

Department of Electronic and Telecommunication
Engineering
University of Moratuwa
Sri Lanka



EN3251 - Internet of Things

Laboratory Exercise 1 MQTT Implementation and Testing

Group 9

Name	Index Number
M. P. Wickramarathne	210703V
A.A.W.L.R.Amarasinghe	210031H
A.D.T. Dabare	210089P

This report is submitted as a partial fulfillment of module EN3251
2024.09.01

Contents

1	Connecting to the Broker	1
1.1	Publisher	1
1.2	Subscriber	2
2	QoS=0 instance	4
3	QoS=1 instance	6
4	QoS=2 instance	8
5	Codes	10
5.1	Publisher	10
5.2	Subscriber	11
6	Homework - Smart Plant Watering System	12
6.1	Controller Code	13
6.2	Moisture Sensor Code	15
6.3	Water Level Sensor Code	17
6.4	Water Pump Code	19
6.5	Outputs	21

1 Connecting to the Broker

1.1 Publisher

567	6.543045	192.168.142.4	91.121.93.94	MQTT	77 Connect Command
590	6.745466	91.121.93.94	192.168.142.4	MQTT	58 Connect Ack

Figure 1: Publisher Connect Wireshark

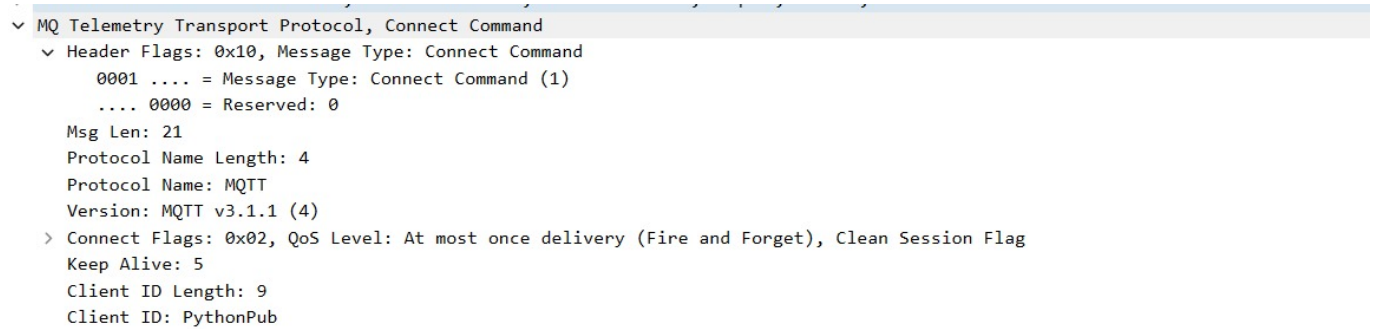


Figure 2: Publisher Connect Command

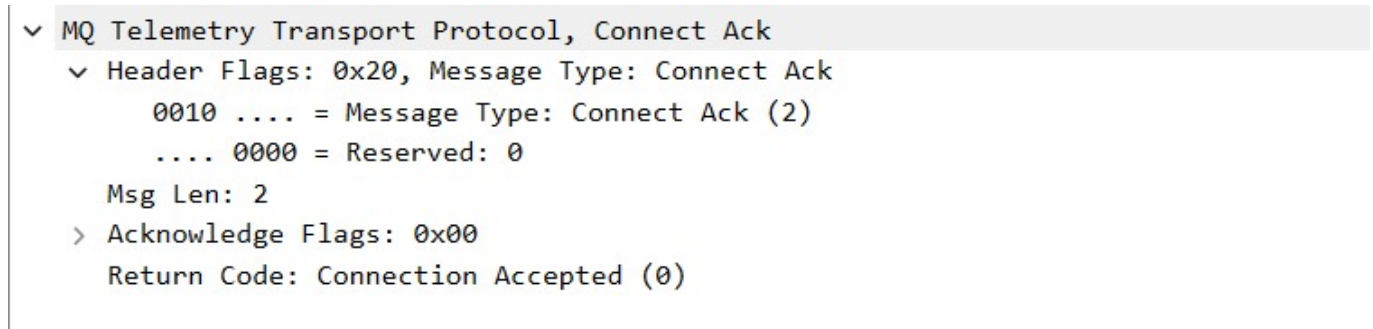


Figure 3: Publisher Connect ACK

- **Connect Command:** The first row shows the client (192.168.142.4) sending a *Connect Command* to the MQTT broker (91.121.93.94). This command initiates the connection request, where the client identifies itself and specifies any connection parameters (like keep-alive time, client ID, etc.).
- **Connect Ack:** The second row indicates that the broker (91.121.93.94) responds with a *Connect Ack (Acknowledgment)* to the client (192.168.142.4). This acknowledgment confirms that the connection has been successfully established.

