Ex. No: 9	Implementation of the basic MQTT Communication protocol.
04-10-2024	

Aim: To implement the MQTT communication protocol to enable lightweight, reliable messaging between devices in IoT applications, focusing on efficient data exchange over constrained networks.

1. Introduction to MQTT

- MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol designed specifically for low-bandwidth, high-latency, or unreliable networks, making it ideal for IoT applications. It follows a publish-subscribe model, unlike traditional request-response protocols such as HTTP.

- In the IoT ecosystem, MQTT is widely used for sending data from devices (publishers) to a server (broker), which then delivers this data to interested parties (subscribers). It is designed to minimize the amount of data exchanged between client and server and to ensure efficient use of network bandwidth.

2. Components of MQTT

There are three primary components in MQTT communication:

- Broker: A server that receives all messages from publishers and routes them to the correct subscribers. Examples of brokers include Mosquitto and HiveMQ.

- Publisher: A device or application that sends messages (data) to the broker. It doesn't directly communicate with subscribers; it only needs to know the broker's address.

- Subscriber: A device or application that listens for messages from the broker on specific topics. It only receives data that it has subscribed to.

- These components work together using the topic-based publish-subscribe paradigm. A "topic" is essentially an identifier for each message type.
- Publishers send data to a specific topic, and subscribers subscribe to those topics to receive relevant messages

3. Steps to Implement MQTT

To implement the MQTT protocol, follow these steps:

Step 1: Install Required Libraries

If you’re using Python for the implementation, the first step is to install the necessary libraries. The `paho-mqtt` library is a popular choice for implementing MQTT in Python. You can install it using pip:

```
pip install paho-mqtt
```

Step 2: Setting Up the MQTT Broker

The broker is the core of MQTT communication. One of the most commonly used brokers is Mosquitto. If you're using a local setup, you can install and run Mosquitto on your machine.

For Linux:

```
sudo apt-get install mosquitto

sudo systemctl start mosquitto
```

For a quick test setup, you can also use public brokers such as the one provided by Eclipse Mosquitto (`mqtt.eclipse.org`).

Step 3: Implementing the Publisher

The publisher sends data to a specified topic. Here's an example of a simple publisher in Python:

```
import paho.mqtt.client as mqtt

Define the MQTT broker
broker = "mqtt.eclipse.org"
port = 1883
topic = "iot/lab/mqtt_test"

Create a new MQTT client instance
client = mqtt.Client()

Connect to the broker
client.connect(broker, port)

Publish a message to the topic
client.publish(topic, "Hello from the MQTT Publisher!")

Disconnect after publishing
client.disconnect()
```

In this example, the publisher connects to the broker at `mqtt.eclipse.org`, sends a message "Hello from the MQTT Publisher!" to the topic `iot/lab/mqtt_test`, and then disconnects.

Step 4: Implementing the Subscriber

The subscriber listens for messages on a specific topic. Here's an example of a subscriber in Python:

```
import paho.mqtt.client as mqtt

Define the MQTT broker and topic
broker = "mqtt.eclipse.org"
port = 1883
topic = "iot/lab/mqtt_test"

Callback function when a message is received def
on_message(client, userdata, message):
  print(f"Message received: {message.payload.decode()}")

Create a new MQTT client instance
client = mqtt.Client()

Attach the callback function
client.on_message = on_message

Connect to the broker
client.connect(broker, port)

Subscribe to the topic
client.subscribe(topic)

Start the loop to process incoming messages
client.loop_forever()
```

In this subscriber implementation, the `on_message` callback function is triggered whenever a message is received on the `iot/lab/mqtt_test` topic. The `client.loop_forever()` function ensures that the subscriber continues to listen for incoming messages indefinitely.

Step 5: Testing the Implementation

Now that you have both a publisher and subscriber, you can test the entire system. Start the subscriber script, which will listen for messages, and then run the publisher script to send a message. You should see the message received on the subscriber end.

4. Quality of Service (QoS) in MQTT

MQTT offers different levels of Quality of Service (QoS), which determines the guarantee of message delivery:

- QoS 0: "At most once" delivery. The message is sent once, and no acknowledgment is required.

- QoS 1: "At least once" delivery. The message is sent and acknowledged. It may be delivered more than once if there's a failure in acknowledgment.

- QoS 2: "Exactly once" delivery. The message is ensured to be delivered only once. This is the highest level of QoS and involves a handshake mechanism between the sender and receiver.

You can specify the QoS level in both publisher and subscriber code:

client.publish(topic, "Message with QoS 1", qos=1)

client.subscribe(topic, qos=1)

5. Retained Messages and Last Will Testament (LWT)

- Retained Messages: MQTT allows sending a "retained" message, which means the broker will store the last message sent on a topic and send it to any new subscriber as soon as they subscribe.

- Last Will and Testament (LWT): This is a message set by the client that will be sent by the broker if the client disconnects unexpectedly. It is useful for notifying other devices in the network about a failure or disconnection.

Example:

```
client.will_set("iot/lab/status", "Publisher Disconnected", qos=1, retain=True)
```

6. Security in MQTT

While MQTT is designed to be lightweight, security is critical, especially in IoT applications. You can secure your MQTT communication by using SSL/TLS encryption, username-password authentication, and access control mechanisms.

Here's an example of enabling TLS in the MQTT client:

```
client.tls_set(ca_certs="path_to_ca_certificate.crt")
client.username_pw_set(username="user", password="password")
```

7. Conclusion

MQTT is an efficient protocol for IoT communication due to its lightweight nature and flexibility in delivering messages with different levels of QoS.

8. Result:

- In this implementation, we've covered the core components: setting up an MQTT broker, implementing a publisher and a subscriber, and exploring QoS and security aspects.
- This foundation can be extended to more complex IoT systems, such as connecting multiple devices or implementing bidirectional communication between devices and servers.
- By understanding these fundamentals, we can design robust and scalable IoT systems using MQTT, ensuring efficient and reliable communication between devices and applications.