International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 07 Issue: 04 | April - 2023 Impact Factor: 8.176 ISSN: 2582-3930

PUBLISHING MULTIPLE SENSOR READINGS TO NODE-RED USING MQTT PROTOCOL

Soham Das
Department of Networking and Communication
SRM Institute of Science and Technology
sd8159@srmist.edu.in

Dr. K. Venkatesh
Department of Networking and Communication
SRM Institute of Science and Technology
venkatek2@srmist.edu.in

Abstract **- In this project, we aim to make a deep dive into the concepts of data collection from multiple sensors using the MQTT protocol. We would be using Node-red to publish the sensor readings with the help of ESP8266 MQTT and Arduino. ESP8266 would be taken as a publisher whereas Node-red will be taken as a subscriber. We would be using the humidity, temperature, and pressure sensors reading and publishing it to the dashboard created on Node-Red. Mosquitto broker will help in creating the publisher and subscriber connection.**

For the project, we would use 3 kinds of sensors, the DHT22 sensor, the BME280 sensor, and the BME280 sensor. These 3 sensors are capable of detecting both temperature and humidity. The BME280 additionally also has the feature of sensing pressure We decided to use the MQTT protocol because it is energy efficient and easy to deploy in different machines and is a very influential protocol if we see it in respect of Industry 4.0. MQTT in IoT uses quantity of Service(QoS) levels to ensure proper message delivery to the receivers even when there is an unreliable connection between two devices.

Node-Red is a robust graphical programming platform that is used for programming tools and types of equipment for wiring hardware devices, API, and online services. It provides a browser-based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single click.

The documentation of the project will aim to study past studies and research and also explore the scope of the MQTT Protocol from the Industry 4.0 perspective. In industry, there are several kinds of machinery for which the temperatures are needed to be controlled and substances used where humidity control is necessary. The sensors that can be attached to our machinery can detect the temperature and humidity associated with it and provide valuable insight to the respective authorized personnel working with it. This will help in controlling system overheating.

With the help project, I propose a solution for monitoring the humidity, temperature, and pressure with help of multiple attached sensors and connecting the sensors with a dashboard build by us using Node-Red. The dashboard will receive the data from the sensor through MQTT protocol which is a light weight protocol which will be useful in low bandwidth areas also.

I. INTRODUCTION

A collection of communication standards known as M2M (Machine-to-Machine) protocols allows devices to communicate data with one another without the need for human interaction. These protocols let devices communicate and work together in different types of firms which include manufacturing, transportation, healthcare, and smart homes. A number of M2M protocols are available, each created for particular use cases and applications. Here are a few typical M2M protocols:

1. (MQTT) is a compact pub/sub messaging standard created for IoT and other unpredictable, low-bandwidth networks. It is extensively utilized in sectors including healthcare, industrial automation, and smart homes.
2. CoAP: The lightweight Constrained Application Protocol (CoAP) was created for networks and devices with limited resources, such as those used in smart homes, healthcare, and industrial automation. It may be used with a variety of transport protocols, including UDP and SMS, and is used for device-to-device communication.
3. LwM2M: Lightweight Machine-to-Machine (LwM2M) is a protocol created for controlling and keeping track of Internet of Things (IoT) devices remotely. It offers a common data model for device administration and security and makes use of the CoAP protocol.
4. Real-time data distribution and communications between devices are made possible through the Data Distribution Service (DDS) protocol. Real-time data is crucial in sectors like aerospace, defense, and healthcare, where it is often employed. The communication between devices is standardised thanks to these protocols, making it easier to integrate and interoperate various systems and devices. Moreover, they provide attributes like security, dependability, and scalability, which are crucial for M2M communication across several sectors.

A messaging protocol called MQTT is created specifically for IoT and other networks with limited resources. It was created in the late 1990s by Arlen Nipper of Arcom (now Eurotech) and Andy Stanford-Clark of IBM, and it has since grown to be a widely used standard for IoT communications.

The simplicity of MQTT is its distinguishing quality. MQTT is made to work well on low-power, resource-constrained devices like sensors, actuators, and other Internet of Things (IoT)

devices, in contrast to other messaging protocols that might be resource-intensive and need a lot of bandwidth. This is accomplished through the use of a publish-subscribe communication architecture, in which a device may send a message to a broker, who then distributes it to any subscribing devices. This method does away with the requirement for ongoing polling or upholding a permanent connection, which can save a substantial amount of power and bandwidth.