1.2 Subscriber

No.	Time	Source	Destination	Protocol	Length	Info	Source Port
10	14:23:24.644327	192.168.208.102	91.121.93.94	MQTT	77	Connect Command	61067
13	14:23:24.915981	91.121.93.94	192.168.208.102	MQTT	58	Connect Ack	mqtt
14	14:23:24.917879	192.168.208.102	91.121.93.94	MQTT	68	Subscribe Request (id=1) [oranges]	61067
16	14:23:25.103795	91.121.93.94	192.168.208.102	MQTT	59	Subscribe Ack (id=1)	mqtt

Figure 4: Subscriber - Connect Wireshark

<p>Frame 10: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface \Device\NPF_{45C26253-EEA1-4751-B7AB-8E5E}</p> <p>Ethernet II, Src: IntelCor_f5:71:0c (e0:2b:e9:f5:71:0c), Dst: MS-NLB-PhysServer-32_07:42:a8:ba:1e (02:27:42:a8:ba:1e)</p> <p>Internet Protocol Version 4, Src: 192.168.208.102, Dst: 91.121.93.94</p> <p>Transmission Control Protocol, Src Port: 61067 (61067), Dst Port: mqtt (1883), Seq: 1, Ack: 1, Len: 23</p> <p>MQ Telemetry Transport Protocol, Connect Command</p> <p>Header Flags: 0x10, Message Type: Connect Command</p> <p>0001 = Message Type: Connect Command (1)</p> <p>.... 0000 = Reserved: 0</p> <p>Msg Len: 21</p> <p>Protocol Name Length: 4</p> <p>Protocol Name: MQTT</p> <p>Version: MQTT v3.1.1 (4)</p> <p>Connect Flags: 0x02, QoS Level: At most once delivery (Fire and Forget), Clean Session Flag</p> <p>Keep Alive: 5</p> <p>Client ID Length: 9</p> <p>Client ID: PythonSub</p>	<pre> 0000 02 27 42 a8 ba 1e e0 2b e9 f5 71 0c 08 00 45 b8 'B....+...q...E- 0010 00 3f 59 13 40 00 40 06 00 00 c0 a8 d0 66 5b 79 ?Y.@...f[y 0020 5d 5e ee 8b 07 5b bf 03 c2 3c c1 fa c9 a8 50 18]^...[...P- 0030 02 03 c1 6b 00 00 10 15 00 04 d4 51 54 54 04 02 ...k...MQTT... 0040 00 05 00 09 50 79 74 68 6f 6e 53 75 62 Pyth onSub </pre>
--	---

Figure 5: Subscriber - Connect Command

<p>Frame 13: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface \Device\NPF_{45C26253-EEA1-4751-B7AB-8E5E}</p> <p>Ethernet II, Src: MS-NLB-PhysServer-32_07:42:a8:ba:1e (02:27:42:a8:ba:1e), Dst: IntelCor_f5:71:0c (e0:2b:e9:f5:71:0c)</p> <p>Internet Protocol Version 4, Src: 91.121.93.94, Dst: 192.168.208.102</p> <p>Transmission Control Protocol, Src Port: 61067 (61067), Dst Port: 61067 (61067), Seq: 1, Ack: 24, Len: 4</p> <p>MQ Telemetry Transport Protocol, Connect Ack</p> <p>Header Flags: 0x20, Message Type: Connect Ack</p> <p>0010 = Message Type: Connect Ack (2)</p> <p>.... 0000 = Reserved: 0</p> <p>Msg Len: 2</p> <p>Acknowledge Flags: 0x00</p> <p>Return Code: Connection Accepted (0)</p>	<pre> 0000 e0 2b e9 f5 71 0c 02 27 42 a8 ba 1e 08 00 45 00 +-+q...B....E- 0010 00 2c 2c 19 40 00 33 06 d1 cc 5b 79 5d 5e c0 a8 .., @3...[y]^.. 0020 d0 66 07 5b ee 8b c1 fa c9 a8 bf 03 c2 54 50 18 .f[...TP- 0030 01 f6 41 07 00 00 20 02 00 00 -A... </pre>
---	---

Figure 6: Subscriber - Connect ACK

<p>Frame 14: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface \Device\NPF_{45C26253-EEA1-4751-B7AB-8E5E}</p> <p>Ethernet II, Src: IntelCor_f5:71:0c (e0:2b:e9:f5:71:0c), Dst: MS-NLB-PhysServer-32_07:42:a8:ba:1e (02:27:42:a8:ba:1e)</p> <p>Internet Protocol Version 4, Src: 192.168.208.102, Dst: 91.121.93.94</p> <p>Transmission Control Protocol, Src Port: 61067 (61067), Dst Port: mqtt (1883), Seq: 24, Ack: 5, Len: 14</p> <p>MQ Telemetry Transport Protocol, Subscribe Request</p> <p>Header Flags: 0x82, Message Type: Subscribe Request</p> <p>1000 = Message Type: Subscribe Request (8)</p> <p>.... 0010 = Reserved: 2</p> <p>Msg Len: 12</p> <p>Message Identifier: 1</p> <p>Topic Length: 7</p> <p>Topic: oranges</p> <p>Requested QoS: At least once delivery (Acknowledged deliver) (1)</p>	<pre> 0000 02 27 42 a8 ba 1e e0 2b e9 f5 71 0c 08 00 45 00 'B....+...q...E- 0010 00 36 59 14 40 00 40 06 00 00 c0 a8 d0 66 5b 79 ?Y.@...f[y 0020 5d 5e ee 8b 07 5b bf 03 c2 54 c1 fa c9 ac 50 18]^...[...T...P- 0030 02 03 4a 0f 00 00 82 0c 00 01 00 07 bf 72 61 6e ...k...bP- 0040 67 65 73 01 ges- </pre>
--	---

