Subject: Embedded System and RTOS	
Name of student:	
Class: BE (E&TC)	**Roll no.:**
Semester/Year: VII (2022-23)	**Exam no.:**
Date of performance:	**Date of submission:**
Examined by:	

Experiment No :

Introduction to the MQTT and sending sensor data to the cloud using Raspberry-Pi/ Beagle Board/ Arduino.

AIM: Introduction to the MQTT and sending sensor data to the cloud using Raspberry-Pi/ Beagle Board/ Arduino.

OBJECTIVES:

1. To understand the working of the MQTT sensor.

APPARATUS:

1. Arduino Uno Wifi Rev2
2. Computer/Laptop
3. Arduino IDE

THEORY:

MQTT (originally an initialism of **MQ Telemetry Transport**) is a lightweight, publish-subscribe, machine to machine network protocol for Message queue/Message queuing service. It is designed for connections with remote locations that have devices with resource constraints or limited network bandwidth. It must run over a transport protocol that provides ordered, lossless, bi-directional connections—typically, TCP/IP. It is an open OASIS standard and an ISO recommendation (ISO/IEC 20922).

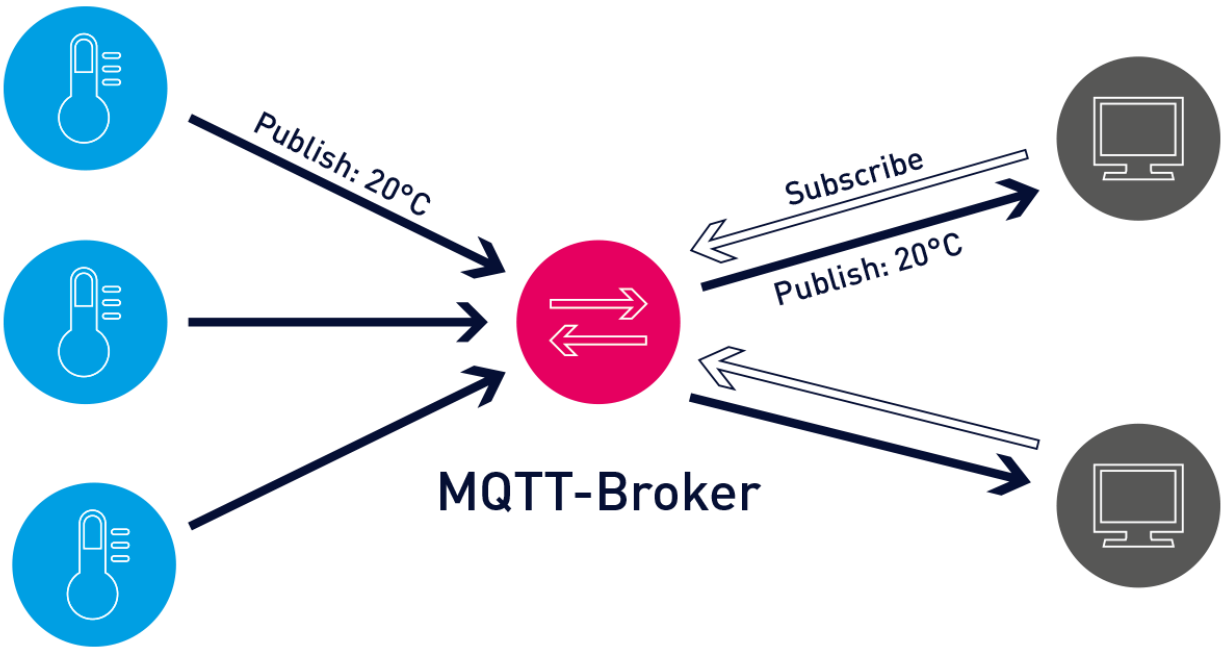
The MQTT protocol defines two types of network entities: a message broker and a number of clients. An MQTT broker is a server that receives all messages from the clients and then routes the messages to the appropriate destination clients. An MQTT client is any device (from a micro controller up to a fully-fledged server) that runs an MQTT library and connects to an MQTT broker over a network.

