

2020

Gokul Shaji & Jitto
Thomas

[INTERNET OF THINGS DOCUMENTATION]

The Internet of Things is a system whereby physical devices are connected to the internet and are able to communicate with other devices. IoT enables autonomous and secure transfer of data between physical devices and software applications.

1	INTRODUCTION	2
2	OVERVIEWS OF INTERNET OF THINGS (IOT)	3
2.1	Examples of IoT Applications:	7
3	IOT ARCHITECTURE	8
4	IOT PROTOCOLS	13
4.1	MQTT	13
4.2	DDS	13
4.3	AMQP	14
4.4	Bluetooth	14
4.5	ZigBee	14
4.6	WiFi	15
4.7	Cellular	15
4.8	LoRaWAN	16
5	IOT IMPLEMENTATION FOR AIR CONDITION MONITORING	16
5.1	The Raspberry Pi	17
5.2	Temperature Sensor (DSB1820)	20
5.3	LORA Module (LoStik)	23
5.4	Cayenne Practical Implementation	25
6	PROGRAM EXPLANATION	32
6.1	LoRaReAd Git folder Overview	32
6.1.1	Thread_to_MQTT.py	33
7	ALGORITHM AND BLOCK DIAGRAM	43
8	CONCLUSION	45
9	REFERENCES	46

1 INTRODUCTION

The Internet of Things (IoT) is a computing system by which physical devices are connected to the internet and be able to communicate with other devices/systems. IoT devices can communicate using IP connectivity without human interference. The devices need to have smart systems which are capable of collecting and transferring the data over the network. The collected data can be processed at the edge device or can be sent to the cloud infrastructure through different connectivity options. 2

There are numerous options for connecting the Internet of Things. Among these connectivity options are cellular, satellite, Wi-Fi, Bluetooth, RFID, NFC, LPWAN, and the Ethernet. A suitable connectivity option is chosen based on power consumptions level, range that can be covered and bandwidth for the data transfer. Usually a tradeoff between these factors needs to be made when choosing the most suitable connection option. After the data has been transferred to the cloud, software platforms perform data processing on it. The software can send back instructions to the 'thing' or it performs action automatically based on the instructions programmed on it.

There are different IoT platform options by different vendors. An IoT platform called Cayenne from a company called myDevices is used in this project. Cayenne is a GUI (graphical user interface) based IoT project builder. A temperature monitoring system had been built using the Cayenne platform. A PicAxe processor is used to read sensors and transmit data over LoRa which is received by another sensor attached to a Raspberry Pi. The sensors will send sensor values to the Cayenne cloud infrastructure. There is an MQTT connection protocol between the sensors and the Cayenne cloud. This IoT platform can be accessed from the web or a mobile application.

This project will discuss the theory behind the IoT operation in chapter two. Chapter three will be about the different layers of IoT architecture. Different computing architectures are also discussed in this chapter. Chapter four will be about the different protocols used in IoT applications. These protocols will be divided and discussed across the four layers of the Internet protocol suite. In chapter five there is a practical IoT implementation using the Cayenne platform. The results of this implementation are also discussed. In chapter six, the conclusion about the thesis is discussed.

2 OVERVIEWS OF INTERNET OF THINGS (IOT)

The Internet of Things is a system whereby physical devices are connected to the internet and are able to communicate with other devices. This communication allows the devices to send and receive data over the network. Human-human communication is the main form of present day internet technology. The internet of things (IoT) can be considered as the next breakthrough of the internet by enabling machine-machine communication.

IoT enables autonomous and secure transfer of data between physical devices and software applications. The physical devices can have different sizes, data storage capabilities and processing power. They also support different set of applications. IoT brings the physical world and the virtual world together.

The term 'Internet of things' was first coined and used by Kevin Ashton, a British technology Innovator who was working on the implementation of radio frequency identification (RFID) technology to connect physical devices to the internet. Since then, Internet of Things (IoT) has seen rapid development and expansion into different fields. Organizations in different sectors are increasingly adopting IoT to gather more data about physical devices, their services and

customers. This allows companies to operate more efficiently, improve customer experience and create new business models thereby generating more revenue.

The internet of things is a big network and is getting bigger. Different organizations provide different estimates on the number of connected devices in year 2020. The technology research firm Gartner Inc. forecasts that this number could be about 8 billion. Ericsson and Cisco provide separate forecasts and put the number at 50 billion devices.

According to IBM, 90 percent of all the data generated by smart devices are never analysed and about 60 percent of this data starts losing its value within milliseconds of being generated.

The IOT concept refers to the use of objects, sensor devices, communication infrastructure, and data processing. The sensor devices can be RFID tags, electronic sensors, scanners, actuators and other sensing devices. The objects in IoT are the physical devices that are uniquely identifiable (for example with IPv6 addressing) and can be accessed through the internet. The data processing unit on the cloud helps decision making and control of the whole system.

Sensors and actuators are physical devices that allow the IOT system to interact with the outside environment. In this case, the term sensor can be broadly defined as a device that provides inputs about the current state of the environment/system. The same way, an actuator can be defined as a device used to induce a change on the environment/system such as the temperature controller of a smart air conditioner.

The data collected by the different sensors can be stored and processed on the edge of the network or on a remote server. Data processing is typically more efficient when performed in close proximity to the physical device/object. This is because no time will be wasted traversing to a remote server and some decisions

need to be made quickly without the need for a remote processing unit. Due to the processing capacity limitations on physical devices, part of the data or a pre-processed data can be sent to the remote server with bigger computing power for more robust and advanced analysis.

The exchange of the data happens based on protocols specifically designed for IOT applications. These applications are suitable for working with the constrained resources (like processing power, memory capacity, bandwidth and battery power) IoT devices have. The IOT is an implementation of more than just a single technology; instead it is a blending of different technologies that operate together.

The following figure 1 shows a simplified example of a typical IOT system. The system comprises three building blocks. The first is a set of IoT devices like sensors, antenna or microcontroller used for collecting data. The second one is the IoT hub used as the connectivity infrastructure on which the communication occurs. The third block represents the business applications, user interfaces or back-end systems on the server.

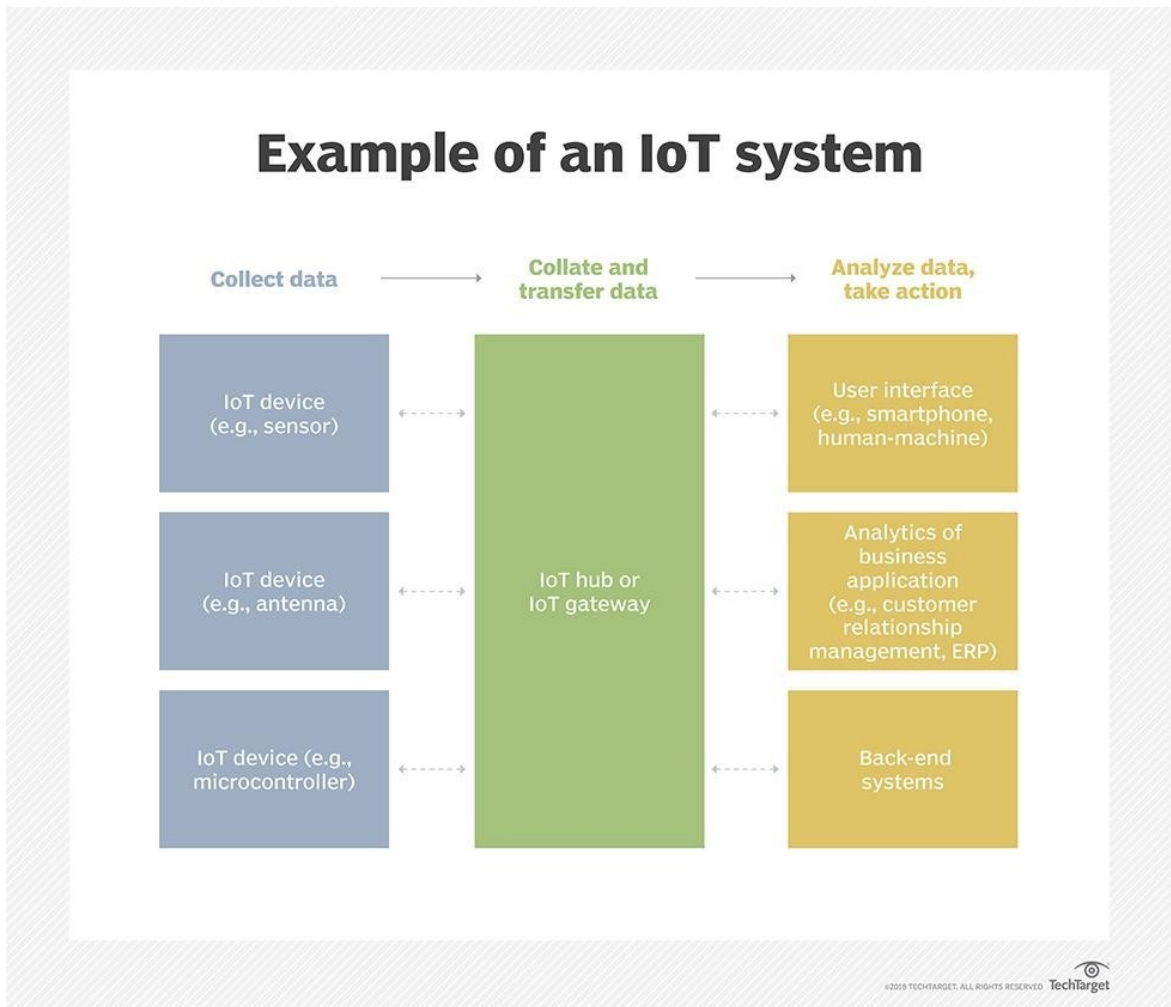


Figure 1: IoT system building blocks

Figure 1 shows a simplified block diagram representation of the internet of things. Among the numerous applications of the internet of things in the real-world are smart cars, wearable devices, smart appliances, home automation, agriculture, industrial IoT (IIoT) and healthcare.