Figure 7: Subscriber - Subscribe Request

<p>Frame 16: 59 bytes on wire (472 bits), 59 bytes captured (472 bits) on interface \Device\NPF_{45C26253-EEA1-4751-B7AB-8E5E}</p> <p>Ethernet II, Src: MS-NLB-PhysServer-32_07:42:a8:ba:1e (02:27:42:a8:ba:1e), Dst: IntelCor_f5:71:0c (e0:2b:e9:f5:71:0c)</p> <p>Internet Protocol Version 4, Src: 91.121.93.94, Dst: 192.168.208.102</p> <p>Transmission Control Protocol, Src Port: mqtt (1883), Dst Port: 61067 (61067), Seq: 5, Ack: 38, Len: 5</p> <p>MQ Telemetry Transport Protocol, Subscribe Ack</p> <p>Header Flags: 0x90, Message Type: Subscribe Ack</p> <p>1001 = Message Type: Subscribe Ack (9)</p> <p>.... 0000 = Reserved: 0</p> <p>Msg Len: 3</p> <p>Message Identifier: 1</p> <p>Granted QoS: At least once delivery (Acknowledged deliver) (1)</p>	<pre> 0000 e0 2b e9 f5 71 0c 02 27 42 a8 ba 1e 08 00 45 00 +-+q...B....E- 0010 00 2d 2c 1a 40 00 33 06 d1 ca 5b 79 5d 5e c0 a8 .., @3...[y]^.. 0020 d0 66 07 5b ee 8b c1 fa c9 ac bf 03 c2 62 50 18 .f[...bP- 0030 01 f6 cf f1 00 00 90 03 00 01 01 </pre>
---	--

Figure 8: Subscriber - Subscribe ACK

The first packet (Packet 10) shows the MQTT client with the IP address 192.168.208.102 sending a *Connect Command* to the MQTT broker at 91.121.93.94. This command is the initial step in establishing a connection between the client and the broker. The *Connect Command* typically includes information such as the client's identifier, protocol level, and any optional flags for features like authentication or last will messages.

In the second packet (Packet 13), the MQTT broker responds to the client's connection request with a *Connect Ack* message. This acknowledgment signifies that the broker has accepted the client's

connection request, and the communication channel is now established. The *Connect Ack* may also include information such as the session present flag, which indicates whether the broker has a persistent session for the client.

The third packet (Packet 14) involves the client sending a *Subscribe Request* to the broker. In this case, the client is subscribing to the topic "oranges," which is identified by an ID of 1. This subscription request indicates that the client wishes to receive any messages that are published to the "oranges" topic. The broker will process this request and, if successful, start sending messages on this topic to the client.

Finally, the fourth packet (Packet 16) shows the MQTT broker sending a *Subscribe Ack* message back to the client. This acknowledgment confirms that the subscription request was successful and that the client is now subscribed to the "oranges" topic. From this point onward, the client will receive messages that are published to the "oranges" topic, completing the typical MQTT subscribe workflow.