Information is organized in a hierarchy of topics. When a publisher has a new item of data to distribute, it sends a control message with the data to the connected broker. The broker then distributes the information to any clients that have subscribed to that topic. The publisher does not need to have any data on the number or locations of subscribers, and subscribers, in turn, do not have to be configured with any data about the publishers.

If a broker receives a message on a topic for which there are no current subscribers, the broker discards the message unless the publisher of the message designated the message as a retained message. A retained message is a normal MQTT message with the retained flag set to

true. The broker stores the last retained message and the corresponding QoS for the selected topic. Each client that subscribes to a topic pattern that matches the topic of the retained message receives the retained message immediately after they subscribe. The broker stores only one retained message per topic. This allows new subscribers to a topic to receive the most current value rather than waiting for the next update from a publisher. When a publishing client first connects to the broker, it can set up a default message to be sent to subscribers if the broker detects that the publishing client has unexpectedly disconnected from the broker. Clients only interact with a broker, but a system may contain several broker servers that exchange data based on their current subscribers' topics.A minimal MQTT control message can be as little as two bytes of data. A control message can carry nearly 256 megabytes of data if needed. There are fourteen defined message types used to connect and disconnect a client from a broker, to publish data, to acknowledge receipt of data, and to supervise the connection between client and server.

MQTT relies on the TCP protocol for data transmission. A variant, MQTT-SN, is used over other transports such as UDP or Bluetooth. MQTT sends connection credentials in plain text format and does not include any measures for security or authentication. This can be provided by using TLS to encrypt and protect the transferred information against interception, modification or forgery. The default unencrypted MQTT port is 1883. The encrypted port is 8883.

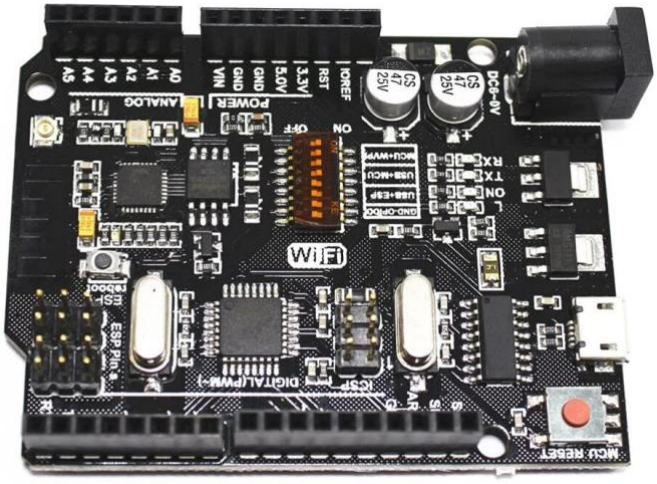


Arduino:

The Arduino Uno WiFi is an Arduino Uno with an integrated WiFi module. The board is based on the ATmega328P with an ESP8266WiFi Module integrated. The ESP8266WiFi Module is a self contained SoC with integrated TCP/IP protocol stack that can give access to

your WiFi network (or the device can act as an access point). One useful feature of Uno WiFi is support for OTA (over-the-air) programming, either for transfer of Arduino sketches or WiFi firmware.

The Arduino Uno WiFi is programmed using the Arduino Software (IDE), our Integrated Development Environment common to all our boards and running both online and offline.



CONCLUSION:

Code:

Header File:

```
#define SECRET_SSID "MESCOE"
#define SECRET_PASS "ENTC"
```

Sender Code:

```
#include <ArduinoMqttClient.h>
#include <WiFiNINA.h>
#include "arduino_secrets.h"

///////please enter your sensitive data in the Secret tab/arduino_secrets.h
char ssid[] = SECRET_SSID;        // your network SSID (name)
char pass[] = SECRET_PASS;    // your network password (use for WPA, or use as key for WEP)

WiFiClient wifiClient;
MqttClient mqttClient(wifiClient);

const char broker[] = "test.mosquitto.org";
int        port     = 1883;
const char topic[]  = "real_unique_topic";
const char topic2[]  = "real_unique_topic_2";
const char topic3[]  = "real_unique_topic_3";

//set interval for sending messages (milliseconds)
const long interval = 8000;
unsigned long previousMillis = 0;

int count = 0;

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
```

```
  }

  // attempt to connect to Wifi network:
  Serial.print("Attempting to connect to WPA SSID: ");
  Serial.println(ssid);
  while (WiFi.begin(ssid, pass) != WL_CONNECTED) {
    // failed, retry
    Serial.print(".");
    delay(5000);
  }

  Serial.println("You're connected to the network");
  Serial.println();

  Serial.print("Attempting to connect to the MQTT broker: ");
  Serial.println(broker);

  if (!mqttClient.connect(broker, port)) {
    Serial.print("MQTT connection failed! Error code = ");
    Serial.println(mqttClient.connectError());

    while (1);
  }

  Serial.println("You're connected to the MQTT broker!");
  Serial.println();
}

void loop() {
  // call poll() regularly to allow the library to send MQTT keep alive which
  // avoids being disconnected by the broker
  mqttClient.poll();

  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    // save the last time a message was sent
    previousMillis = currentMillis;

    //record random value from A0, A1 and A2
    int Rvalue = analogRead(A0);
    int Rvalue2 = analogRead(A1);
```

```
    int Rvalue3 = analogRead(A2);

    Serial.print("Sending message to topic: ");
    Serial.println(topic);
    Serial.println(Rvalue);

    Serial.print("Sending message to topic: ");
    Serial.println(topic2);
    Serial.println(Rvalue2);

    Serial.print("Sending message to topic: ");
    Serial.println(topic2);
    Serial.println(Rvalue3);

    // send message, the Print interface can be used to set the message contents
    mqttClient.beginMessage(topic);
    mqttClient.print(Rvalue);
    mqttClient.endMessage();

    mqttClient.beginMessage(topic2);
    mqttClient.print(Rvalue2);
    mqttClient.endMessage();

    mqttClient.beginMessage(topic3);
    mqttClient.print(Rvalue3);
    mqttClient.endMessage();

    Serial.println();
  }
}
```

