CoAP based IoT data transfer from a Raspberry Pi to Cloud

Thomas Scott

20th November 2018

1 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, if any, 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.

2 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 (Miorandi et al. 2012). 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 (Pi 2018) 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. (Kumar & Rajasekaran 2016)

With 48% of the UK market considering their smartphone as the most important device for internet access (Ofcom 2018) 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 the 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.

3 The Constrained Application Protocol

The Constrained Application Protocol (CoAP) is a transfer protocol specialised for use with the web, constrained nodes and constrained networks (Shelby et al. 2014). The protocol is designed for Machine-to-Machine (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 HyperText Transfer Protocol (HTTP) (Kovatsch et al. 2014).

CoAP recognises that web services have become dependant in Representational State Transfer (REST) architecture and works to implement a subset of REST common with HTTP while optimising for M2M applications (Shelby et al. 2014). It achieves this by offering built-in discovery, multicast support and asynchronous message exchanges (Shelby et al. 2014).

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 (Bellavista & Zanni 2016). 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 (Shelby et al. 2014).

4 Design

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.

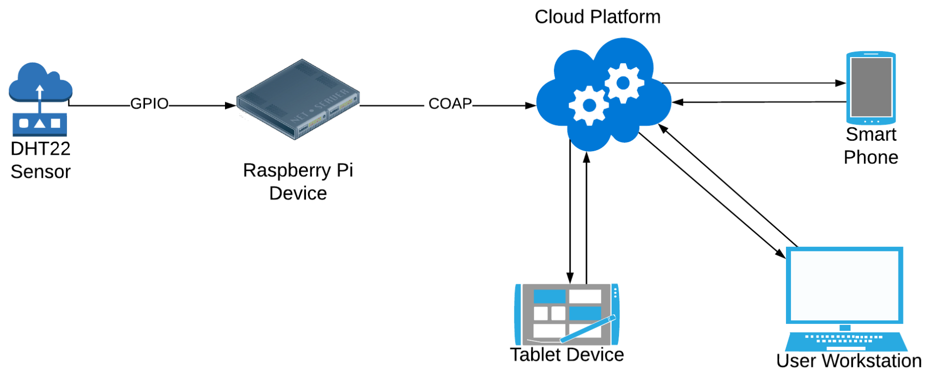

Figure 1: The project infrastructure.