2 QoS=0 instance

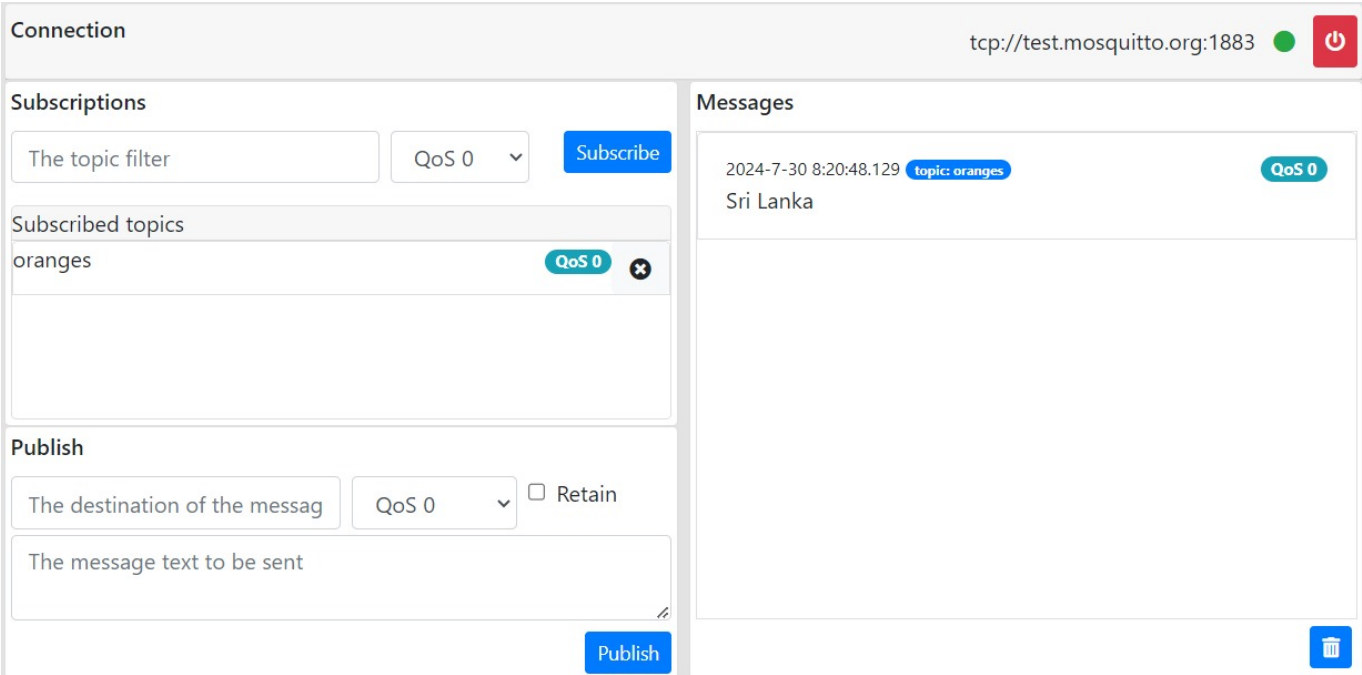


Figure 9: Test QoS 0

The client has subscribed to the topic "oranges" with a Quality of Service (QoS) level of 0, which is visible under the "Subscribed topics" section. In the "Messages" panel, the client has received a message with the content "Sri Lanka," which was published to the "oranges" topic. The QoS level for this message is also 0, indicating that the message was delivered without requiring acknowledgment. This setup demonstrates a basic MQTT interaction where the client successfully subscribes to a topic and receives a message published to that topic.

7	0.929201	192.168.142.4	91.121.93.94	MQTT	56 Ping Request
8	1.224446	91.121.93.94	192.168.142.4	MQTT	56 Ping Response
20	5.387192	192.168.142.4	91.121.93.94	MQTT	74 Publish Message [oranges]
24	6.392468	192.168.142.4	91.121.93.94	MQTT	56 Ping Request
28	6.665245	91.121.93.94	192.168.142.4	MQTT	56 Ping Response

Figure 10: Publisher QoS 0 - Wireshark

The screenshot shows a Wireshark packet capture of MQTT traffic. The packet list table is as follows:

No.	Time	Source	Destination	Protocol	Length	Info	Source Port
14	13:40:11.572604	192.168.208.102	91.121.93.94	MQTT	56	Ping Request	60738
16	13:40:11.848981	91.121.93.94	192.168.208.102	MQTT	56	Ping Response	mqtt
51	13:40:16.909761	192.168.208.102	91.121.93.94	MQTT	56	Ping Request	60738
52	13:40:17.063615	91.121.93.94	192.168.208.102	MQTT	56	Ping Response	mqtt
72	13:40:22.109451	192.168.208.102	91.121.93.94	MQTT	56	Ping Request	60738
74	13:40:22.345412	91.121.93.94	192.168.208.102	MQTT	56	Ping Response	mqtt
86	13:40:26.568438	91.121.93.94	192.168.208.102	MQTT	74	Publish Message [oranges]	mqtt
96	13:40:27.570227	192.168.208.102	91.121.93.94	MQTT	56	Ping Request	60738
98	13:40:27.848712	91.121.93.94	192.168.208.102	MQTT	56	Ping Response	mqtt

Figure 11: Subscriber QoS 0 - Wireshark

1. **Consistent Testing Environment:** Both captures display MQTT traffic, indicating that they are part of the same set of tests. The presence of "Ping Request" and "Ping Response" packets in both captures shows that the devices are maintaining a connection via the MQTT protocol, which is standard behavior in MQTT to keep the connection alive.

2. **Publish Message [oranges]:** In both captures, there is a "Publish Message [oranges]," suggesting that you were testing message publishing as part of the MQTT protocol. The repeated appearance of this specific message in both captures indicates that this was part of the testing script or sequence you used to verify the correct operation of MQTT message publishing.
3. **Different IP Pairs:** The first capture involves communication between 192.168.142.4 and 91.121.93.94, while the second involves 192.168.208.102 and 91.121.93.94. This suggests that the same MQTT broker or service (likely the one at 91.121.93.94) was tested with different clients (the different local IPs). This helps validate that the broker can handle multiple clients and that the MQTT protocol works consistently across different devices.
4. **Same MQTT Process:** The patterns in both captures are identical in terms of the MQTT protocol usage (pings and publishing messages), which demonstrates that both sets of packets are part of the same overall testing process. The tests were aimed at verifying whether the MQTT protocol handles connections and message publishing reliably across different devices or scenarios.

```
> Frame 14: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_{45C26253-EEA1-4751-B7AB-8E5E}
> Ethernet II, Src: IntelCor_f5:71:0c (e0:2b:e9:f5:71:0c), Dst: MS-NLB-PhysServer-32_07:42:a8:ba:1e (02:27:42:a8:ba:1e)
> Internet Protocol Version 4, Src: 192.168.208.102, Dst: 91.121.93.94
> Transmission Control Protocol, Src Port: 60738 (60738), Dst Port: mqtt (1883), Seq: 1, Ack: 1, Len: 2
MQ Telemetry Transport Protocol, Ping Request
  [Expert Info (Note/Protocol): Unknown version (missing the CONNECT packet?)]
  Header Flags: 0xc0, Message Type: Ping Request
    1100 .... = Message Type: Ping Request (12)
    .... 0000 = Reserved: 0
  Msg Len: 0
```