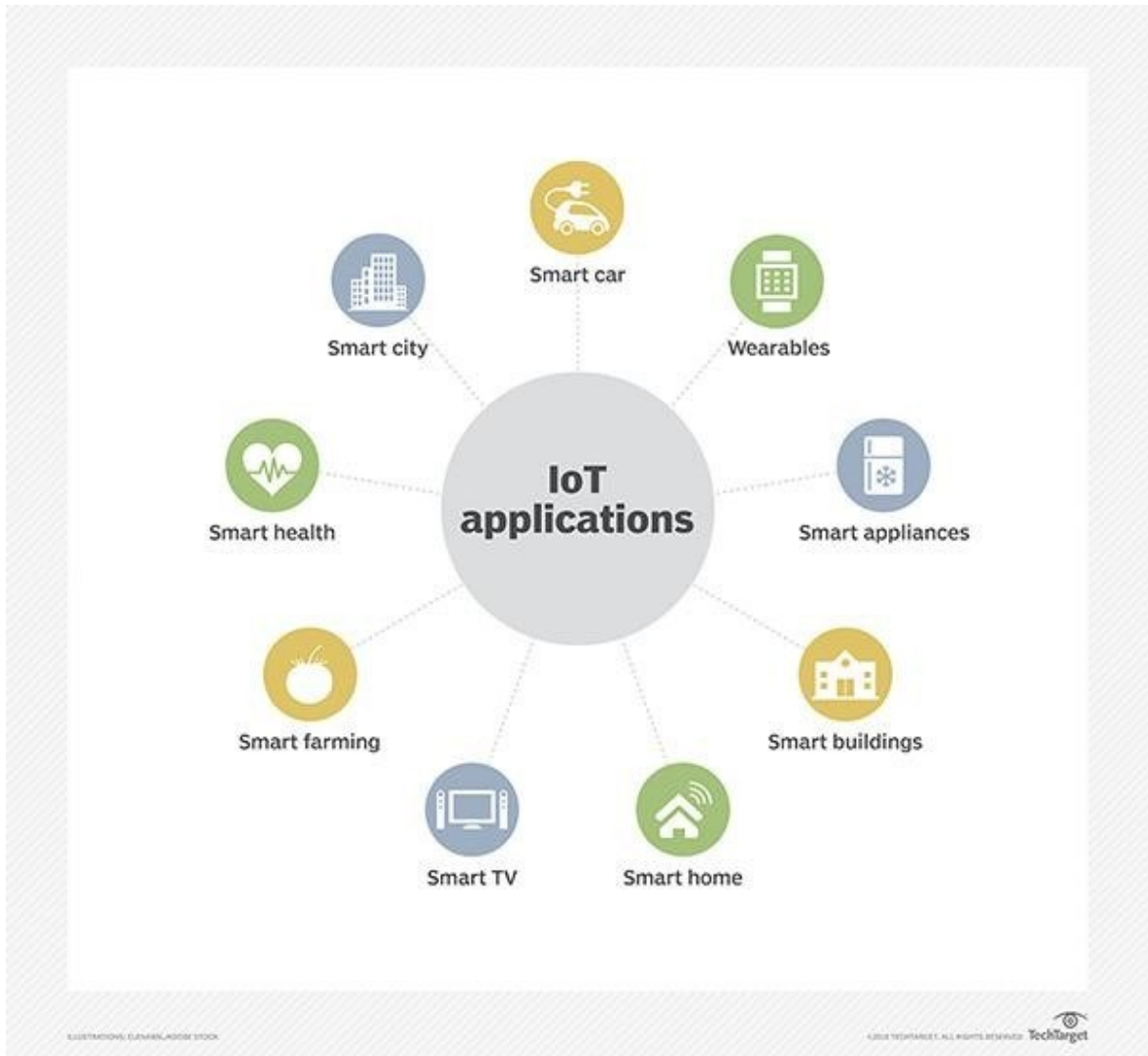


Figure 2: Example Applications of IoT

Figure 2 shows that the areas of IoT applications are diverse and can affect the lives of billions of people.

2.1 Examples of IoT Applications:

Smart cars containing different vehicle sensors can be used to gather real-time data about the working condition of the car, its location, the road conditions, traffic condition, safety condition and others. IoT application in the automotive industry can improve fleet monitoring systems, logistics management, road traffic management, and safety. Preventive maintenance suggested by an IoT system can avoid operations breakdown thereby saving time and money. So a more efficient

and safer running condition for cars can be achieved with proper implementation of IoT.

Wearable devices can collect and analyze data about physical fitness and health condition of a person. Smart appliances like refrigerators can send important information to the user and also run more efficiently using IoT applications. The information can be about the refrigerator's working conditions or the amount/kind of food items inside it.

Smart buildings can increase their energy efficiency using temperature sensors and turning the air conditioner only when the temperature level goes below a certain temperature. Motion sensors can also be employed whereby the lighting bulbs will remain OFF if there is no motion detected in the rooms for a certain amount of time. The moment motion is detected, the lights will turn on or smoke sensors can detect a potential fire accident and send alert message to the appropriate body.

IoT-based smart farming systems gather information about environmental conditions like rainfall level, temperature, humidity, soil moisture and others. Artificial air conditioning systems can then use this data to keep these parameters in a suitable range so that the plants will produce the most yields at the lowest cost. Remote monitoring of patients health condition and analyzing the effectiveness of drugs can be performed using an IoT system in healthcare.

3. IOT ARCHITECTURE

Because of outstanding opportunities IoT promises, more organizations seek for the inclusion of its products in their business processes. However, when it comes to reality, this brilliant idea appears too complicated to be implemented — given the number of devices and conditions needed to make it work. In other

words, the problem of establishing a reliable architecture of Internet of Things inevitably enters the stage.

Among all, to deal with the whole variety of factors affecting IoT architecture, it's easier and more effective to find a reliable provider of IoT solutions. This decision will significantly reduce the number of resources spent on the way. Though it's possible to comprehend the process of creating software, the practical application of its 4 stages contains too many nuances and aspects to be described in simple words. Because of that, use this guide for establishing a proper understanding of what's going on during IoT architecture — but consider referring to the specialist to make this process actually happen. This decision will facilitate getting the needed result and guarantee being a satisfied client of a software development company.

Before revealing the secrets and providing a clear structure of this initiative, it's important to understand what this concept actually means. In essence, IoT architecture is the system of numerous elements: sensors, protocols, actuators, cloud services, and layers. Given its complexity, there exist 4 stages of IoT architecture. Such a number is chosen to steadily include these various types of components into a sophisticated and unified network.

In addition, Internet of Things architecture layers are distinguished in order to track the consistency of the system. This should also be taken into consideration before the IoT architecture process start.

Basically, there are three IoT architecture layers:

1. The client side (IoT Device Layer)
2. Operators on the server side (IoT Getaway Layer)
3. A pathway for connecting clients and operators (IoT Platform Layer)

In fact, addressing the needs of all these layers is crucial on all the stages of IoT architecture. Being the basis of feasibility criterion, this consistency makes the result designed really work. In addition, the fundamental features of sustainable IoT architecture include functionality, scalability, availability, and maintainability. Without addressing these conditions, the result of IoT architecture is a failure.

Therefore, all the above-mentioned requirements are addressed in 4 stages of IoT architecture described here — on each separate stage and after completing the overall building process.

In simple terms, 4-Stage IoT architecture consists of

1. Sensors and actuators
2. Internet gateways and Data Acquisition Systems
3. Edge IT
4. Data center and cloud.

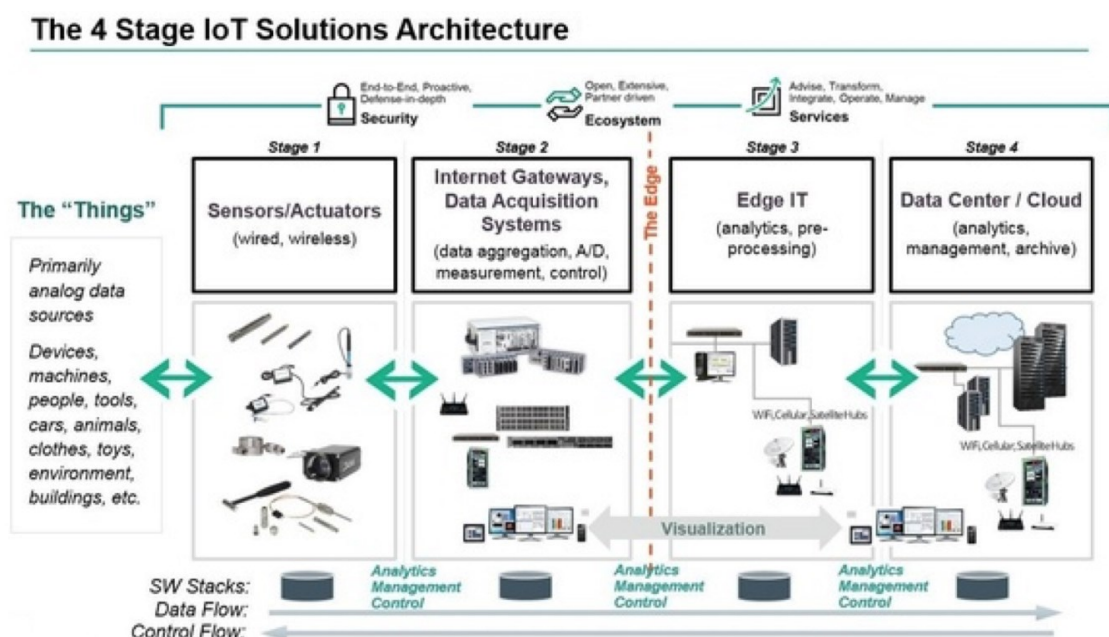


Figure 3: Four Stage IoT Architecture

To get the proper understanding of the main actions and the importance of each stage in this process, refer to the detailed reviews presented below.

Stage 1. Networked things (wireless sensors and actuators)

The outstanding feature about sensors is their ability to convert the information obtained in the outer world into data for analysis. In other words, it's important to start with the inclusion of sensors in the 4 stages of an IoT architecture framework to get information in an appearance that can be actually processed.

For actuators, the process goes even further, these devices are able to intervene the physical reality. For example, they can switch off the light and adjust the temperature in a room. Because of this, sensing and actuating stage covers and adjusts everything needed in the physical world to gain the necessary insights for further analysis.

Stage 2. Sensor data aggregation systems and analogue-to-digital data conversion

Even though this stage of IoT architecture still means working in a close proximity with sensors and actuators, Internet gateways and data acquisition systems (DAS) appear here too. Specifically, the later connect to the sensor network and aggregate output, while Internet gateways work through Wi-Fi, wired LANs and perform further processing.

The vital importance of this stage is to process the enormous amount of information collected on the previous stage and squeeze it to the optimal size for further analysis. Besides, the necessary conversion in terms of timing and structure happens here. In short, Stage 2 makes data both digitalized and aggregated.