The DHT22 sensor will connect to the RPi using the RPi's on board General Purpose Input Output (GPIO) ports as shown in Figure 2. Using the AdaFruit Python DHT library, a library that provides methods to interact with DHT sensors connected to the RPi's GPIO pins, and the CoAPthon library, a Python implementation of the CoAP protocol, a Python script will create a CoAP endpoint. This endpoint will act as an Application programming interface (API) for the DHT 22 sensor. The script will use the AdaFruit DHT methods to get the DHT 22 sensors temperature and humidity data. This data will then be formatted into JSON in order to be transmitted. Then using the CoAPthon library a CoAP message will be created with the sensors JSON data as the payload. This message will then be sent over UDP to a CoAP URI hosted by the cloud platform. The cloud endpoint will receive the JSON data, format it and display it to the user, who will be accessing the cloud platform via HTTP. This process is shown in Algorithm 1.

```
Data: Temperature and humidity data from sensor
Result: Sensor data sent to Cloud CoAP endpoint
initialization;
define interval in seconds (DHT 22 sensor interval is 2 seconds);
while running do
    get latest sensor data from DHT 22 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.

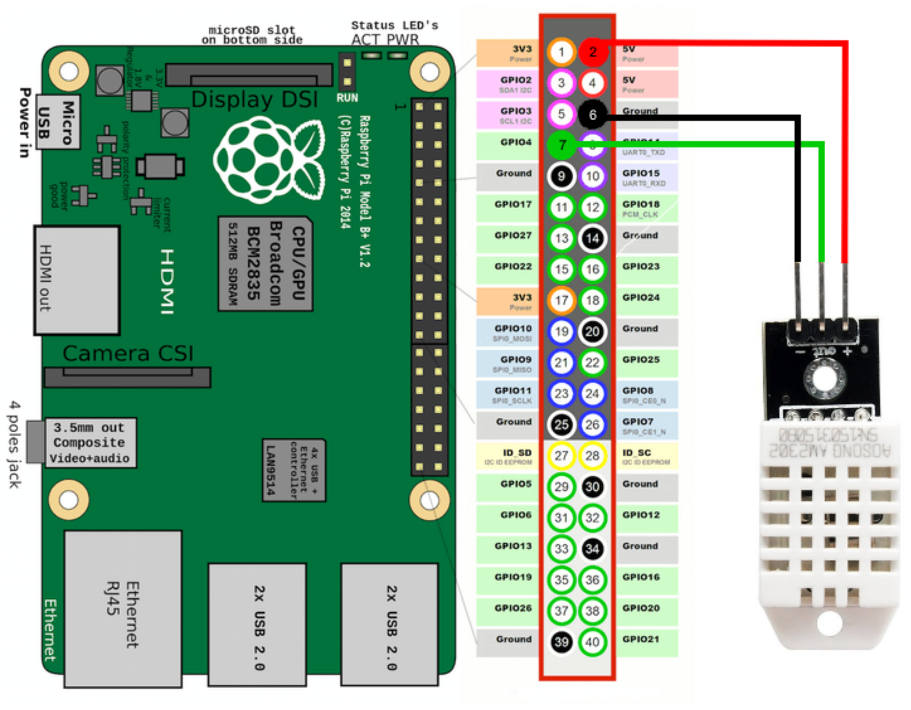

Figure 2: Wiring diagram for connecting the sensor to the RPi.

5 Related Work

Rode et al. (2017) 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 HTTP server which will receive the sensor data and display it to the user. Rode et al. (2017) 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) (Hunkeler et al. 2008). MQTT works on a publish/subscribe model, in Rode et al. (2017) 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 (Hunkeler et al. 2008). 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 et al. (2015) used a RPi connected to sensors to measure patient's body temperature and transmit this data wirelessly to the cloud. In that paper, the data was transmitted to a Amazon Web Services (AWS) cloud computing platform where the data was stored, mined in order to make decisions based on the data and provided a service for updating, reviewing and displaying the data to users. 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 and MQTT could allow these health monitoring RPis to save power, save network bandwidth and potentially receive more readings to process.

Lee et al. (2018) used a RPi combined with a DHT22 sensor to measure the indoor temperature in real time. This data was transmitted using HTTP to a REST 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.

6 Methodology

The aim of the proposed system is to investigate the implementation of CoAP on a RPi and how CoAP can be used to transmit data to the cloud.

To achieve this a CoAP endpoint will need to be created on the RPi. The RPi will collect sensor data at intervals and store them locally on the device. The RPi will act as a CoAP server that will send to REST POST requests with the sensor data and the time the reading was taken in a JSON format.

The clouds responsibility will be to receive the POST requests from the CoAP endpoint hosted on the RPi and to store and format the data. The cloud 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 (Shelby et al. 2014).

With CoAP endpoints acting as both a client, that sends requests, and a server implementation of CoAP will be needed in each the RPi and the cloud platform. The RPi will then regularly retrieve readings from the sensor and send a POST request at intervals to the cloud CoAP endpoint.

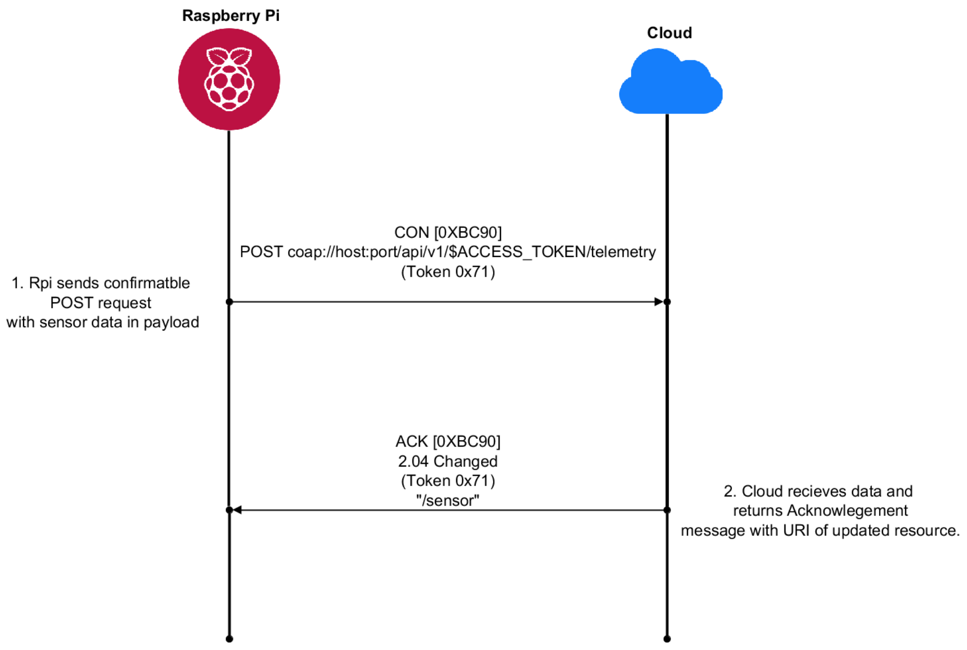
Raspberry Pi
Cloud
CON [0XBC90]
POST coap://host:port/api/v1/$ACCESS_TOKEN/telemetry
(Token 0x71)
1. Rpi sends confirmatble
POST request
with sensor data in payload
ACK [0XBC90]
2.04 Changed
(Token 0x71)
"/sensor"
2. Cloud recieves data and
returns Acknowlegement
message with URI of updated resource.

Figure 3: Diagram showing the communication between RPi and the Cloud.

7 Software and Hardware specification

7.1 Hardware

1. Raspberry Pi 3

   Raspberry Pi 3 Model B, includes built in WiFi, GPIO ports and a 1.2GHz Quad-Core processor.

2. Micro SD card

   Used to load operating system onto the RPi.

3. Power cord

   Supplies power to the RPi.

4. DHT22 temperature and humidity sensor

   Connects to the RPi using the RPi's GPIO ports. Will be used to provide data.

7.2 Software

1. Python 3

   The Python programming language will be used to create the scripts and software needed on the RPi. This is due to the languages popularity when creating projects on the RPi and the languages wide selection of networking packages.

2. CoAPthon

   Python implementation of CoAP. Licensed under the MIT license. Github.

3. AdaFruit Python DHT

   Python library to retrieve sensor data from the DHT22. Github.

4. Git Version Control

   Source code version control system to allow for adjustments to the code.

5. ThingsBoard Cloud Platform

   Will act as an end point to the RPi where the sensor data will be stored. ThingsBoard Homepage

8 Conclusion

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, if any, it offers to other protocols. It also shows how a RPi can be used with Python to create an IoT device and connect to the cloud.

Appendices

A Python Code to Get DHT22 Data