MQTT also supports three levels of quality of service (QoS) for message delivery, which allows for fine-grained control over the reliability and consistency of message transmission. The three levels of QoS are:

1. QoS 0: Just one delivery at most. There is no assurance that the subscriber will get the message, which is transmitted just once. Although there is the least dependability at this level, there is also the least overhead.
2. QoS 1: Delivery at least once. At least one attempt is made to send the message, and the broker will keep trying until the subscriber responds by acknowledging receipt. This level has a little larger overhead than QoS 0, but it offers a higher level of dependability.
3. Precisely once delivery under QoS 2. The broker will make sure the subscriber receives the message precisely once after it is transmitted exactly once. Although it has the biggest overhead, this level offers the best level of dependability.

The support for security provided by MQTT is another crucial element. To guarantee that messages are transmitted safely between devices and brokers, MQTT may be protected using Transport Layer Security (TLS) encryption and authentication. This is particularly crucial for Internet of Things (IoT) applications, where security is essential to thwart assaults and data breaches. Overall, MQTT is a popular option for IoT applications due to its lightweight design, support for various QoS levels, and security features. It is widely utilized for a range of purposes, including remote monitoring, control, and data collecting, in industries including smart homes, industrial automation, healthcare, and agriculture.

With the help of ESP32 MQTT and the Arduino IDE, we will be posting sensor values to Node-Red. By posting and subscribing to MQTT topics, we will carry out MQTT communication between the ESP32 and Node-Red. One ESP32 MQTT publisher and one Node-Red subscriber will be present. With the help of the ESP32, we will use the DS18B20, DHT, and BME280 sensors to publish sensor data to MQTT. The Dashboard will be sued to subscribe to the MQTT topics and display the sensor values on Node-Red.

Industry 4.0 is fundamentally based on the MQTT protocol. Industry 4.0, is the integration of cutting-edge digital technology into industrial processes in order to build "smart factories" and increase production productivity, efficiency, and flexibility. The German government first used the phrase "Industry 4.0" in 2011 as part of a high-tech initiative to encourage the use of digital technology in manufacturing. The preceding three industrial revolutions, which introduced automation, electricity, and steam power to production, served as the foundation for Industry 4.0. IoT, AI, ML, 3D printing, AR and VR capabilities are just a few examples of the digital and physical systems that make up Industry 4.0. These technologies are utilized to build autonomous, intelligent, networked, and systems that can continuously monitor and improve manufacturing processes.

Improved product quality, better productivity and adaptability, less downtime, and cost savings are all advantages of Industry 4.0. Moreover, it enables businesses to offer novel goods and services that are tailored and made to suit the requirements of certain clients. The workforce must be retrained, new business models and strategies must be developed, and there must be a large investment in new technology. Industry 4.0, as a whole, points to a dramatic evolution in production is conducted, with the scope to completely change the BPS, spur economic expansion, and increase competitiveness.

II. SCOPE

The MQTT protocol has a wide range of applications in a wide range of sectors, including:

1. Internet of Things: Because of its portability, low bandwidth, and low-power requirements, MQTT is frequently employed in the IoT space. In smart homes, industrial automation, and healthcare, it is frequently used for machine-to-machine communication, sensor data transfer, and device administration.

2. Mobile apps: For real-time communications and notifications, MQTT is also utilized in mobile applications. It is perfect for usage in messaging applications, chatbots, and push notifications since it enables effective and dependable communication between mobile devices and back-end systems.

3. Industrial automation: To facilitate communication between machines, sensors, and other devices in a manufacturing environment, MQTT is utilized in industrial automation. It offers a dependable and effective method for data sharing between machines and the optimization of manufacturing procedures.

4. Transportation: To facilitate communication between vehicles and traffic management systems, MQTT is utilized in transportation systems like linked automobiles and smart traffic systems. It enables real-time data analysis and transfer, enhancing efficiency, safety, and traffic flow.

III. CHALLENGES AND LIMITATIONS

Although the MQTT (Message Queuing Telemetry Transport) protocol has numerous advantages and is extensively utilized in a variety of fields and applications, there are several difficulties and restrictions that should be taken into account:

1. Security: As MQTT does not by default offer end-to-end encryption, communications may be intercepted and read by unauthorized parties. Further security measures like Transport Layer Security (TLS) and authentication should be used to ensure a safe connection.

2. Compatibility: Older hardware or gadgets that don't support MQTT may not work with it. The complexity and expense of implementation may rise in some circumstances if extra hardware or software is needed to support MQTT communication.