Stage 3. The appearance of edge IT systems

During this moment among the stages of IoT architecture, the prepared data is transferred to the IT world. In particular, edge IT systems perform enhanced analytics and pre-processing here. For example, it refers to machine learning and visualization technologies. At the same time, some additional processing may happen here, prior to the stage of entering the data centre.

Likewise, Stage 3 is closely linked to the previous phases in the building of architecture of IoT. Because of this, the location of edge IT systems is close to the one where sensors and actuators are situated, creating a wiring closet. At the same time, the residing in remote offices is also possible.

Stage 4. Analysis, management, and storage of data

The main processes on the last stage of IoT architecture happen in data centre or cloud. Precisely, it enables in-depth processing, along with a follow-up revision for feedback. Here, the skills of both IT and OT (operational technology) professionals are needed. In other words, the phase already includes the analytical skills of the highest rank, both in digital and human worlds. Therefore, the data from other sources may be included here to ensure an in-depth analysis.

After meeting all the quality standards and requirements, the information is brought back to the physical world — but in a processed and precisely analysed appearance already.

Stage 5 of IoT Architecture?

In fact, there is an option to extend the process of building a sustainable IoT architecture by introducing an extra stage in it. It refers to initiating a user's control over the structure — if only your result doesn't include full automation, of course. The main tasks here are visualization and management. After including Stage 5, the system turns into a circle where a user sends commands to sensors/actuators (Stage 1) to perform some actions.

And the process starts all over again.

4 IOT PROTOCOLS

Now, let's get to the specifics of IoT wireless protocols, standards and technologies. There are numerous options and alternatives, but we'll discuss the most popular ones.

4.1 MQTT

MQTT is a lightweight transport protocol for IoT devices. It is also suitable for IoT devices with low and it consumes low power. It is the most widely used IoT protocol nowadays. [22]

MQTT uses publish/subscribe architecture which responds to new events. Subscribers register their interest in certain kinds of data from certain devices. Publishers are the source of new information. MQTT broker plays a central role between the publishers & subscribers and it helps in making sure that a device receives data that it previously registered for.

4.2 DDS

DDS (Data Distribution Service) is an IoT standard for real-time, scalable and high-performance machine-to-machine communication. It was developed by the Object Management Group (OMG). You can deploy DDS both in low-footprint devices and in the cloud. The DDS standard has two main layers:

Data-Centric Publish-Subscribe (DCPS), which delivers the information to subscribers

Data-Local Reconstruction Layer (DLRL), which provides an interface to DCPS functionalities

4.3 AMQP

AMQP (Advanced Message Queuing Protocol) is an application layer protocol for message-oriented middleware environments. It is approved as an international standard. The processing chain of the protocol includes three components that follow certain rules.

Exchange — gets messages and puts them in the queues

Message queue — stores messages until they can be safely processed by the client app

Binding — states the relationship between the first and the second components

4.4 Bluetooth

Bluetooth is a short-range communications technology integrated into most smartphones and mobile devices, which is a major advantage for personal products, particularly wearables. Bluetooth is well-known to mobile users. But not long ago, the new significant protocol for IoT apps appeared — Bluetooth Low-Energy (BLE), or Bluetooth Smart. This technology is a real foundation for the IoT, as it is scalable and flexible to all market innovations. Moreover, it is designed to reduce power consumption.

Standard: Bluetooth 4.2

Frequency: 2.4GHz

Range: 50-150m (Smart/BLE)

4.5 ZigBee

ZigBee 3.0 is a low-power, low data-rate wireless network used mostly in industrial settings. The Zigbee Alliance even created the universal language for the Internet of Things — Dotdot — which makes it possible for smart objects to work securely on any network and seamlessly understand each other.

Standard: ZigBee 3.0 based on IEEE802.15.4

Frequency: 2.4GHz

Range: 10-100m

Data Rates: 250kbps

4.6 WiFi

Wi-Fi is the technology for radio wireless networking of devices. It offers fast data transfer and is able to process large amounts of data.

This is the most popular type of connectivity in LAN environments.

Standard: Based on IEEE 802.11

Frequencies: 2.4GHz and 5GHz bands

Range: Approximately 50m

Data Rates: 150-200Mbps, 600 Mbps maximum

4.7 Cellular

Cellular technology is the basis of mobile phone networks. But it is also suitable for the IoT apps that need functioning over longer distances. They can take advantage of cellular communication capabilities such as GSM, 3G, 4G (and 5G soon). The technology is able to transfer high quantities of data, but the power consumption and the expenses are high too.

Thus, it can be a perfect solution for projects that send small amounts of information.

Standard: GSM/GPRS/EDGE (2G), UMTS/HSPA (3G), LTE (4G)

Frequencies: 900/1800/1900/2100MHz

Range: 35km (GSM); 200km (HSPA)

Data Rates: 35-170kps (GPRS), 120-384kbps (EDGE), 384Kbps-2Mbps (UMTS), 600kbps-10Mbps (HSPA), 3-10Mbps (LTE)

4.8 LoRaWAN

LoRaWAN (Long Range Wide Area Network) is a protocol for wide area networks. It is designed to support huge networks (e.g. smart cities) with millions of low-power devices. LoRaWAN can provide low-cost mobile and secure bidirectional communication in various industries.

Standard: LoRaWAN

Frequency: Various

Range: 2-5km (urban area), 15km (suburban area)

Data Rates: 0.3-50 kbps

5 IOT IMPLEMENTATION FOR AIR CONDITION MONITORING

In this section of the project, the practical application of IoT using the Cayenne IoT platform is demonstrated by building a monitoring system. The system consists of a temperature sensor (DS18B20) and a LORA module connected to a raspberry pi. The temperature sensor gathers the data from the atmosphere and forwards it to the raspberry pi. The LORA module gets the serial data coming from another LORA module wirelessly which is attached to the PicAxe Processor. The raspberry pi CPU temperature and CPU load is also measured. The raspberry pi performs

pre-processing on the data and forwards it to the Cayenne IoT cloud which performs a more robust data analysis.

The special IOT session protocol; MQTT is used when collecting sensory data from devices and sending it to the cloud. A Python application for performing the MQTT session is installed on the raspberry pi. This application also performs the MQTT authentication between the publishing devices and subscriber applications on the cloud. The cayenne cloud acts as a broker for the MQTT process.

The Cayenne dashboard is the graphical user interface displayed on the Cayenne cloud application. The dashboard is available in both the web application and the mobile application (for both Android and IOS devices).

Users can get the access to the data and can control different aspects of the device operation using the cloud applications. It is possible to define rules algorithms on the dashboard using if....else statement. This allows performing a certain task when certain requirements are met. Notification and alert messages can also be configured to receive important information about the system via text message or an email.

The raspberry pi, DSB1820 temperature sensor, LORA module the Cayenne IoT platform and the findings of the practical implementation are discussed in the following subsections.

5.1 The Raspberry Pi

The raspberry pi is a small single board computer used mainly for educational and prototype development purposes. it uses a Broadcom System on Chip (SoC) with an ARM- compatible central processing unit (CPU). All the series up to pi 3 model B+ have on- chip graphics processing unit (GPU). Secure digital (SD) cards are used for storing the operating system, other programs and files.

Figure 4 shows the raspberry pi model B+ v1.2, the different ports, the GPIO pins and the different PCB components.

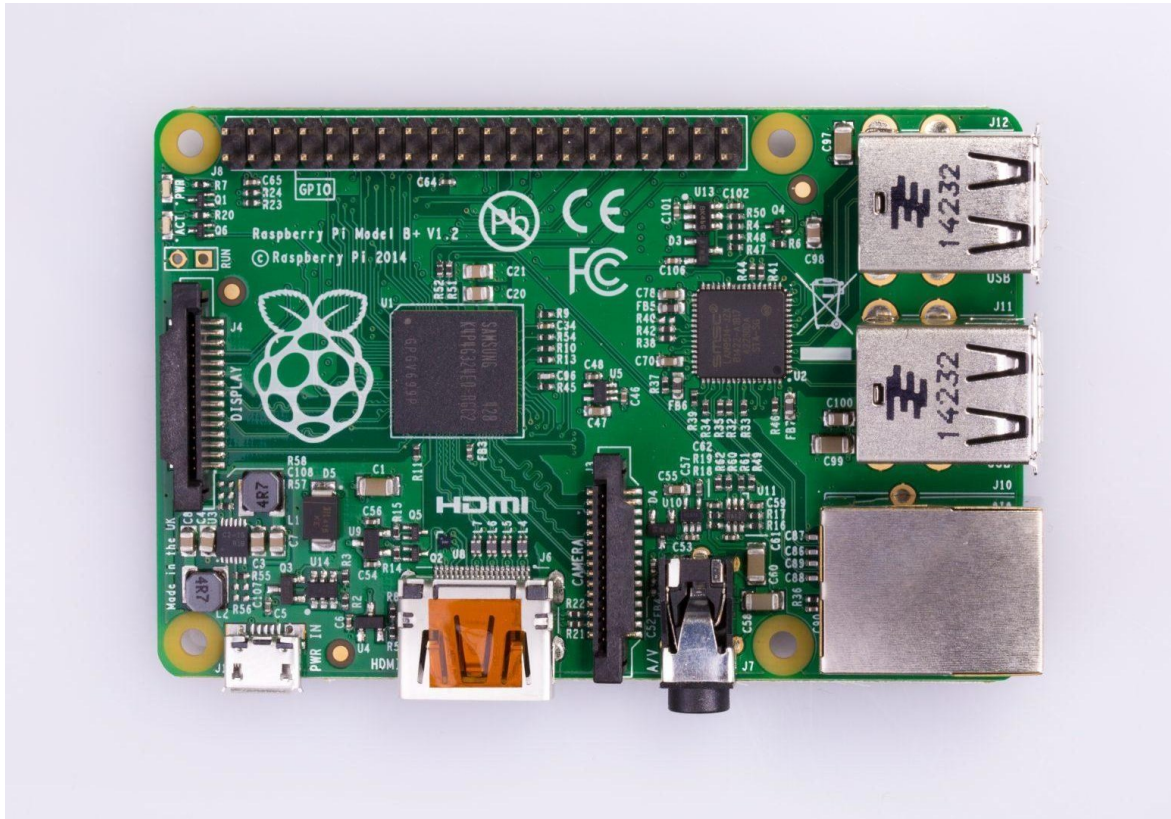


Figure 4: raspberry pi Model B+

The model used in this implementation project was Raspberry pi Model B+ v1.2. It was by developed and released by the Raspberry pi Foundation in 2014. This model has a Broadcom CPU clocked at 700MHz and 512MB RAM. It supports wireless, Bluetooth and Ethernet communication. Its specification can be summarized as follows:

- Basic size, 85mm x 56mm
- Broadcom SoC clocked at 700MHz
- 512MB RAM soldered on top of the chip
- Micro USB power connector
- Dual step-down (buck) power supply for 3.3V and 1.8V
- 5V supply with polarity protection and 2A fuse
- 4 USB ports
- 40 GPIO pins

- HDMI port
- 3.5mm 'headphone' jack
- Camera and DSI(Display Serial Interface) Display connector
- MicroSD card socket
- Ethernet socket

The USB ports can be used to connect any USB-based devices like mouse, keyboard, Wi-Fi dongles, Bluetooth adapters, etc. external monitors can be connected to it using the HDMI connector.

