB mouse and keyboard, this study will connect to the RPi using SSH. The RPi will run a Python script which will collect the data from the DHT22 sensor and send it to the CoAP endpoint in the cloud.

Gateway As this is a CoAP based architecture and as such mirrors the workings of HTTP the gateway will be the local networks router. This will be responsible for directing the CoAP messages to the CoAP endpoint identified by the cloud platforms URI.

Cloud This study will make use of an open source cloud platform, ThingsBoard. ThingsBoard is an IoT platform that allows the collection and visualisation of data from IoT devices, connection from IoT devices using MQTT, CoAP or HTTP, the definition of rules to validate incoming data among other features. The RPi will use the URI provided by the ThingsBoard CoAP API to send the sensor data to. Once ThingsBoard receives the sensor data it will update a dashboard which will show the temperature and humidity data provided by the sensor over time, this dashboard is displayed in Figure 5.

The complete design of this system and how each component will interact is illustrated in Figure 2.

The DHT22 sensor will connect to the RPi using the RPi's on board GPIO ports as shown in Figure 3. The AdaFruit Python DHT library will be used in a Python script to get the sensor data from the DHT22 sensor. The AdaFruit Python DHT library is a library that provides methods to interact with DHT sensors connected to the RPi's GPIO pins. A CoAP endpoint will need to be created on the RPi in order to send the sensor data to the cloud. Using the CoAPthon Python library the script will create a CoAP endpoint on the RPi. The CoAPthon library is a Python implementation of the CoAP protocol.

Data: Temperature and humidity data from sensor
Result: Sensor data sent to Cloud CoAP endpoint
initialisation;
define interval in seconds (DHT22 sensor interval is 2 seconds);
while *running* **do**
  get latest sensor data from DHT22 sensor;
  if *sensor data is returned* **then**
    format data into json object;
    create CoAP Post message with cloud uri as destination;
    send CoAP message;
  else
    wait interval time and continue;
  end
end

Algorithm 1: How to get data from sensor and send to cloud.

This endpoint will act as an interface for the DHT22 sensor. At intervals the script will use the AdaFruit DHT library [15] methods to retrieve the DHT22 sensor's current temperature and humidity readings. This data will then be formatted into JSON in order to be transmitted. Then using the CoAPthon [16], [17] library a CoAP message will be created with the sensors JSON data as the payload. This loop will then repeat while the RPi is active, this process is formalised in Algorithm 1. This message will then be sent over UDP to

a CoAP URI hosted by the cloud platform as illustrated in Figure 4.

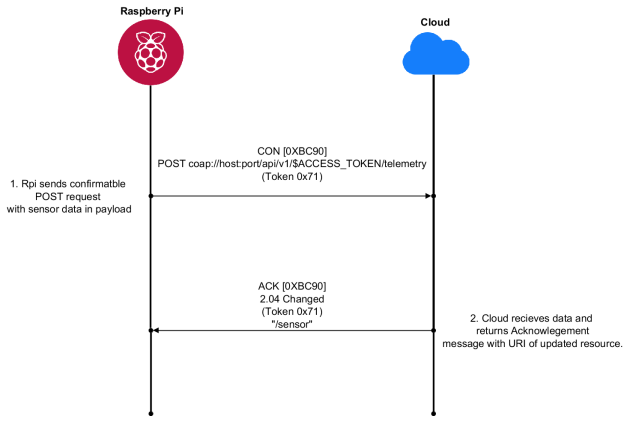

Figure 4: Diagram showing the communication between RPi and the Cloud.

The cloud's responsibility will be to receive the POST requests from the RPi containing the JSON data, format it and display it to the user, who will be accessing the cloud platform via HTTP. Once the ThingsBoard platform has received the request it should send an Acknowledgement message to the CoAP endpoint, containing a 2.01 (Created) Response Code or a 2.04 (Changed) Response Code and the URI of the created / updated resource [11].

IV. CoAP-based IoT Architecture Implementation

To implement the proposed architecture, first the RPi was loaded with a headless distribution of the Raspbian [14] operating system. Raspbian is a specialised GNU/Linux distribution based on Debian optimized for the RPi. It comes with over 35,000 packages including the Python programming language. Python will be used to create and run the script that comprises of the implementation of the client side system. After setting up the RPi and connecting the DHT22 sensor to the RPi's GPIO pins as per Figure 3, a Python script was constructed to collect the sensor data and using CoAP send the data to the ThingsBoard [18] endpoint.

The CoAP client is created using the Python package, CoAPthon [17]. This package contains an `HelperClient` class that takes a CoAP path and port upon initialisation. An object is created using the path to the ThingsBoard telemetry endpoint and the default CoAP port of 5683. This CoAP client object will be used to send the POST message containing the data to the ThingsBoard cloud.