3. Restricted features: Although MQTT offers certain fundamental messaging functions, it could be missing some more sophisticated functionalities that are present in other messaging protocols. For instance, file transmission is not natively supported by MQTT, even though it may be necessary in specific use cases.

4. Implementation complexity: The implementation of MQTT's extra components, such as a broker and client libraries, might make the system more difficult overall.

Apart from technological issues, the Windows operating system faces the following difficulties:

1. Ports need to be manually opened because they are not opened automatically.

2. To record the readings, the windows defender firewall may need to be disabled.

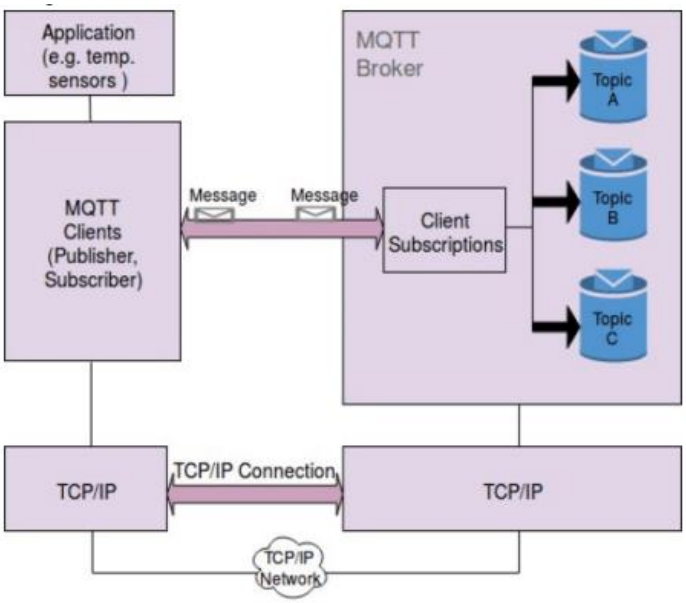
IV. LITERATURE REVIEW

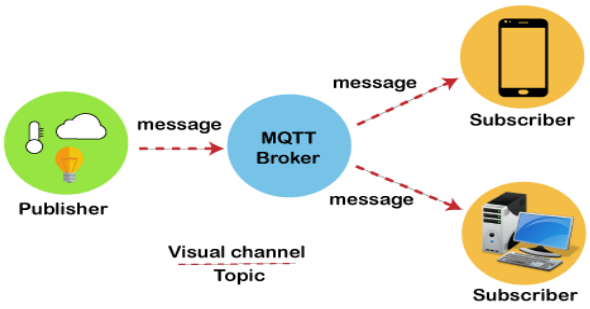
A promising technology has arisen called the Internet of Things (IoT). Data transport has been reduced as a result of the IoT objects' constrained resources. In order to comply with these demands and limitations, new protocols have been established. Many other IoT application protocols have been proposed, MQTT, CoAP, and many more [1].

MQTT is classified as being open-source, having a small amount of network bandwidth requirements, and being suitable for remote locations. Mobile devices, such as tablets and smartphones, are compatible with MQTT apps. Telemetry is also used to manage statistics that are collected from remote sensors. The fact that MQTT is a flexible and lightweight protocol is what led people and builders to choose it. The publish/subscriber system is made to be simple to use in practice. These qualities make it ideal for data and analytics as well as trading between devices [3].

Applications for this developing technology include data monitoring and home automation. With the help of home automation, we are achieving comfort in our daily lives. Homes with practically everything hooked up to a remotely scalable framework, including outlets, heating and cooling systems, and appliances, are more accurately portrayed by homes with home automation. A significant number of IoT devices must communicate with one another for home automation. Firmware needs to be updated frequently due to the noticeable increase in the number of devices on cloud platforms. It involves removing previously installed devices, making the necessary code modifications, and then flashing the modified code once more. Information processing should be done somewhere else if possible in order to solve these problems. Node-RED, a visual wiring tool, facilitates quick and simple connection configurations by enabling the association of devices. ESP8266 and an MQTT broker are connected to a range of devices using Node-RED, a monitoring and control dashboard will be developed and settled

According to [3], the MQTT orientation will be classified into 2 parts. This part of the module gives an explanation of each component





IoT has evolved into the term "smart things," which are objects with internal computation and system linkages. They are especially capable of sensing our surroundings and acting shrewdly as a result. Node-RED is used to make relevant connections between two or more hardware components, web services, and APIs. Node-RED was developed by IBM Emerging Technologies and is open source It functions essentially as a visual graphical programming tool created for the IoT, but it may also be used in other applications for rapid collection of flows of different services. It is built on Node.js, a framework for server-side java scripting. By substituting common coding activities, With the help of Node-RED, people would directly integrate equipment and Online services, and this should be accomplished through the use of a graphical stretch layout.