The raspberry pi works with different operating systems; both Linux based operating systems and non-Linux based Operating systems like Windows 10 IoT core. But the Raspberry pi Foundation provides a debian-based Linux distribution called raspbian which can be downloaded from their website for free.

By connecting the raspberry pi to different sensors, and using its different connectivity options, different IoT projects can be implemented. The sensors will send the collected data to the raspberry pi for processing. The raspberry pi can process the data locally and make necessary decisions based on its program or if the data gathered is too much/too complex for the raspberry pi, it can send it to a more powerful remote server for a more detailed data analysis and storage.

In the IoT implementation for this thesis, the raspberry pi was connected to a temperature sensor (DSB1820) and LORA module, thus part of the data processing was performed by the raspberry pi and the remote Cayenne IoT cloud handled the more demanding and complex problems.

Locally the raspberry pi had been connected to the internet through an Ethernet cable. it could also be connected using the Wi-Fi which would require a Wi-Fi dongle connected to the USB port of the raspberry pi. The pi could be accessed using an SSH connection from another computer and programs could be installed and run remotely. SSH remote connection offers more security as the packets are

encrypted during transmission. A command line interface (CLI) also consumes less processing power to the raspberry pi when compared with a graphical user interface.

MQTT was the chosen messaging protocol for the transport layer in this IOT project. The cayenne MQTT API was used to connect the raspberry pi to the remote cayenne cloud. Once the pi is connected, data can be transferred from the sensors and displayed on the cayenne dashboard application. The data can also be displayed in the form of widgets. Commands can also be sent from the cayenne cloud to the local devices.

The cayenne cloud infrastructure acts as a broker in the MQTT architecture. It manages the sensors, actuators, applications and devices. MQTT uses a subscriber-publisher architecture model with the broker acting in the middle.

To connect and communicate with an MQTT broker, an open-source MQTT client called the eclipse paho project was used. The eclipse paho project can be used for different languages like C/C++, Python, Java, JavaScript, Go and C#.

The eclipse paho MQTT Python library provides a client class that enables clients to communicate to the MQTT broker (the cayenne cloud) to publish messages. Different applications (subscribers) can also subscribe to different topics and receive the published messages.

5.2 Temperature Sensor (DS18B20)

The DS18B20 is a 1-wire programmable Temperature sensor from Maxim Integrated. It is widely used to measure temperature in hard environments like in chemical solutions, mines or soil etc. The construction of the sensor is rugged and also can be purchased with a waterproof option making the mounting process easy. It can measure a wide range of temperature from -55°C to $+125^{\circ}$ with a decent accuracy of $\pm 5^{\circ}\text{C}$. Each sensor has a unique address and requires only one pin of the MCU to transfer data so it is a very good choice for measuring

temperature at multiple points without compromising much of your digital pins on the microcontroller.

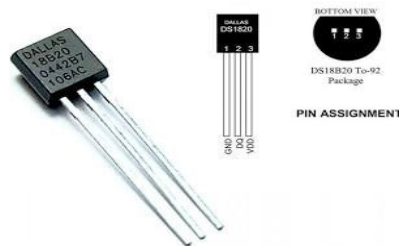


Fig 5: DSB1820 Sensor

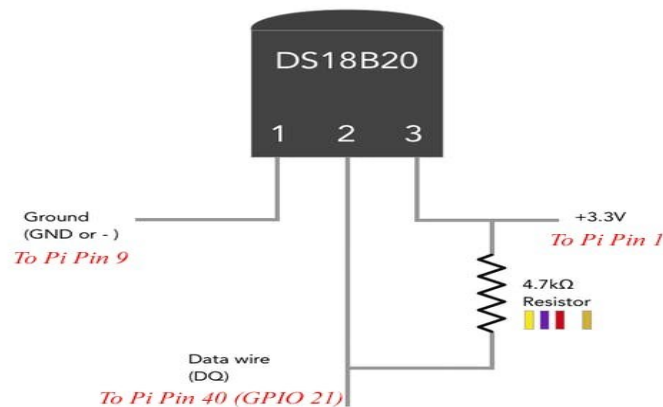


Fig 5: DSB1820 Connections

The pull-up resistor is used to keep the line in high state when the bus is not in use. The temperature value measured by the sensor will be stored in a 2-byte register inside the sensor. This data can be read by the using the 1- wire method by sending in a sequence of data. There are two types of commands that are to be sent to read the values, one is a ROM command and the other is function command. The address value of each ROM memory along with the sequence is given in the datasheet below. You have to read through it to understand how to communicate with the sensor.

5.3 LORA Module (LoStik)

LoStik is an affordable, easy to use, LoRaWAN™ compatible device. It lets IoT (Internet of Things) integrators, network testers, and hobbyists get connected to LoRa® networks faster, diagnose network issues more easily, and build new and exciting connected devices.

LoRa® technology is one of the leading wireless technologies powering the emerging Internet of Things. It is used in tons of cool technologies such as automatic meter reading, smart parking, and even livestock tracking. It has the unique ability to send packets over fairly long distances using minimal power (We're talking kilometers!). LoRa® technology is perfect for devices that are located far away and need to send small amounts of data.

Unfortunately, configuring and deploying these devices can be more challenging than it needs to be. Setting up and testing new networks can also be tiresome and complex. That's where this device comes in.



Fig 6: LoStik (LORA module with USB port)

LoStik technology is ideal for anyone working with existing LoRa® networks or just getting started in IoT. It utilizes the RN2903/RN2483 SOM (System on a Module) chip by Microchip which comes with an intuitive text interface. Just plug your device in to a computer and connect via terminal to start coding.

Hobbyists can connect their Raspberry Pi or PC to any LoRa® gateway and start sending messages or sensor data with minimal fuss or configuration. There is

no easier way to get connected to a LoRaWAN™ network. LoStik is ideal for connecting your Raspberry Pi, BeagleBone, or any PC to the LoRaWAN™ network. It connects over USB leaving your precious Raspberry PI GPIO pins free. If you're looking to build an automated backyard garden, DIY home automation, or a garage door opener, LoStik can help you out. Thus we use this module in our Project for serial input reading.

Specifications:

Connectivity: USB 2.0

Power Consumption: 140 ma typical TX, 20 ma idle (with power LED)

Dimensions: 80 mm x 25 mm x 12 mm (without antenna)

Receiver Sensitivity: down to -146 dBm

TX Power: adjustable up to +18.5 dBm

Range: up to 15 km coverage in suburban and up to 5 km coverage in urban areas

LoStik uses a simple ASCII interface. You can configure it and send commands over serial / COM port. Microchip provides a detailed command reference RN2903, RN2483. The radio can be used in packet mode sending packets between nodes. This provides simplex, single-channel communications. If you want to speak LoRaWAN you need a gateway or access to one. Some countries in Europe provide nation-wide coverage.

5.3 The Cayenne IoT Platform

The Cayenne IoT platform was created by an IoT Solutions company called myDevices. myDevices was founded in 2013 by Kevin Bromer who also had been the CEO of the company. The cayenne mobile apps can be used to monitor and control remotely located physical devices. The Cayenne dashboard contains widgets that can be configured to visualize data, create rules, schedule events, and make notification rules and more.

The Cayenne platform can be used to connect different kinds of sensors, lights, motors, valves, relay and generic actuators. It also supports the Raspberry pi, Arduino, ESP8266, LoRa devices and other development boards.

For this specific thesis, the Raspberry pi was connected to the Cayenne IoT platform. A temperature sensor (DSB1820) and LoStik was connected to the Raspberry pi and was used to monitor temperature as well as Serial Data reading. Along with that, the CPU temperature and CPU load was also made displayed to the platform.

The Raspbian or any other Linux-based operating system needs to be installed before starting installation of Cayenne on the Raspberry pi. Cayenne installations take place in 4 steps:

1. Installing libraries
2. Installing agent
3. Installing software
4. Installing drivers

Once the installation is done, the Cayenne Online dashboard will appear automatically. It shows information about the Raspberry pi model, the Operating system, CPU, RAM and Disk usage level.

Once sensors are connected to the Raspberry pi and the Raspberry pi is connected to the Cayenne cloud platform, advanced rules can be made on the dashboard. These rules can be about what needs to be performed when a certain reading level is reached. One simple example can be to turn ON a fan once the room temperature reading reaches a certain level.

The notification engine can send notifications through text messages or email. The rules for the notification can be configured from the dashboard as

required. Events scheduling can be used to perform certain actions during certain time intervals without the need for external action.

The Cayenne IoT cloud uses Cayenne MQTT API to connect devices. This lightweight protocol enables smooth transfer of data. MQTT is designed especially for IoT applications where there is a constraint of computing resources, battery power and bandwidth.

The Cayenne cloud acts as a broker and manages the different sensors and actuators (publishers). the applications running on the cloud act as subscribers that will receive data about events that they subscribed to.

The client devices used in this Project interact with the MQTT broker (Cayenne cloud) using the Eclipse Paho project. The Eclipse Paho project offers open-source MQTT clients for C, C++, Python, Java, JavaScript, Go and C#. It comes bundled with the Cayenne libraries making it easy to use for this application.

When establishing MQTT connection between devices and the Cayenne cloud, the MQTT credentials; MQTT Username, MQTT Password and Client ID are used. The values for each credentials need to be included in the code (Python in this case) on the raspberry pi. When the program with the MQTT parameters is compiled and run on the raspberry pi, an MQTT connection is established and data transfer starts. When new devices are added, these MQTT credentials from the online dashboard need to be used again. This ensures that the device is connected to the correct account.