Figure 12: Subscriber QoS 0 - Ping Request

```
> Frame 16: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_{45C26253-EEA1-4751-B7AB-8E5E}
> Ethernet II, Src: MS-NLB-PhysServer-32_07:42:a8:ba:1e (02:27:42:a8:ba:1e), Dst: IntelCor_f5:71:0c (e0:2b:e9:f5:71:0c)
> Internet Protocol Version 4, Src: 91.121.93.94, Dst: 192.168.208.102
> Transmission Control Protocol, Src Port: mqtt (1883), Dst Port: 60738 (60738), Seq: 1, Ack: 3, Len: 2
MQ Telemetry Transport Protocol, Ping Response
  [Expert Info (Note/Protocol): Unknown version (missing the CONNECT packet?)]
  Header Flags: 0xd0, Message Type: Ping Response
    1101 .... = Message Type: Ping Response (13)
    .... 0000 = Reserved: 0
  Msg Len: 0
```

Figure 13: Subscriber QoS 0 - Ping Response

```
> Frame 86: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{45C26253-EEA1-4751-B7AB-8E5E}
> Ethernet II, Src: MS-NLB-PhysServer-32_07:42:a8:ba:1e (02:27:42:a8:ba:1e), Dst: IntelCor_f5:71:0c (e0:2b:e9:f5:71:0c)
> Internet Protocol Version 4, Src: 91.121.93.94, Dst: 192.168.208.102
> Transmission Control Protocol, Src Port: mqtt (1883), Dst Port: 60738 (60738), Seq: 7, Ack: 7, Len: 20
MQ Telemetry Transport Protocol, Publish Message
  [Expert Info (Note/Protocol): Unknown version (missing the CONNECT packet?)]
  Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
    0011 .... = Message Type: Publish Message (3)
    .... 0... = DUP Flag: Not set
    .... 00. = QoS Level: At most once delivery (Fire and Forget) (0)
    .... ...0 = Retain: Not set
  Msg Len: 18
  Topic Length: 7
  Topic: oranges
  Message: 537269204c616e6b61
```

Figure 14: Subscriber QoS 0 - Publish Message

3 QoS=1 instance

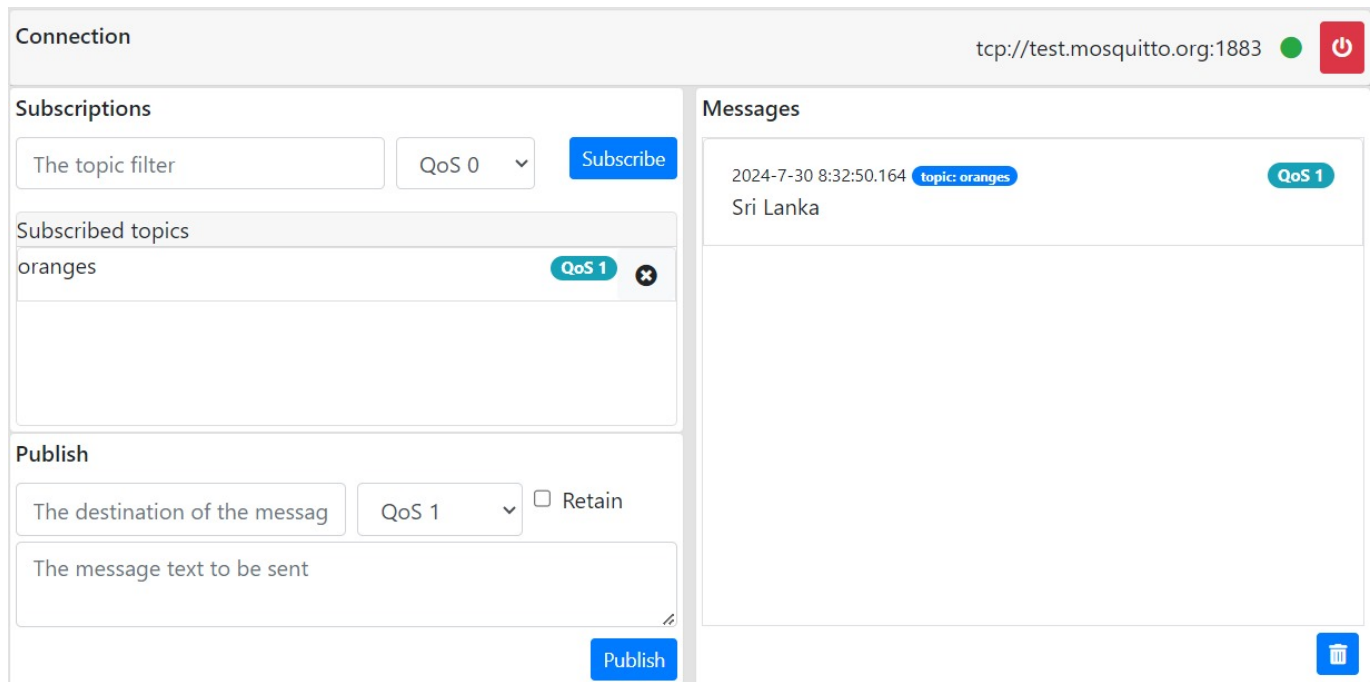


Figure 15: Test QoS 1

This demonstrates the test setup for MQTT Quality of Service (QoS) level 1 using the Mosquitto broker. The interface shows a subscription to the topic "oranges" with QoS 1, ensuring that messages are delivered at least once. A message titled "Sri Lanka" is published to the topic "oranges" and successfully received with QoS 1, as indicated in the Messages panel. The test confirms that the MQTT broker correctly handles message delivery and acknowledgment under QoS 1, where the message is delivered reliably with confirmation of receipt.