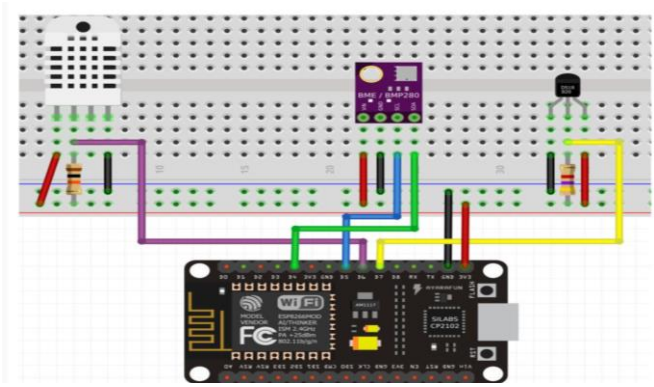
By linking several Node-RED components, a stream is generated in the Node-RED. The Node-RED flows are typically preserved in JSON, which functions as a demonstration of how easy it is to transmit the information throughout devices for collaboration with others. Nodes are interconnected to one another through flow keeping to achieve a predefined purpose. similar to how flows may be combined appropriately to achieve higher-order outcomes. Node-RED can be utilized locally by establishing a browser at http://localhost:1880. A cluster of nodes contained within a graphical user module for Node-RED can be employed to quickly construct a live data showcase. Using their local browser, users may view the user interface via http://localhost:1880/ui.

V. METHODOLOGY

Ensure that Thonny IDE and Micro-Python are both up-to-date and installed on your computer. The firmware ought to have already been installed onto your ESP8266 board.

The MQTT broker will be contacted by an ESP8266 board outfitted with DHT22, BME280, and DS18B20 sensors. We'll employ a Mosquito broker. This ESP8266 board transmits the DHT22 humidity and temperature measurements on the MQTT topic esp8266/dht/temperature and esp8266/dht/humidity, respectively. The BME280 temperature values are also published under the topic: esp8266/bme280/temperature. In the page esp8266/bme280/humidity, the BME280 humidity values are published. The BME280 pressure values are also published under the topic: esp8266/bme280/pressure. Similar to other message boards, this one posts DS18B20 temperature values in both Celsius and Fahrenheit on the MQTT topic esp8266/ds18b20/temperatureC and esp8266/ds18b20/temperatureF. With regard to these seven topics, Node-Red is a subscriber. As soon as Node-Red gets sensor data, it presents it interactively on its dashboard.

VI. CONNECTIONS



The power supply (VCC) pin is the first pin on the DHT22 sensor. It should be connected to the ESP8266's 3.3-volt or Vin pin. The pin designated as "Data Out" allows us to obtain samples of temperature and humidity from the DHT sensor. Attach the data pin to the 10k pull-up resistor and this pin to the ESP8266's GPIO12 at the same time. You may also use any digital pin on the ESP8266, though. For proper communication between the microcontroller and sensor, a pull-up resistor is employed to maintain the data pin at a high voltage. To learn more about the DHT22, you may look at its datasheet. DHT22 is sometimes referred to as AM2302. The resistor is not required if you are using a DHT22 sensor module. The BME280 is fairly simple to connect to the ESP8266 boards. The VCC terminal must be connected to 3.3V, the sensor's SCL to the module's SCL pin, the grounding hub to commonalities, and the sensor's SDA to the ESP8266 component's SDA pin. The ESP8266's GPIO4 and GPIO5 are SDA and SCL's conventional I2C pins, respectively.