To retrieve the readings from the DHT22 sensor first the Adafruit DHT22 library [15] was imported to the script, this library contains a `read_retry()` method that will attempt to read the temperature and humidity data from the DHT22 sensor and return values as floating point decimals, if no reading is available it will try again up to a specified number of retries, defaulting to fifteen.

```
1        {
2            'temperature': 22.6,
3            'humidity': 30.05
4        }
```

Listing 1: Sensor data formatted to JSON for message payload.

Once the sensor data is returned, the payload is constructed. The payload consists of the temperature and humidity data formatted into a JSON object, shown in Listing 1. This data is converted using the built-in Python JSON library. With the formatted payload the CoAP message is sent to the ThingsBoard endpoint, the response from the message is printed to the console. Here the script will wait a specified amount of time and then will repeat the process from retrieving the data. This process is shown in Algorithm 1. The Python script that accomplishes this is shown in the Appendix.

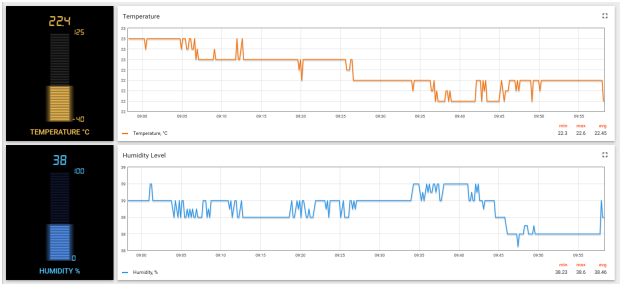

Figure 5: ThingsBoard dashboard showing temperature and humidity data.

Once the ThingsBoard cloud receives the data, the data is processed through the ThingsBoard cloud's rules engine. For this study no extra rules have been applied to the data handling. The default handling of data sent to the telemetry endpoint is to identify the device that is sending the data, store the values and update any dashboards associated with that device. The device is identified by the device access token sent as part of the request. An image of the ThingsBoard dashboard for this study is shown in Figure 5. These dashboards are customisable and allow for the data to be shown in realtime or show a selection of historical results. The dashboard in Figure 5 has four display widgets: one showing the last temperature reading, one showing all the latest temperature readings for the previous hour, one showing the last humidity reading and one showing all humidity readings in the last hour.

V. Conclusion and Future Work

This research investigates the use of CoAP in transmitting sensor data to the cloud. It aims to explore how CoAP fits into the IoT ecosystem and what advantages it offers over other IoT protocols. It also shows how a RPi can be used with Python to create an IoT device capable of transferring data to the cloud.

The RPi has been connected to a DHT22 temperature and humidity sensor using the RPis GPIO pins. A Python script running on the RPi uses an external library provided by AdaFruit to poll the sensor at intervals and obtain the current

temperature and humidity. This data is then formatted and displayed to the console.

Further work can be done with the data once it has been sent to the cloud. The ThingsBoard platform offers many options with regards to actions performed based on data received from IoT devices. Setting up alerts when data falls outside of expected ranges and sending messages to other IoT devices as a result would allow this study to be applied to other uses, such as smart home automation. It would also be of interest to connect an actuator to the RPi and see how this could be manipulated based on CoAP messages sent from the cloud.

APPENDIX
PYTHON SCRIPT TO RETRIEVE SENSOR DATA AND SEND TO THE CLOUD.

```
#!/usr/bin/env python3
"""
Script to get sensor data from DHT22 and provide the data through a
CoAP endpoint
"""

__author__ = "Tom Scott"

import Adafruit_DHT
from json import dumps
from socket import gethostbyname, gaierror
from time import sleep
from socket import gethostbyname, gaierror

from coapthon.client.helperclient import HelperClient
from coapthon.utils import parse_uri

_SENSOR = Adafruit_DHT.AM2302
_GPIO_PIN = 4
_HOST = "demo.thingsboard.io"
_DEVICE_AUTH_TOKEN = "9tQn151djuEdRX0ucu0k"
_PORT = 5683 # Default CoAP port
_SLEEP_INTERVAL = 5

class SensorData(object):
    """
    Data object containing humidity and temperature data.
    """

    def __init__(self, humidity, temperature):
        """
        Set humidity and temperature
        """
        self.humidity = humidity
        self.temperature = temperature

    def has_data(self):
        """
        Returns True if humidity and temperature are not None.
        """
        return self.humidity is not None and \
               self.temperature is not None

    def as_json(self):
        """
        Returns the humidity and temperature data as a JSON string.
        """
        return dumps({'humidity': self.humidity, \
                'temperature': self.temperature})

    def __str__(self):
        humidity_string = 'Humidity={0:.1f}%'.format(self.humidity)
        temperature_string = 'Temp={0:.1f}*C'.format(self.temperature)
        return humidity_string + ' ' + temperature_string

def get_sensor_data():
    """
    Returns a SensorData object containing the latest data
    from the DHT22 sensor.
    The method will 15 times to get data, if there is still no
    data available it will return an empty SensorData object
    """

    # Try to grab a sensor reading.  Use the read_retry method
    # which will retry up to 15 times to get a sensor reading
    # (waiting 2 seconds between each retry).
    humidity, temperature = Adafruit_DHT.read_retry(_SENSOR, _GPIO_PIN)

    return SensorData(humidity, temperature)

def get_coap_client():
    """
    Procedure for creating the CoAP client.
    Attempts to get IP address for host from the host name.
    Returns CoAP HelperClient
    """
    host = _HOST

    # Try to get ip address
    try:
        tmp = gethostbyname(host)
        host = tmp
    except gaierror:
        # use domain if cannot get ip
        pass

    # create CoAP client
    return HelperClient(server=(host, _PORT))

def main():
    """ Main entry point of the app """

    # Create path that the data will be sent to
    path = "api/v1/" + _DEVICE_AUTH_TOKEN + "/telemetry"
    # Create a CoAP client
    client = get_coap_client()

    try:
        while True:

            # Note that sometimes you won't get a reading and
            # the results will be null (because Linux can't
            # guarantee the timing of calls to read the sensor).
            sensor_data = get_sensor_data()

            if sensor_data.has_data():
                # Format the data to JSON to be sent
```

```
                payload = sensor_data.as_json()
                # Send POST message with payload
                ↪  to Cloud
                response = client.post(path,
                ↪  payload)
                # Print response from cloud to
                ↪  console
                print(response.pretty_print())

            # Wait specified time and repeat
            sleep(_SLEEP_INTERVAL)

    # Allow Ctrl + C to stop the script
    except KeyboardInterrupt:
        print("Interrupted by keyboard, stopping
        ↪  client")
        client.stop()
        exit(0)

if __name__ == "__main__":
    """ This is executed when run from the
    ↪  command line """
    main()
```