The Cayenne Cloud API enables interaction and development of applications using the mydevices IoT RESTful API. the REST API offered by mydevices uses OAuth2 protocol which offers increased security.in order to utilize the Cayenne Cloud API, the App Key and App Secret need to be obtained after signing up for the service. After using these applications credentials, the custom applications can connect to the Cayenne cloud API. Cayenne offers authorization

and Authentication features to ensure data security and all transport communications are encrypted using TLS/SSL endpoints.

5.4 Cayenne Practical Implementation


This section will describe the steps involved in connecting the DSB1820 (Temperature) sensor, LoStik(LORA Module) to the raspberry pi, installing the Cayenne software on the raspberry pi and then setting up the connection between the raspberry pi and the Cayenne cloud. Cayenne application dashboard features are also shown at the end.

Once the connection is setup, then sensory data can be uploaded to the Cayenne cloud using the Cayenne MQTT Python Library. The Cayenne dashboard application allows defining rules algorithm, scheduling important events for devices and sending alert (notification) to the user.

The following steps were followed for this implementation:

1. The first step will be creating a free account on mydevices Cayenne. [32] Cayenne offers paid services which are preferable for more scalable solutions with more features added.
2. Register/connect the raspberry pi to the account created earlier. This can be done either through a smartphone or the SSH terminal to the raspberry pi. The SSH terminal option was used in this thesis, and the cayenne software was installed.

Two ways to install myDevices Cayenne on your Pi



myDevices Cayenne Smartphone App
Our smartphone app can be used to automatically locate and install myDevices Cayenne on your Pi.

Download on the **App Store**

ANDROID APP ON **Google Play**

Coming soon. [Notify me](#)



Terminal / SSH
To download and install myDevices Cayenne on your Pi, use the Terminal on your Pi or SSH. Run the following commands:

```
wget https://cayenne.mydevices.com/dl/rpi_...sh
sudo sh rpi_...sh -v
```

[Back](#)

Figure 7: Cayenne Installation and connection

As it can be seen from Figure 7, there are two options of installing myDevices Cayenne on the raspberry pi, using mobile phone application or through the SSH terminal.

3.Connecting the DHT11 sensor to the raspberry pi

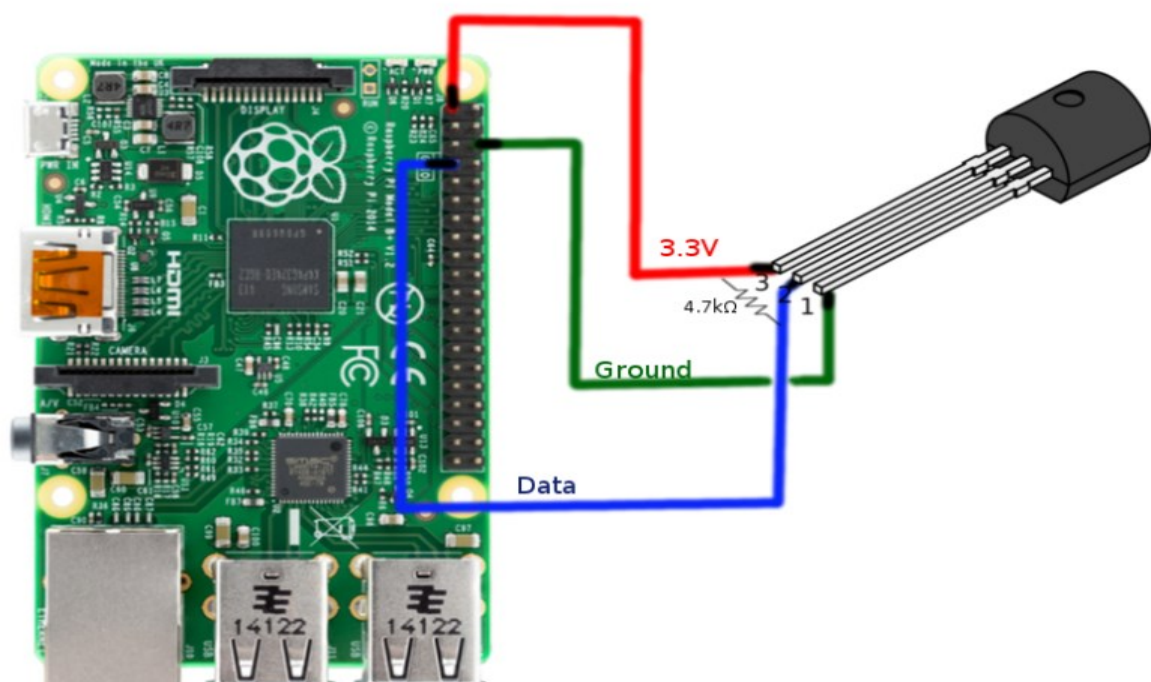


Figure 8: DSB1820 to Raspberry Pi

4.Connecting LoStik to the raspberry pi



Figure 9: DSB1820 to Raspberry Pi

5.Cayenne Installation, Mqtt details Collection, Configuration and Dashboard.

Once we got sign in to cayenne site, We should select CayenneAPI(Bring your own thing.Here we get our MQTT username, password, client id and server name.

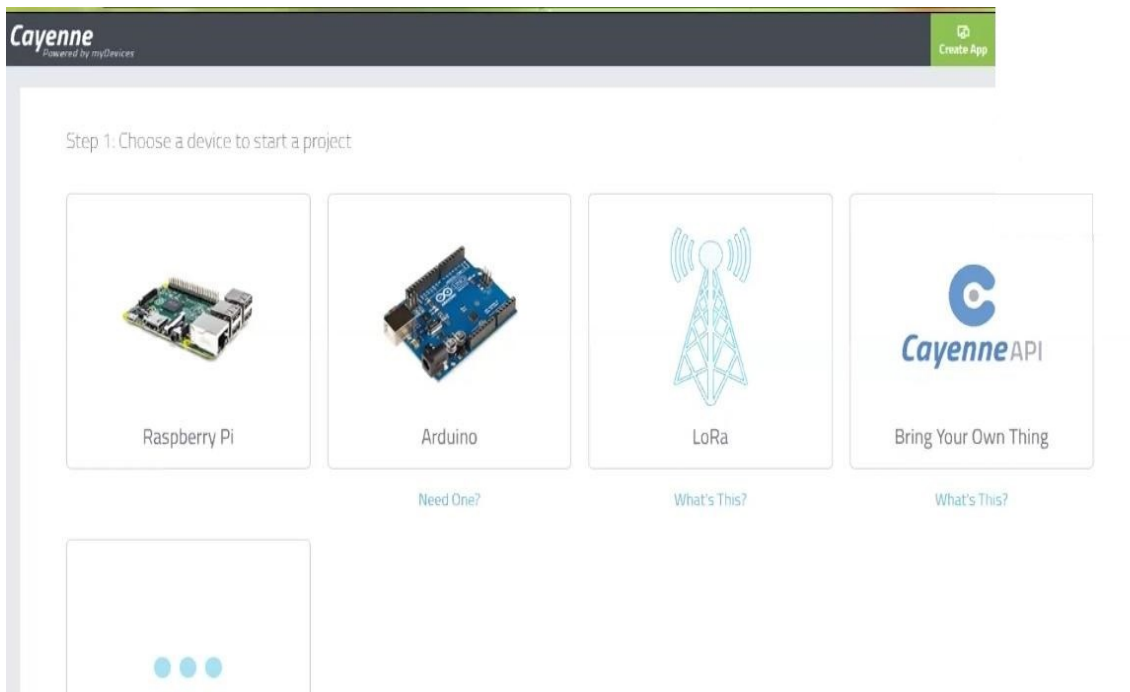


Figure 10: Cayenne Sign in Window

MQTT USERNAME:
[Redacted] [Copy]

MQTT PASSWORD:
[Redacted] [Copy]

CLIENT ID:
[Redacted] [Copy]

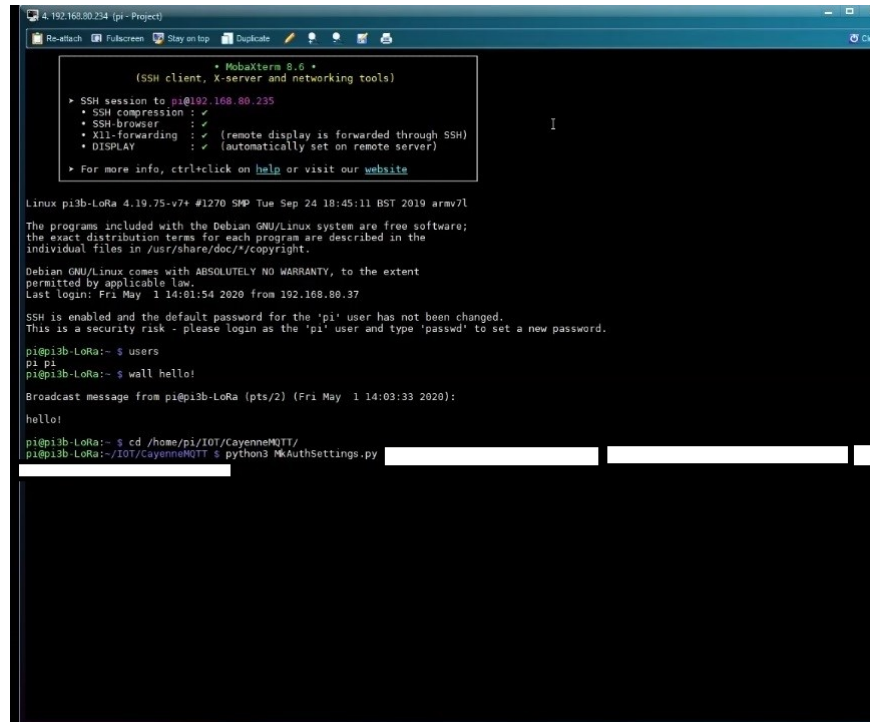
MQTT SERVER: mqtt.mydevices.com MQTT PORT: 1883

NAME YOUR DEVICE (optional):
Device dajO

Figure 11: MQTT Credentials

The credentials are copied to the clipboard. This is used in MKAuthsettings.py python program thus connecting our main

program(Thread_to_MQTT.py). After adding credentials, the main program is started.



```

Linux pi3b-LoRa 4.19.75-v7+ #1270 SMP Tue Sep 24 18:45:11 BST 2019 armv7l
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri May 1 14:01:54 2020 from 192.168.80.37
SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password.

pi@pi3b-LoRa:~$ users
pi pi
pi@pi3b-LoRa:~$ wall hello!
Broadcast message from pi@pi3b-LoRa (pts/2) (Fri May 1 14:03:33 2020):
hello!

pi@pi3b-LoRa:~$ cd /home/pi/IOT/CayenneMQTT/
pi@pi3b-LoRa:~/IoT/CayenneMQTT$ python3 MKAuthSettings.py

```

Figure 12: MKAuthSettings.py in terminal

Once program is started the values will be received in the cayenne platform. This should be added to the dash board using the + icon.