6	3.043824	91.121.93.94	192.168.142.4	MQTT	56 Ping Response
30	7.334033	192.168.142.4	91.121.93.94	MQTT	73 Publish Message (id=4) [oranges]
54	7.650797	91.121.93.94	192.168.142.4	MQTT	58 Publish Ack (id=4)

Figure 16: Publisher QoS 1 - Wireshark

- Ping Response:**
 The broker (91.121.93.94) sends a Ping Response to the client (192.168.142.4), confirming the MQTT connection is still active.
- Publish Message (id=4):**
 The client (192.168.142.4) sends a Publish message with QoS 1, containing the payload "oranges" and message ID 4, ensuring that the broker receives the message at least once.
- Publish Acknowledgment (id=4):**
 The broker (91.121.93.94) acknowledges the receipt of the Publish message by sending a Publish Ack with the same message ID 4, confirming successful delivery.

No.	Time	Source	Destination	Protocol	Length	Info	Source Port
5	14:02:49.276412	192.168.208.102	91.121.93.94	MQTT	56	Ping Request	60926
6	14:02:49.520942	91.121.93.94	192.168.208.102	MQTT	56	Ping Response	mqtt
10	14:02:50.164096	91.121.93.94	192.168.208.102	MQTT	76	Publish Message (id=1) [oranges]	mqtt
12	14:02:50.164442	192.168.208.102	91.121.93.94	MQTT	58	Publish Ack (id=1)	60926
19	14:02:55.204206	192.168.208.102	91.121.93.94	MQTT	56	Ping Request	60926
21	14:02:55.447408	91.121.93.94	192.168.208.102	MQTT	56	Ping Response	mqtt

Figure 17: Subscriber QoS 1 - Wireshark


```

> Frame 30: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface \Device\NPF_{82BF7957-6334-4670-8DA6-CB55126FD443},
> Ethernet II, Src: Intel_8a:ec:f6 (b0:60:88:8a:ec:f6), Dst: 12:d9:8e:78:a9:74 (12:d9:8e:78:a9:74)
> Internet Protocol Version 4, Src: 192.168.142.4, Dst: 91.121.93.94
> Transmission Control Protocol, Src Port: 28035, Dst Port: 1883, Seq: 3, Ack: 3, Len: 19
√ MQ Telemetry Transport Protocol, Publish Message
  > [Expert Info (Note/Protocol): Unknown version (missing the CONNECT packet?)]
  √ Header Flags: 0x32, Message Type: Publish Message, QoS Level: At least once delivery (Acknowledged deliver)
    0011 .... = Message Type: Publish Message (3)
    .... 0... = DUP Flag: Not set
    .... .01. = QoS Level: At least once delivery (Acknowledged deliver) (1)
    .... ...0 = Retain: Not set
  Msg Len: 17
  Topic Length: 7
  Topic: oranges
  Message Identifier: 4
  Message: 53616e756a61

```

Figure 18: Subscriber QoS 1 - Publish Message

The MQTT Publish message has the topic "oranges" with QoS Level 1, meaning the message will be delivered at least once with acknowledgment. The message identifier is 4, and the payload of the message is "53616e756a61" (hexadecimal representation). The message does not have the DUP or Retain flags set, indicating it is not a duplicate and should not be stored by the broker after delivery.

```

> Frame 12: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface \Device\NPF_{45C26253-EEA1-4751-B7AB-8E5E}
> Ethernet II, Src: IntelCor_f5:71:0c (e0:2b:e9:f5:71:0c), Dst: MS-NLB-PhysServer-32_07:42:a8:ba:1e (02:27:42:a8:ba:1e)
> Internet Protocol Version 4, Src: 192.168.208.102, Dst: 91.121.93.94
> Transmission Control Protocol, Src Port: 60926 (60926), Dst Port: mqtt (1883), Seq: 3, Ack: 25, Len: 4
√ MQ Telemetry Transport Protocol, Publish Ack
  > [Expert Info (Note/Protocol): Unknown version (missing the CONNECT packet?)]
  √ Header Flags: 0x40, Message Type: Publish Ack
    0100 .... = Message Type: Publish Ack (4)
    .... 0000 = Reserved: 0
  Msg Len: 2
  Message Identifier: 1

```

Figure 19: Subscriber QoS 1 - Publish ACK

```

> Frame 10: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface \Device\NPF_{45C26253-EEA1-4751-B7AB-8E5E}
> Ethernet II, Src: MS-NLB-PhysServer-32_07:42:a8:ba:1e (02:27:42:a8:ba:1e), Dst: IntelCor_f5:71:0c (e0:2b:e9:f5:71:0c)
> Internet Protocol Version 4, Src: 91.121.93.94, Dst: 192.168.208.102
> Transmission Control Protocol, Src Port: mqtt (1883), Dst Port: 60926 (60926), Seq: 3, Ack: 3, Len: 22
√ MQ Telemetry Transport Protocol, Publish Message
  > [Expert Info (Note/Protocol): Unknown version (missing the CONNECT packet?)]
  √ Header Flags: 0x32, Message Type: Publish Message, QoS Level: At least once delivery (Acknowledged deliver)
    0011 .... = Message Type: Publish Message (3)
    .... 0... = DUP Flag: Not set
    .... .01. = QoS Level: At least once delivery (Acknowledged deliver) (1)
    .... ...0 = Retain: Not set
  Msg Len: 20
  Topic Length: 7
  Topic: oranges
  Message Identifier: 1
  Message: 537269204c616e6b61

```

Figure 20: Subscriber QoS 1 - Publish Message

- **Ping Request:**

The client (192.168.208.102) sends a Ping Request to the broker (91.121.93.94) to verify the MQTT connection is still active.