```python
#!/usr/bin/env python3
"""
Script to get sensor data from DHT22 and print it to console
"""

__author__ = "Tom Scott"

import Adafruit_DHT

_SENSOR = Adafruit_DHT.AM2302
_GPIO_PIN = 4

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

    def __init__(self, humidity, temperature):
        """
        Set temperature and humidity
        """
        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 get_sensor_data():
    """
    Returns a SensorData object containing the latest data
↪   from the DHT22 sensor.
```

```python
    The method will try 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 main():
    """ Main entry point of the app """

    # 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 and display data to console
        print('Temp={0:0.1f}* Humidity={1:0.1f}%'.format(sensor_data.temperature, sensor_data.humidity))


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

Acronyms

API Application programming interface.

AWS Amazon Web Services.

CoAP The Constrained Application Protocol.

DTLS Datagram Transport Layer Security.

GPIO General Purpose Input Output.

HTTP HyperText Transfer Protocol.

IoT Internet of Things.

JSON JavaScript Object Notation.

M2M Machine-to-Machine.

MQTT Message Queuing Telemetry Transport.

REST Representational State Transfer.

RPi Raspberry Pi.

SSL Secure Socket Layer.

UDP User Datagram Protocol.

URI Uniform Resource Identifier.

WSN Wireless Sensor Network.

References

Bellavista, P. & Zanni, A. (2016), 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)', IEEE, Bologna, Italy, pp. 1–6.
URL: *http://ieeexplore.ieee.org/document/7740614/*

Hunkeler, U., Truong, H. L. & Stanford-Clark, A. (2008), 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)', pp. 791–798.

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

Kovatsch, M., Lanter, M. & Shelby, Z. (2014), Californium: Scalable cloud services for the Internet of Things with CoAP, *in* '2014 International Conference on the Internet of Things (IOT)', pp. 1–6.

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

Lee, C., Park, S., Jung, Y., Lee, Y. & Mathews, M. (2018), Internet of Things: Technology to Enable the Elderly, *in* '2018 Second IEEE International Conference on Robotic Computing (IRC)', pp. 358–362.

Miorandi, D., Sicari, S., De Pellegrini, F. & Chlamtac, I. (2012), 'Internet of things: Vision, applications and research challenges', *Ad Hoc Networks* **10**(7), 1497–1516.
URL: *http://www.sciencedirect.com/science/article/pii/S1570870512000674*

Ofcom (2018), 'The Communications Market 2018: Narrative report'.
URL: *https://www.ofcom.org.uk/research-and-data/multi-sector-research/cmr/cmr-2018/report*

Pi, R. (2018), 'model B', *Raspberrypi. org. Saatavissa: https://www. raspberrypi. org/products/raspberry-pi-3-model-b/. Hakupäivä* **6**, 3.

Rode, S. D., Sagrolikar, S. & Kulkarni, M. S. (2017), ‘IOT based Raspberry PI home Automation Using Cloud’, **3**(2), 5.

Shelby, Z., Hartke, K. & Bormann, C. (2014), The Constrained Application Protocol (CoAP), Technical Report RFC7252, RFC Editor.
URL: *https://www.rfc-editor.org/info/rfc7252*