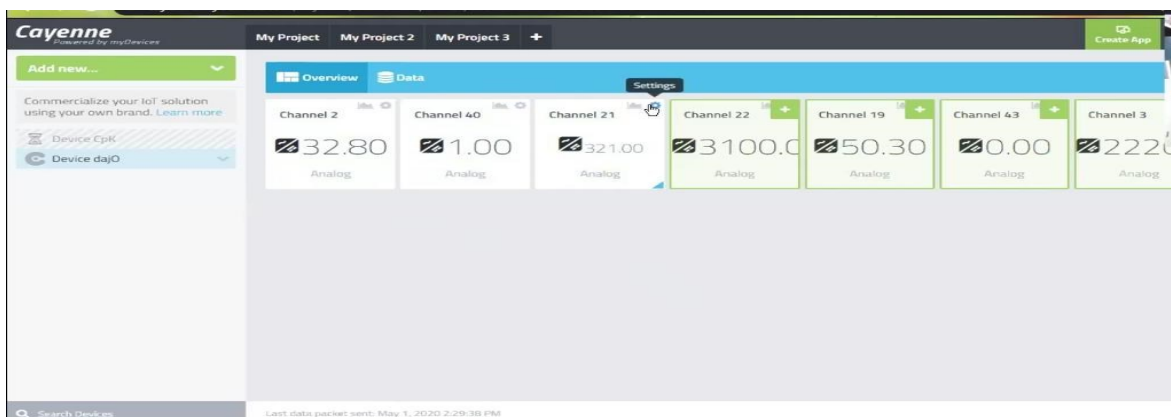


Fig 13: Device Information Received On Raspberry Pi

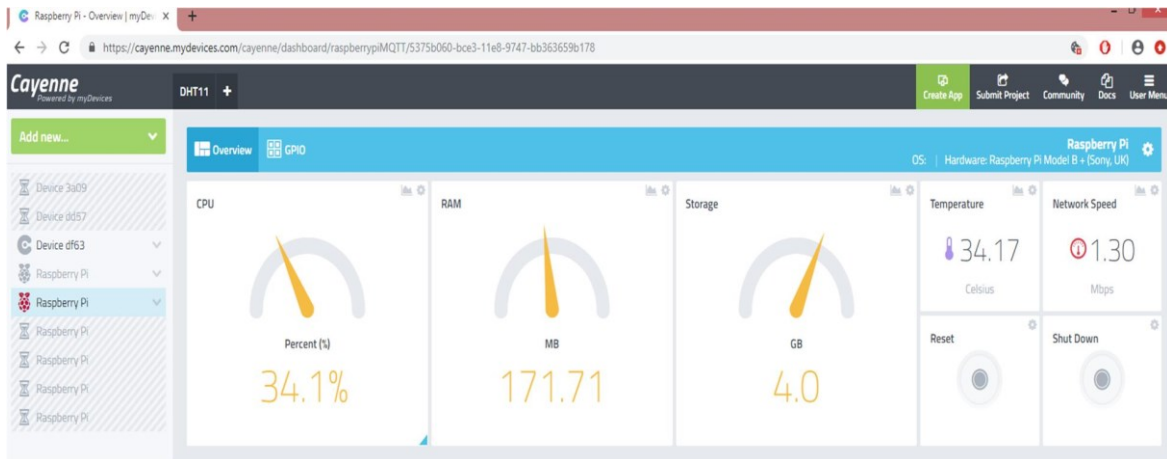


Fig 14: Dashboard Model with Titles and Unit Changed

Figure 14 shows basic information about the raspberry pi like the Operating System, device model, and the specifications of the device. The temperature and Serial input values are also displayed.

The result can also be visualized in graphical format as shown below:

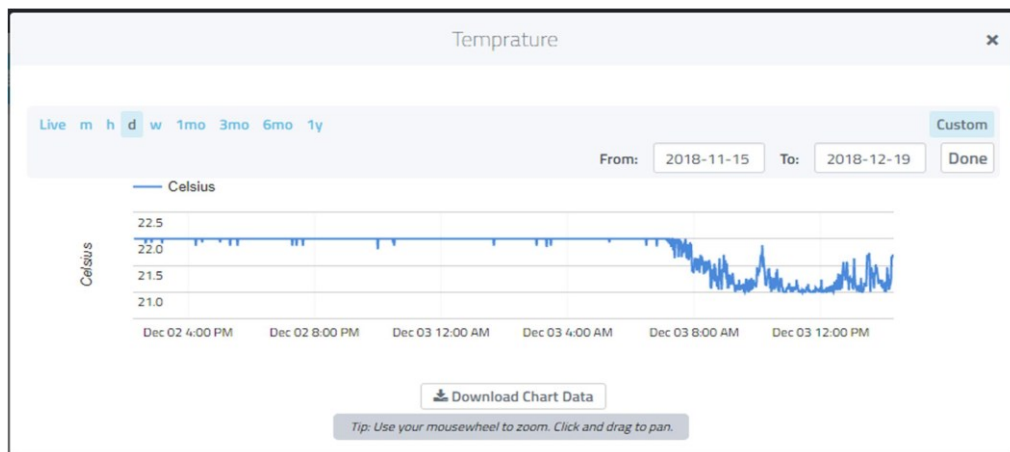


Figure 15: Graphical output of Temperature values

6 PROGRAM EXPLANATION

Programs and all relevant data required for the project can be found in <https://github.com/SteveCossy/IOT>. The LoRaReAd Git folder contains the Files created to get data direct from LoRa module to Pi. The CayenneMQTT & WebPy Git folder contains all the files to get MQTT data from Cayenne in to a csv file, then to geojson (CayenneMQTT) and on to an OpenStreetMap page (WebPy). This code uses Python and Javascript to create web pages for visualization of GPS data previously read in by the LoRaReAd project. The PicAxe Git folder contains the BASIC code that going into the PicAxe 08M2 processor used for most of our sensors and some gateways. The Test Git folder contains the short Python scripts for test concepts and other generally useful standalone things. The MQTTupload Git folder contains the major accumulation of working files across these projects.

6.1 LoRaReAd Git folder Overview

1)Thread_to_MQTT.py Read data from a LoRa Module serial connection (GPIO pins or USB) or from onboard sensors. Publish all data to an MQTT broker.

- This code creates a thread for each sensor.
- Serial data is read as it arrives from sensors.
- Other data is read on a frequency specified by hard-coded constants.
- Error handling is also managed in this code. Each thread must have it's own error handling routines which catch code exceptions, check them, and (optionally) continue. Errors are referred to the ProcessError function. If there are too many (hard-coded threshold) exceptions within a specified (also hard-coded) time, then this function returns False, which effectively stops the thread it was called from. Otherwise the function returns True.
- All the hard-coded constants in this code should be moved to a TOML file so they can be changed without having to change any code.

2)SensorLib.py contains all functions required to read the serial and onboard sensors.

3)MQTTUtils.py contains other common routines to make these Python scripts work. These scripts, and others, use various text file (TOML or csv) to store data externally. These files are described here.

4)LoRa_to_MQTT.py Old version of Thread_to_MQTT.py which does very similar functions, but doesn't use threads. Only kept for reference.

6.1.1 Thread_to_MQTT.py

```
1  # Based on ideas from: https://realpython.com/intro-to-python-threading/
2  # Steve Cosgrove 28 March 2020
3  import logging
4  import threading
5  import time
6  import os
7  import sys
8  import cayenne.client
9  import datetime
10 import toml
11 import string
```

Fig 16 : Code 1st part

Import in python is similar to `#include header_file` in C/C++. Python modules can get access to code from another module by importing the file/function using `import`.

The "**import logging**" module in Python is a ready-to-use and powerful module that is used by most of the third-party Python libraries, so you can integrate your log messages with the ones from those libraries to produce a homogeneous log for your application. With the logging module imported, you can use something called a "logger" to log messages that you want to see. By default, there are 5 standard levels indicating the severity of events. Each has a corresponding method that can be used to log events at that level of severity. The defined levels, in order of increasing severity, are the following:

- DEBUG
- INFO
- WARNING
- ERROR
- CRITICAL

The "**import threading**" module in Python allows you to have different parts of your program run concurrently and can simplify your design. A thread is a separate flow of execution. This means that your program will have two things happening at once. But for most Python 3 implementations the different threads do not actually execute at the same time: they merely appear to. Here this code creates a thread for each sensor.

The "**import time**" module provides a function for getting local time from the number of seconds elapsed since the epoch called `localtime()` . In a computing context, an epoch is the date and time relative to which a computer's clock and timestamp values are determined.

The "**os**" and "**sys**" modules provide numerous tools to deal with filenames, paths, directories. The `os` module contains two sub-modules `os.sys` (same as `sys`) and `os.path` that are dedicated to the system and directories; respectively. Whenever possible, you should use the functions provided by these modules for file, directory, and path manipulations.

The "**import cayenne.client**" module is the main client class for connecting to Cayenne and sending and receiving data.

The "**datetime**" module supplies classes for manipulating dates and times.

The "**toml**" module takes in a string containing standard TOML-formatted data and returns a dictionary containing the parsed data.

The "**string**" module provides additional tools to manipulate strings.

```
12 # import requests, glob, uuid, traceback # (Unnecssesary things I used to import)
13 from SensorLib import GetWirelessStats
14 from SensorLib import GetSerialData
15 from SensorLib import ReadTemp
16 from MQTTUtils import Save2Cayenne
17 from MQTTUtils import Save2CSV
18 from MQTTUtils import DataError
19 from MQTTUtils import PiSerial
20 # from MQTTUtils import ProcessError
21 from gpiozero import CPUtemperature
22 from gpiozero import DiskUsage
23 from gpiozero import LoadAverage
24
```

Fig 17 : Code 2nd part

SensorLib contains all functions required to read the serial and on board sensors. The GetWirelessStats can get wireless statistics from SensorLib.py program created. Similarly serial data can be obtained with the help of GetSerialData module.