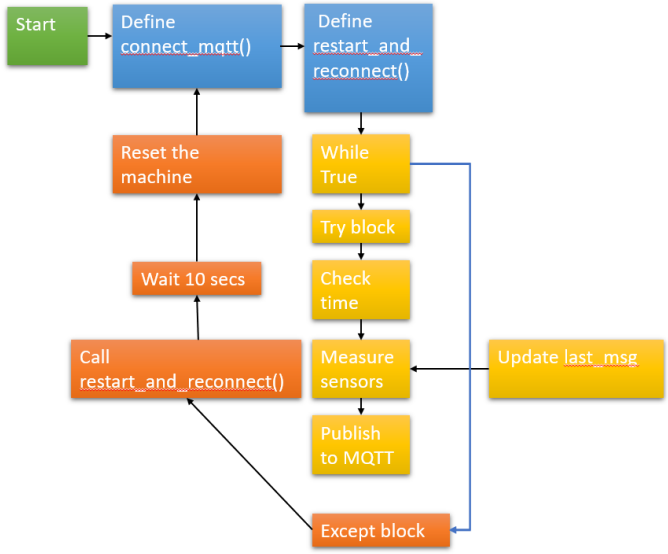
There are two ways to power the DS18B20 sensor. Normal Mode: The VDD pin and 4.7K ohm pull-up resistor are used to power the sensor from an external source. The sensor is powered by its own data line in parasite mode. As a result, no external power source is needed. Although DS10B20 will be operated in normal mode, the ESP8266 board's 3.3V pin serves as its Vcc pin's power source. There are three connections on the DS18B20 sensor. The ESP8266 board grounds the first terminal. A 4.7k-ohm pull-up resistor is used to link the center terminal of the sensor's data line to GPIO13. Every other GPIO

pin is also an option. The ESP board's 3.3V supplies power to the third terminal.

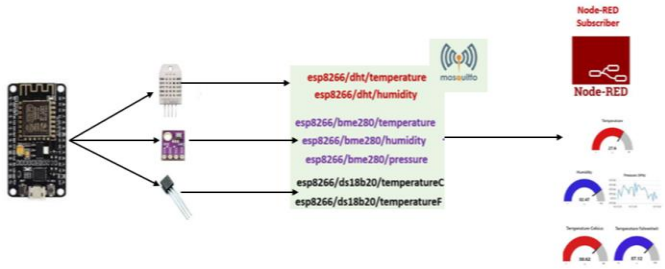
Activate the Thonny Shell. The MQTT broker and Wi-Fi will first be established connections on the ESP8266 board. It will post the sensor readings on the relevant subjects every 10 seconds.

VI. ALGORITHM

The flowchart of how the main micropython file is working:



The full project structure:



The ESP8266 board will be connected to the three types of sensors and data from the sensor will be published to the topics through the MQTT Broker. The Node red dashboard will subscribe to each of the topics to capture the published data and would be displaying it subsequently as it changes.

VII. LIBRARIES AND DEPENDENCIES

The 'umqttsimple' library for Micro Python is the first package that must be installed since we must utilize the MQTT protocol. It is a lightweight Micro python MQTT client. To proceed with our project, we will need to install a BME280 library for Micro Python. Micropython-bme280 will be used in our project. With the DS18x20 digital thermometer, for example, the OneWire library offers routines for communication via the Dallas OneWire protocol. OneWire is a Master/Slave protocol where just one wire is needed for communication. Then we need to install node-red. Before installation of node-red, we need to make sure there is node.JS in our machine. A programming instrument called Node-RED has been utilized to produce fresh and compelling connections between physical objects, APIs, and networking sites. It delivers a browser-based editor that makes it simple to wire up processes utilizing a diverse range of nodes in the range of shades that are able to be deployed to its runtime with a click of the mouse. Inside node–red a dashboard will be created for which we need to install another dependency. The Node-RED Dashboard is a group of nodes that lets you build a web dashboard that communicates with your flow directly. Inject nodes may be swapped out for buttons utilizing the different nodes, and data can now be sent straight to a web component like a gauge or table without having to first print it to the debug.

VIII. CONCLUSION

The MicroPython ESP8266 MQTT Post Multiple Sensor Readings to Node-Red project uses an ESP8266 module powered by MicroPython to receive sensor data and broadcast it using the MQTT protocol to a Node-Red server. The project offers a straightforward method for gathering and instantly visualising sensor data from far-off sites.

In conclusion, this project shows how simple it is to gather and share sensor data using MicroPython. It also emphasises how effective the MQTT protocol is in enabling device connection over the internet. On the other hand, Node-Red offers a straightforward and user-friendly interface for developing dashboards and real-time representations of sensor data.

Overall, the project provides a workable method for remotely monitoring sensor data, making it applicable to many different fields, such as home automation, agricultural, and industrial monitoring.

REFERENCES

[1] "A QOS approach for Internet of Things(IOT) environment using MQTT protocol" by Abdulrahman A. Sadeq, Rosilah Hassan, Salah S. Al-rawi, Ahmed M. Jubair and Azana H.M. Aman

[2] "IOT based Home Automation using Node-RED" by Ravi Kishore Kodali and Arshiya Anjum (2018).

[3] "IoT Monitoring System Based on MQTT Publisher/Subscriber Protocol" by Narges A-hussein and Ayman D. Salman (2020).