- **Ping Response:**

The broker (91.121.93.94) replies to the client's Ping Request, confirming connection is alive.

- **Publish Message (id=1):**

The broker (91.121.93.94) sends a Publish message containing "oranges" with message ID 1 to the client (192.168.208.102) under QoS 1.

- **Publish Acknowledgment (id=1):**

The client (192.168.208.102) sends a Publish Ack with message ID 1 back to the broker (91.121.93.94), confirming successful receipt of the message.

4 QoS=2 instance

99	3.347005	192.168.142.4	91.121.93.94	MQTT	70 Publish Message (id=2) [oranges]
100	3.541833	91.121.93.94	192.168.142.4	MQTT	58 Publish Received (id=2)
101	3.542431	192.168.142.4	91.121.93.94	MQTT	58 Publish Release (id=2)
102	3.726005	91.121.93.94	192.168.142.4	MQTT	58 Publish Complete (id=2)

Figure 21: Publisher QoS 2 - Wireshark

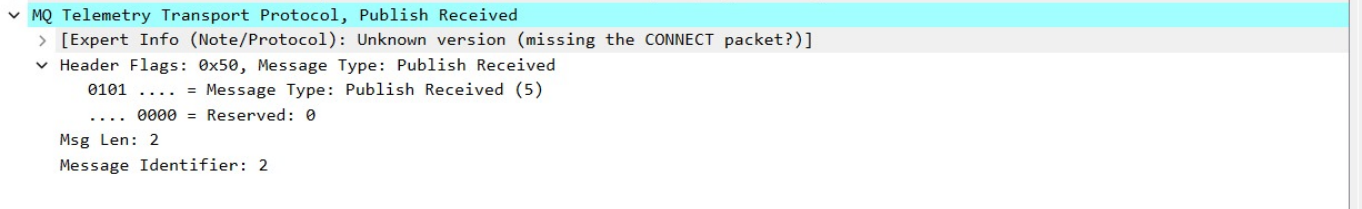


Figure 22: QoS 2 - Publish Received

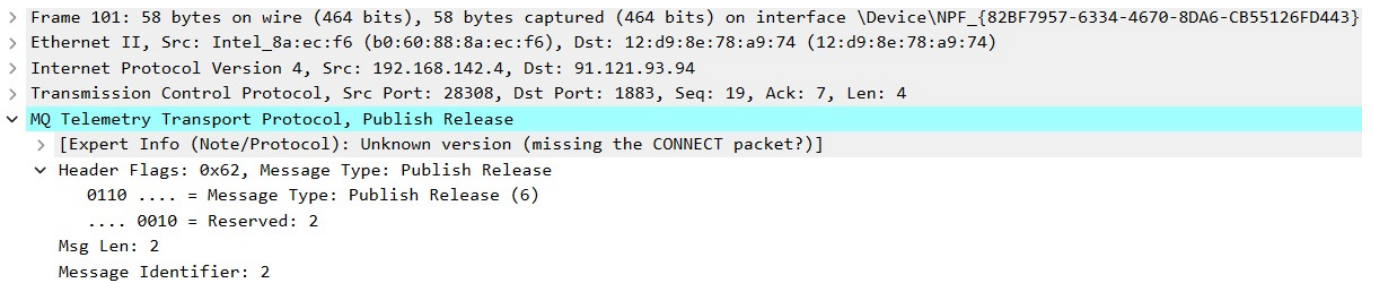


Figure 23: QoS 2 - Publish Release

- Publish Message (id=2):**
 The client (192.168.142.4) sends a Publish message containing "oranges" with message ID 2 to the broker (91.121.93.94) under QoS 2, ensuring the message is delivered exactly once.
- Publish Received (id=2):**
 The broker (91.121.93.94) acknowledges the receipt of the Publish message by sending a Publish Received message with message ID 2 back to the client, indicating that the message has been received but not yet completely processed.
- Publish Release (id=2):**
 The client (192.168.142.4) responds to the broker's acknowledgment by sending a Publish Release message with message ID 2, indicating that the message can now be processed and delivered.
- Publish Complete (id=2):**
 The broker (91.121.93.94) confirms that the message processing is complete by sending a Publish Complete message with message ID 2 back to the client, finalizing the QoS 2 message flow.