REFERENCES

[1] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, Sep. 1, 2012.

[2] R. Pi, "Model b," *Raspberrypi. org. Saatavissa: https://www. raspberrypi. org/products/raspberry-pi-3-model-b/. Hakupäivä*, vol. 6, p. 3, 2018.

[3] R. Kumar and M. P. Rajasekaran, "An IoT based patient monitoring system using raspberry pi," in *2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16)*, Jan. 2016, pp. 1–4.

[4] Ofcom. (Aug. 1, 2018). The communications market 2018: Narrative report, Ofcom.

[5] S. D. Rode, S. Sagrolikar, and M. S. Kulkarni, "IOT based raspberry PI home automation using cloud," vol. 3, no. 2, p. 5, 2017.

[6] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "MQTT-s — a publish/subscribe protocol for wireless sensor networks," in *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, Jan. 2008, pp. 791–798.

[7] M. S. Jassas, A. A. Qasem, and Q. H. Mahmoud, "A smart system connecting e-health sensors and the cloud," in *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*, May 2015, pp. 712–716.

[8] C. Lee, S. Park, Y. Jung, Y. Lee, and M. Mathews, "Internet of things: Technology to enable the elderly," in *2018 Second IEEE International Conference on Robotic Computing (IRC)*, Jan. 2018, pp. 358–362.

[9] M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi, "Long-range communications in unlicensed bands: The rising stars in the IoT and smart city scenarios," *IEEE Wireless Communications*, vol. 23, no. 5, pp. 60–67, Oct. 2016.

[10] R. Minerva, A. Biru, and D. Rotondi, "Towards a definition of the internet of things (IoT)," *IEEE Internet Initiative*, vol. 1, pp. 1–86, 2015.

[11] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (CoAP)," RFC Editor, RFC7252, Jun. 2014.

[12] M. Kovatsch, M. Lanter, and Z. Shelby, "Californium: Scalable cloud services for the internet of things with CoAP," in *2014 International Conference on the Internet of Things (IOT)*, Oct. 2014, pp. 1–6.

[13] P. Bellavista and A. Zanni, "Towards better scalability for IoT-cloud interactions via combined exploitation of MQTT and CoAP," in *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, Bologna, Italy: IEEE, Sep. 2016, pp. 1–6.

[14] Raspbian, *Raspbian GNU/linux 9.6 (stretch)*, version 9.6, Nov. 2018. [Online]. Available: https://www.raspbian.org.

[15] Adafruit, *Adafruit_python_dht*, version 1.4.0, Nov. 7, 2018. [Online]. Available: https://github.com/adafruit/Adafruit_Python_DHT.

[16] G. Tanganelli, C. Vallati, and E. Mingozzi, "CoAPthon: Easy development of CoAP-based IoT applications with python," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Dec. 2015, pp. 63–68.

[17] G. Tanganelli, *CoAPthon3*, version 1.0.1, Jan. 25, 2018. [Online]. Available: https://github.com/Tanganelli/CoAPthon3.

[18] ThingsBoard, Inc., *ThingsBoard*, version 2.2.0, Nov. 29, 2018. [Online]. Available: https://thingsboard.io.