MQTTUtils contains other common routines to make these Python scripts work. The MQTTUtils is a library of routines to assist with using getting IoT data, then using MQTT protocol to manipulate it, particularly for use in the Cayenne Cloud . From that different modules named Save2Cayenne, Save2CSV, DataError, ReaTemp and ProcessError are imported.

The Save2Cayenne module is used to add various paramteres like CPU temperature, External Temperature , Network data status etc to Cayenne platform. Here mapping is done. Save2CSV module is used to create a .csv file which contains the path for file, Cayenne device ID, The unique letter used to distinguish different sensors, and the data going to write. The ReadTemp is a function that grabs the raw temp data from a single DS18B20. The ProcessError is used to save message to a file and cayenne. gpiozero is a simple interface to GPIO devices with Raspberry Pi. Form this CPUtemperature module is imported to get the value of raspberry pi's CPU temperature.

```

29 # The callback for when a message is received from Cayenne.
30 def on_message(client, userData, message):
31 # based on https://developers.mydevices.com/cayenne/docs/cayenne-mqtt-api/#cayenne
32 # global COUNTER
33     print("message received: " ,str(message.payload.decode("utf-8")))
34
35 def on_connect(client, userData, flags, rc):
36     print("Connected with result code "+str(rc))
37
38 def ReadTempThread(Freq,CSVPath,ClientID,client):
39     DoRead = True
40     while DoRead :
41         try:
42             Value = ReadTemp()
43 # logging.info("Temp Loop: %s", Value)
44             Channel = 'ExtTemp'
45             Save2CSV (CSVPath, ClientID, Channel, Value)
46             Save2Cayenne (client, Channel, Value, 1)
47             time.sleep(Freq)
48         except :
49             Message = "Exception reading Onboard Temperature"
50             CSV_Message = Message
51             DoRead = ProcessError(CSVPath, ClientID, '', CSV_Message, Message)
52
53
54 def ReadDiskThread(Freq,CSVPath,ClientID,client):
55     DoRead = True
56     while DoRead :
57         try:
58             Value = round( DiskUsage().value,2)
59 # logging.info("Disk Loop: %s", Value)
60             Channel = 'DiskAvg'
61             Save2CSV (CSVPath, ClientID, Channel, Value)
62             Save2Cayenne (client, Channel, Value, 1)
63             time.sleep(Freq)
64         except :
65             Message = "Exception reading Disk Usage"
66             CSV_Message = Message
67             DoRead = ProcessError(CSVPath, ClientID, '', CSV_Message, Message)
68

```

Fig 18: Code 3rd part

The paho mqtt client `class` has several methods. The main ones are:

- connect() and disconnect()
- subscribe() and unsubscribe()
- publish()
- Each of these methods is associated with a callback.

First is importing the Client Class then creating a client instance after that connecting to a broker or server. Before you can publish messages or subscribe to topics you need to establish a connection to a broker. To do this use the connect method of the Python mqtt client. Along with the section in figure 16, there shows two threads to read temperature from external sensor connected and also another thread to read CPU temperature. Also the message received is decoded to utf-8 before using.

The function ReadTempThread is for reading temperature data. This function takes 4 parameters

- CSVPath - It is the path to csv file (line 297)
- ClientID - ID of CayenneMQTT client
- Client - CayenneMQTTClient()

This function reads the temperature value from the sensor and value is saved into a csv file along with client details. The CSV file stores this data in a tabular form, such as a spreadsheet or database for later use or current reference. The temperature value is also sent to cayenne. In cayenne, details of Raspberry pi signals is defined like CPU temperature, Status, External temperature etc. If an error occurs during the above case, then a process named ProcessError() will be called along with the message "Exception reading Onboard Temperature".

The function ReadDiskThread defines the current Disk usage. Here we use DiskUsage() which returns the current disk usage in percentage and this function resides in gpiozero library. The gpiozero library is made to work on all Raspberry Pi models, and is compatible with both Python 2 and Python 3. gpiozero is a simpler way to interact with physical components with Raspberry pi, a new friendly Python API. Value of disk usage is stored in csv file and is send to cayenne for processing.

```

70 def ReadLoadThread(Freq,CSVPath,ClientID,client):
71     DoRead = True
72     while DoRead :
73         try:
74             Value = LoadAverage().load_average
75             # logging.info("Load Loop: %s", Value)
76             Channel = 'LoadAvg'
77             Save2CSV (CSVPath, ClientID, Channel, Value)
78             Save2Cayenne (client, Channel, Value, 1)
79             time.sleep(Freq)
80             # raise Exception('Test exception at line 79 of Thread2MQTT.py')
81         except :
82             Message = "Exception reading Load Average"
83             CSV_Message = Message
84             DoRead = ProcessError(CSVPath, ClientID, '', CSV_Message, Message)
85
86 def ReadCPUThread(Freq,CSVPath,ClientID,client):
87     DoRead = True
88     while DoRead :
89         try:
90             Value = CPUTemperature().temperature
91             # logging.info("CPU Loop: %s", Value)
92             Channel = 'CPUtemp'
93             Save2CSV (CSVPath, ClientID, Channel, Value)
94             Save2Cayenne (client, Channel, Value, 1)
95             time.sleep(Freq)
96         except :
97             Message = "Exception reading CPU Temperature"
98             CSV_Message = Message
99             DoRead = ProcessError(CSVPath, ClientID, '', CSV_Message, Message)
100

```

Fig 19: Code 4th part

This function `ReadLoadThread` uses `LoadAverage()` which is a function imported from `gpiozero` library. It provides information about the device that gets activated when the CPU load average exceeds the threshold value. load average value is stored in csv file and is send to cayenne for processing.

This function `ReadCPUThread` uses `CPUTemperature()` which is a function imported from `gpiozero` library. It provides information about device that gets activated when the CPU temperature exceeds the threshold value. The Parameter temperature returns the current CPU temperature in degrees celsius. Temperature value is stored in csv file and is send to cayenne for processing.

```

102 def ReadWifiThread(Freq,CSVPath,ClientID,client):
103     DoRead = True
104     while DoRead :
105         try:
106             Value = GetWirelessStats()
107             # logging.info("Wifi Loop: %s", Value)
108             Link = Value['wlan0']['link']
109             Channel = 'WifiLnk'
110             Save2CSV (CSVPath, ClientID, Channel, Link)
111             Save2Cayenne (client, Channel, Link, 100)
112             Level = Value['wlan0']['level']
113             Channel = 'WifiLvl'
114             Save2CSV (CSVPath, ClientID, Channel, Level)
115             Save2Cayenne (client, Channel, Level, 100)
116             time.sleep(Freq)
117         except :
118             Message = "Exception reading Wifi Data"
119             CSV_Message = Message
120             DoRead = ProcessError(CSVPath, ClientID, '', CSV_Message, Message)
121
122
123 def ReadGPIOData(CSVPath,ClientID,client):
124     # Read serial data from GPIO pins
125     #
126     # Define the PicAxe Divisors
127     DivisorDict = dict.fromkeys(string.ascii_uppercase)
128     for key in DivisorDict :
129         DivisorDict[key] = 1
130     DivisorDict['A'] = 10 # Soil Moisture
131     DivisorDict['B'] = 10 # Temperature
132     DivisorDict['S'] = 10 # Kihi-02 Moisture
133     DivisorDict['T'] = 10 # Kihi-02 Temperature
134     DoRead = True
135

```

Fig 20: Code 5th part

The Function defines wifi status.value of wifi status is taken from GetWirelessStats() which is a function imported from sensor library. GetWirelessStats() works on /proc/net/wireless which acts a file and is designed to give wireless specific statistics on each wireless interface in the system. The value is used for linking system with WIFI.

The function reads serial data from gpio pins. details on serial data resides in GetSerialData() which is defined in Sensor Library. The serial data is about Device name, Client ID, Channel used, RSSI etc. When Received Signal Strength gets weaker and the wireless data rates get slower, this leads to a lower overall data throughput.

```

184 def ReadUSBData(CSVPath,ClientID,client):
185     # Read serial data from USB port
186     #
187     # Define the PicAxe Divisors
188     DivisorDict = dict.fromkeys(string.ascii_uppercase)
189     for key in DivisorDict :
190         DivisorDict[key] = 1
191     DivisorDict['A'] = 10 # Soil Moisture
192     DivisorDict['B'] = 10 # Temperature
193     DivisorDict['S'] = 10 # Kihi-02 Moisture
194     DivisorDict['T'] = 10 # Kihi-02 Temperature
195
196     USBport = 'USB0'
197
198     if (USBport in PiSerial() ):
199         DoRead = True
200         logging.info("USB port found: "+ USBport )
201     else:
202         Message = "Error looking for "+USBport
203         CSV_Message = Message
204         DoRead = ProcessError(CSVPath, ClientID, '', CSV_Message, Message)
205         # Will try to read (twice) data anyway
206
207     SerialDetails = {
208         "DeviceName": "/dev/tty"+USBport,
209         "ModuleType": "DRF126X",
210         "BAUDrate": "2400",
211     }

```

Fig 20: Code 6th part

The function def ReadUSBData(CSVPath,ClientID,client) will read serial data from USB port. The data about temperature and soil moisture is collected to PicAxe processor and these data's must be fed to raspberry pi and it is done by serial communication.

In raspberry pi, we should connect USB to TTL converter and connect it to receive and transmit data's with raspberry. For using serial communication in

raspberry pi, piserial package must be installed (line 198), then we must enable serial communication in raspberry pi. It is done by editing config.txt file present in root directory of raspberry pi. Then after receiving data, status of data is evaluated. If status is 0 it means no data is read else if status other than zero will write the data to csv and cayenne.

```

246 def ProcessError(CSVPath, ClientID, CayClient, CSV_Message, Message):
247     # Save Message to a file and Cayenne
248     global LastError
249     CurrentTime = datetime.datetime.now().isoformat()
250     CSVPathFile = Save2CSV (CSVPath, ClientID, 'Exception', CSV_Message)
251     CurrentTime = datetime.datetime.now().isoformat()
252     LogPathFile = logging.getLoggerClass().root.handlers[0].baseFilename
253
254     ErrorTime = datetime.datetime.now()
255     ErrorGap = ErrorTime - LastError['time']
256     # logging.info( LastError )
257     # logging.info( ErrorGap )
258
259     if ErrorGap.days > 0: # Ages since last error
260         Continue = True
261         ResetCount = True
262     elif ErrorGap.seconds > LastError['period']: # OK time since last error
263         Continue = True
264         ResetCount = True
265     elif LastError['count'] < LastError['threshold']: # Still counting
266         LastError['count'] += 1
267         Continue = True
268         ResetCount = False
269     else: # We have a problem
270         Continue = False
271         ResetCount = True
272
273     if ResetCount:
274         LastError = {
275             'time' : ErrorTime,
276             'count' : 0
277         }
278

```

Fig 21: Code 7th part

Error handling is also managed in this code. The function `def ProcessError(CSVPath, ClientID, CayClient, CSV_Message, Message)` is used for

error handling. Each thread must have its own error handling routines which catch code exceptions, check them, and (optionally) continue. Errors are referred to the `ProcessError` function. If there are too many (hard-coded threshold) exceptions within a specified (also hard-coded) time, then this function returns `False`, which effectively stops the thread it was called from. Otherwise the function returns `True`.