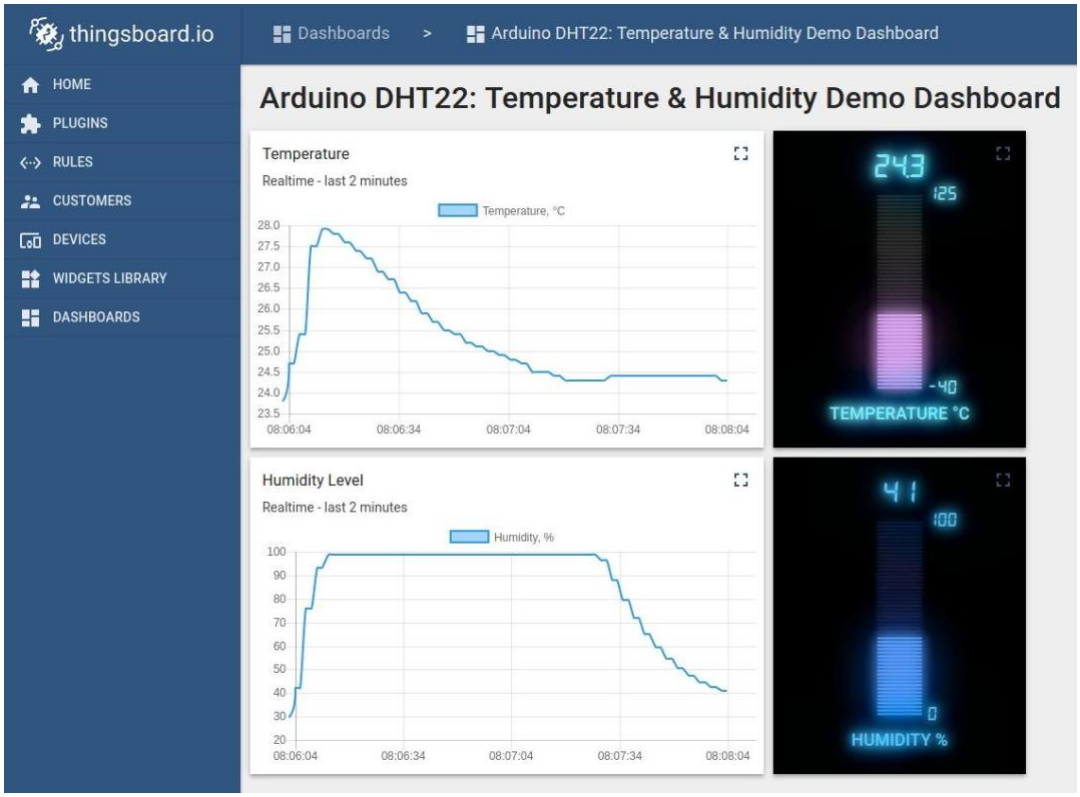
Receivers Code:

```
#include <ArduinoMqttClient.h>
#include <WiFiNINA.h>
#include "arduino_secrets.h"

///////please enter your sensitive data in the Secret tab/arduino_secrets.h
char ssid[] = SECRET_SSID;        // your network SSID
char pass[] = SECRET_PASS;    // your network password

WiFiClient wifiClient;
MqttClient mqttClient(wifiClient);
```

```
const char broker[] = "test.mosquitto.org";
int        port     = 1883;
const char topic[]  = "real_unique_topic";
const char topic2[]  = "real_unique_topic_2";
const char topic3[]  = "real_unique_topic_3";

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
  // attempt to connect to Wifi network:
  Serial.print("Attempting to connect to SSID: ");
  Serial.println(ssid);
  while (WiFi.begin(ssid, pass) != WL_CONNECTED) {
    // failed, retry
    Serial.print(".");
    delay(5000);
  }

  Serial.println("You're connected to the network");
  Serial.println();

  Serial.print("Attempting to connect to the MQTT broker: ");
  Serial.println(broker);

  if (!mqttClient.connect(broker, port)) {
    Serial.print("MQTT connection failed! Error code = ");
    Serial.println(mqttClient.connectError());

    while (1);
  }

  Serial.println("You're connected to the MQTT broker!");
  Serial.println();

  // set the message receive callback
  mqttClient.onMessage(onMqttMessage);

  Serial.print("Subscribing to topic: ");
```

```
  Serial.println(topic);
  Serial.println();

  // subscribe to a topic
  mqttClient.subscribe(topic);
  mqttClient.subscribe(topic2);
  mqttClient.subscribe(topic3);

  // topics can be unsubscribed using:
  // mqttClient.unsubscribe(topic);

  Serial.print("Topic: ");
  Serial.println(topic);
  Serial.print("Topic: ");
  Serial.println(topic2);
  Serial.print("Topic: ");
  Serial.println(topic3);

  Serial.println();
}

void loop() {
  // call poll() regularly to allow the library to receive MQTT messages and
  // send MQTT keep alive which avoids being disconnected by the broker
  mqttClient.poll();
}

void onMqttMessage(int messageSize) {
  // we received a message, print out the topic and contents
  Serial.println("Received a message with topic '");
  Serial.print(mqttClient.messageTopic());
  Serial.print("', length ");
  Serial.print(messageSize);
  Serial.println(" bytes:");

  // use the Stream interface to print the contents
  while (mqttClient.available()) {
    Serial.print((char)mqttClient.read());
  }
  Serial.println();
  Serial.println();
}
```

Output:


thingsboard.io
Dashboards
>
Arduino DHT22: Temperature & Humidity Demo Dashboard
HOME
PLUGINS
RULES
CUSTOMERS
DEVICES
WIDGETS LIBRARY
DASHBOARDS
Arduino DHT22: Temperature & Humidity Demo Dashboard
Temperature
Realtime - last 2 minutes
Temperature, °C
28.0
27.5
27.0
26.5
26.0
25.5
25.0
24.5
24.0
23.5
08:06:04
08:06:34
08:07:04
08:07:34
08:08:04
24.3
125
-40
TEMPERATURE °C
Humidity Level
Realtime - last 2 minutes
Humidity, %
100
90
80
70
60
50
40
30
20
08:06:04
08:06:34
08:07:04
08:07:34
08:08:04
41
100
0
HUMIDITY %