`ProcessError` is the error handling function, where current time, path to csv file, error time and gap is calculated and determined. If error does not occur for long time we can continue the process what we are currently doing. If error is occurring continuously we should stop the process and terminate the execution and the exception is raised with pre-defined message and displays last message occurred for the reference. Error status is sent to cayenne.

7 ALGORITHM AND BLOCK DIAGRAM

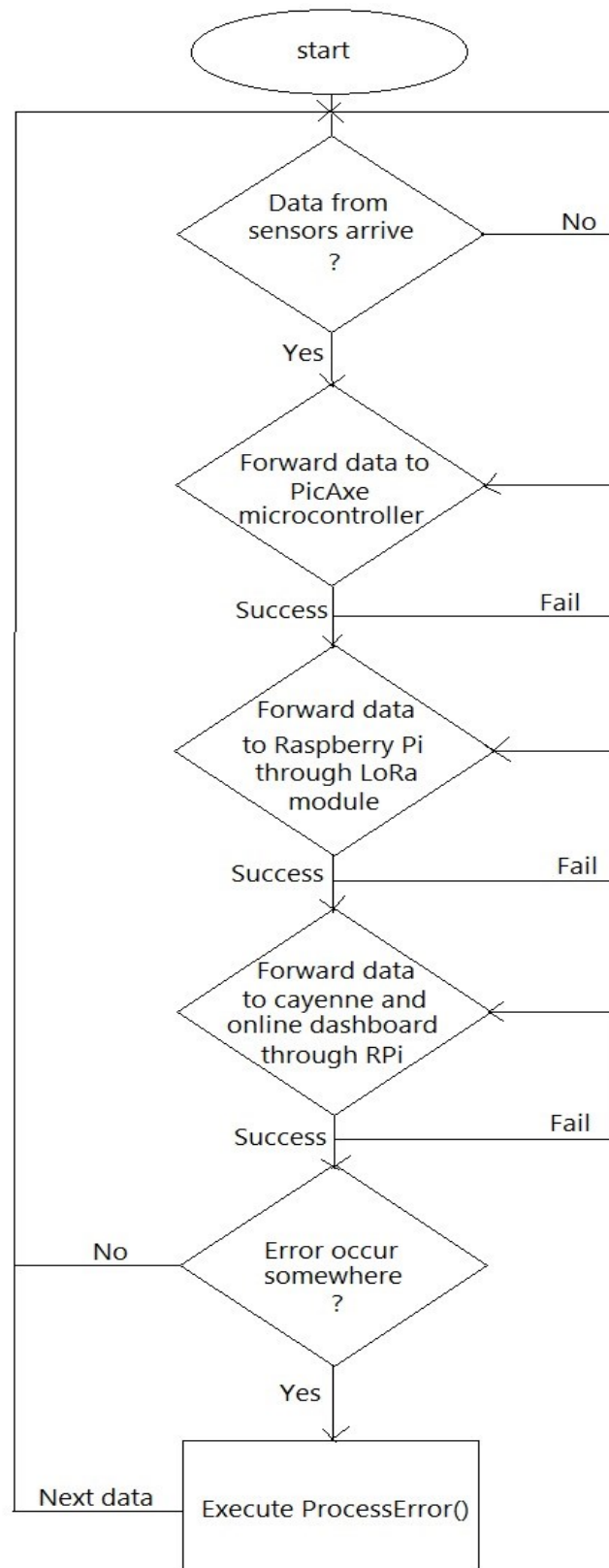


Fig 22 : Algorithm

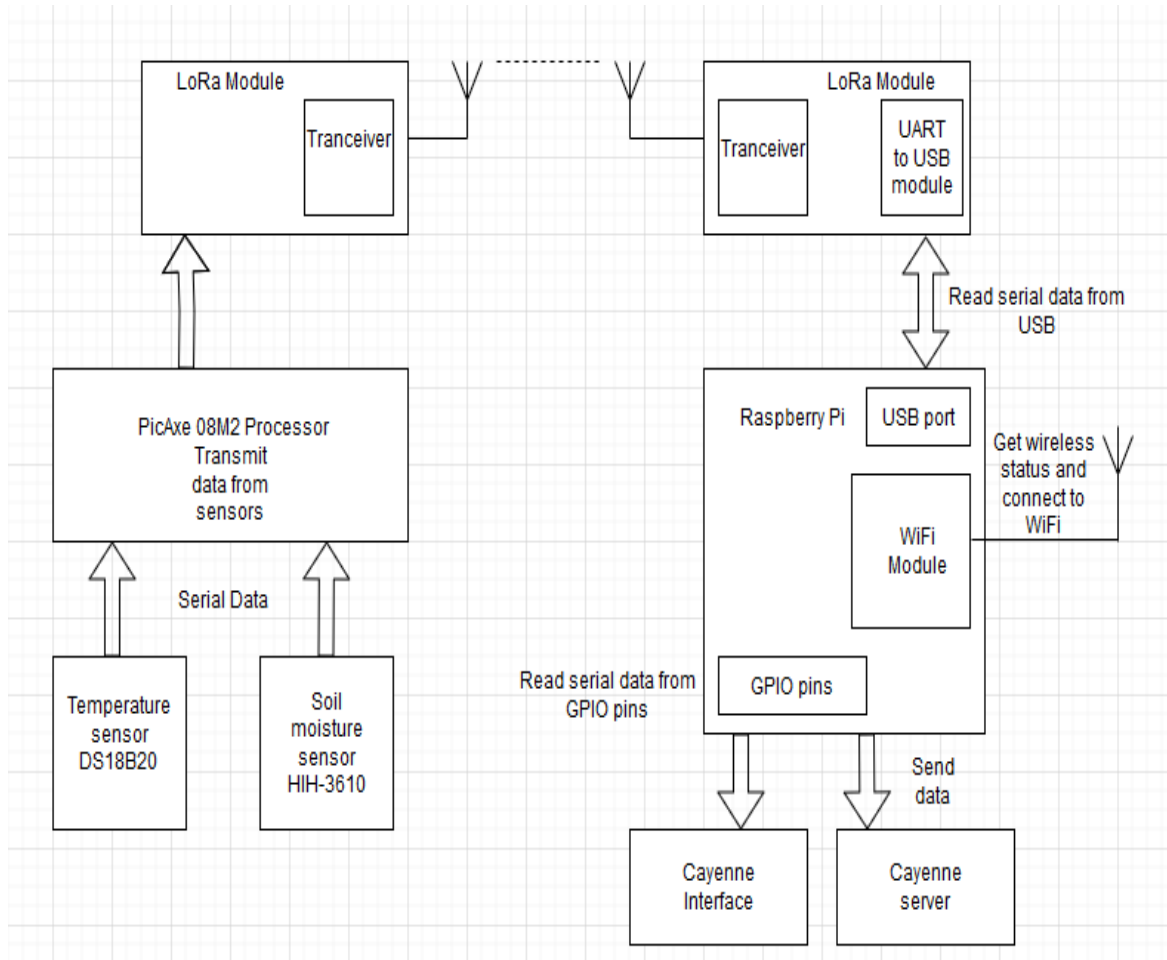


Fig 23 : Block Diagram

8 CONCLUSION

The objective of this Project was to study Internet of Things and implement a practical IoT solution using Cayenne cloud platform. The background theory about IoT had been covered in this Project.

There had been a need to come up with a widely accepted and used IoT architecture model. But no single standard had be brought into practice. Instead there are different IoT architecture models in use today. The absence of a uniform standard among the IoT industry poses a challenge in terms of interoperability of different architecture.

The IoT implementation with the Cayenne platform was used to demonstrate the practical application of IoT. This showed that connecting devices, transferring data and running different applications is simpler when performed using the Cayenne platform. The platform also offers scalable solution that can be applied in a system with hundreds or thousands of devices. Since all data is encrypted during transmission, it would be more difficult for an attacker to get access to the data in case of a cyber-attack.

Although the internet of things is expanding rapidly and more devices are being connected to the internet, there are some challenges facing the IoT industry. Some of these challenges are security, interoperability, weak internet infrastructure and lack of governmental regulation. Currently different actions are being undertaken by the different stake- holders involved to address these challenges and there had been advances made.

9 REFERENCES

Ranger, S. (2020, February 3). What is the IoT? Everything you need to know about the Internet of Things right now. Retrieved March 28, 2020, from <https://www.zdnet.com/article/what-is-the-internet-of-things-everything-you-need-to-know-about-the-iot-right-now/>

Priya. (2019, July 12). IOT Building Blocks and Architecture: IOT Part 2. Retrieved April 2, 2020, from <https://www.engineersgarage.com/tutorials/iot-building-blocks-and-architecture-iot-part-2/>

Stokes, P. (2019, April 12). 4 Stages of IoT architecture explained in simple words. Retrieved April 5, 2020, from <https://medium.com/datadriveninvestor/4-stages-of-iot-architecture-explained-in-simple-words-b2ea8b4f777f>

IoT Standards & Protocols Guide: 2019 Comparisons on Network, Wireless Comms, Security, Industrial. (2020, January 2). Retrieved April 25, 2020, from <https://www.postscapes.com/internet-of-things-protocols/>

IoT Standards & Protocols Guide: 2019 Comparisons on Network, Wireless Comms, Security, Industrial. (2020, January 2). Retrieved April 25, 2020, from <https://www.postscapes.com/internet-of-things-protocols/>

DS18B20 Temperature Sensor Pinout, Specifications, Equivalents & Datasheet. (n.d.). Retrieved May 10, 2020, from <https://components101.com/sensors/ds18b20-temperature-sensor>