No.	Time	Source	Destination	Protocol	Length	Info	Source Port
3	14:10:03.670927	192.168.208.102	91.121.93.94	MQTT	56	Ping Request	60962
4	14:10:03.847416	91.121.93.94	192.168.208.102	MQTT	56	Ping Response	mqtt
8	14:10:08.909333	192.168.208.102	91.121.93.94	MQTT	56	Ping Request	60962
9	14:10:08.927878	91.121.93.94	192.168.208.102	MQTT	70	Publish Message (id=4) [oranges]	mqtt
13	14:10:09.087534	91.121.93.94	192.168.208.102	MQTT	56	Ping Response	mqtt
14	14:10:09.087632	192.168.208.102	91.121.93.94	MQTT	58	Publish Received (id=4)	60962
15	14:10:09.247864	91.121.93.94	192.168.208.102	MQTT	58	Publish Release (id=4)	mqtt
16	14:10:09.248158	192.168.208.102	91.121.93.94	MQTT	58	Publish Complete (id=4)	60962
23	14:10:14.289209	192.168.208.102	91.121.93.94	MQTT	56	Ping Request	60962
25	14:10:14.487597	91.121.93.94	192.168.208.102	MQTT	56	Ping Response	mqtt

Figure 24: Subscriber QoS 2 - Wireshark

Figure 25: Subscriber QoS 2 - Publish Message

Figure 26: Subscriber QoS 2 - Publish Received

Figure 27: Subscriber QoS 2 - Publish Release

Figure 28: Subscriber QoS 2 - Publish Complete

- **Ping Response:**

The broker (91.121.93.94) sends a Ping Response to the client (192.168.208.102), confirming the MQTT connection is still active.

- **Publish Message (id=4):**

The client (192.168.208.102) sends a Publish message containing "oranges" with message ID 4 to the broker (91.121.93.94) under QoS 2.

- **Publish Received (id=4):**

The broker (91.121.93.94) acknowledges the receipt of the Publish message by sending a Publish Received message with message ID 4 back to the client.

- **Publish Release (id=4):**

The client (192.168.208.102) sends a Publish Release message with message ID 4 to the broker, indicating the message can be processed.

- **Publish Complete (id=4):**

The broker (91.121.93.94) confirms the message processing is complete by sending a Publish Complete message with message ID 4 back to the client.

5 Codes

5.1 Publisher

```
1 from paho.mqtt import client as mqtt_client
2 import paho.mqtt.client as mqtt
3 import time
4
5 # Callback when the client connects to the MQTT broker
6 def on_connect(client, userdata, flags, rc):
7     if rc == 0:
8         print("Connected to MQTT broker\n")
9     else:
10        print("Connection failed with code {rc}")
11
12
13 # Create an MQTT client instance
14 client = mqtt.Client(mqtt_client.CallbackAPIVersion.VERSION1, "PythonPub")
15
16 # Set the callback function
17 client.on_connect = on_connect
18
19 broker_address = "test.mosquitto.org" # broker's address
20 broker_port = 1883
21 keepalive = 5
22 qos = 1
23 publish_topic = "oranges"
24
25 # Connect to the MQTT broker
26 client.connect(broker_address, broker_port, keepalive)
27
28 # Start the MQTT loop to handle network traffic
29 client.loop_start()
30
31 # Publish loop
32
33 try:
34     while True:
35         # Publish a message to the send topic
36
37         value = input('Enter the message: ')
38         client.publish(publish_topic, value, qos)
39         print(f"Published message '{value}' to topic '{publish_topic}'\n")
40
41         # Wait for a moment to simulate some client activity
42         time.sleep(6)
43
44 except KeyboardInterrupt:
45     # Disconnect from the MQTT broker
46     pass
47 client.loop_stop()
48 client.disconnect()
49
50 print("Disconnected from the MQTT broker")
```

Listing 1: Publisher Code

5.2 Subscriber

```
1 from paho.mqtt import client as mqtt_client
2 import paho.mqtt.client as mqtt
3 import time
4
5 # Callback when the client connects to the MQTT broker
6 def on_connect(client, userdata, flags, rc):
7     if rc == 0:
8         print("Connected to MQTT broker")
9         client.subscribe(subscribe_topic, qos) # Subscribe to the receive topic
10    else:
11        print("Connection failed with code {rc}")
12
13 # Callback when a message is received from the subscribed topic
14 def on_message(client, userdata, msg):
15     print("Message received " + "on " + subscribe_topic + ": " +
16           str(msg.payload.decode("utf-8")))
17
18 # Create an MQTT client instance
19 client = mqtt.Client(mqtt_client.CallbackAPIVersion.VERSION1, "PythonSub")
20
21 # Set the callback functions
22 client.on_connect = on_connect
23 client.on_message = on_message
24
25 # Connect to the MQTT broker
26 broker_address = "test.mosquitto.org" # broker's address
27 broker_port = 1883
28 keepalive = 5
29 qos = 1
30
31 subscribe_topic = "oranges"
32 client.connect(broker_address, broker_port, keepalive)
33
34 # Start the MQTT loop to handle network traffic
35 client.loop_start()
36
37 # Subscribe loop
38 try:
39     while True:
40         time.sleep(6)
41 except KeyboardInterrupt:
42     # Disconnect from the MQTT broker
43     pass
44 client.loop_stop()
45 client.disconnect()
46
47 print("Disconnected from the MQTT broker")
```

Listing 2: Subscriber Code

6 Homework - Smart Plant Watering System

This project implements a Smart Plant Watering System using MQTT, which automates the process of watering a plant based on soil moisture and water level readings. The system consists of a controller, moisture sensor, water level sensor, and a water pump, all communicating via MQTT. The controller makes decisions to start or stop the water pump based on real-time data from the sensors, ensuring the plant is adequately watered while conserving resources. This project demonstrates the effective use of MQTT in IoT-based automation.

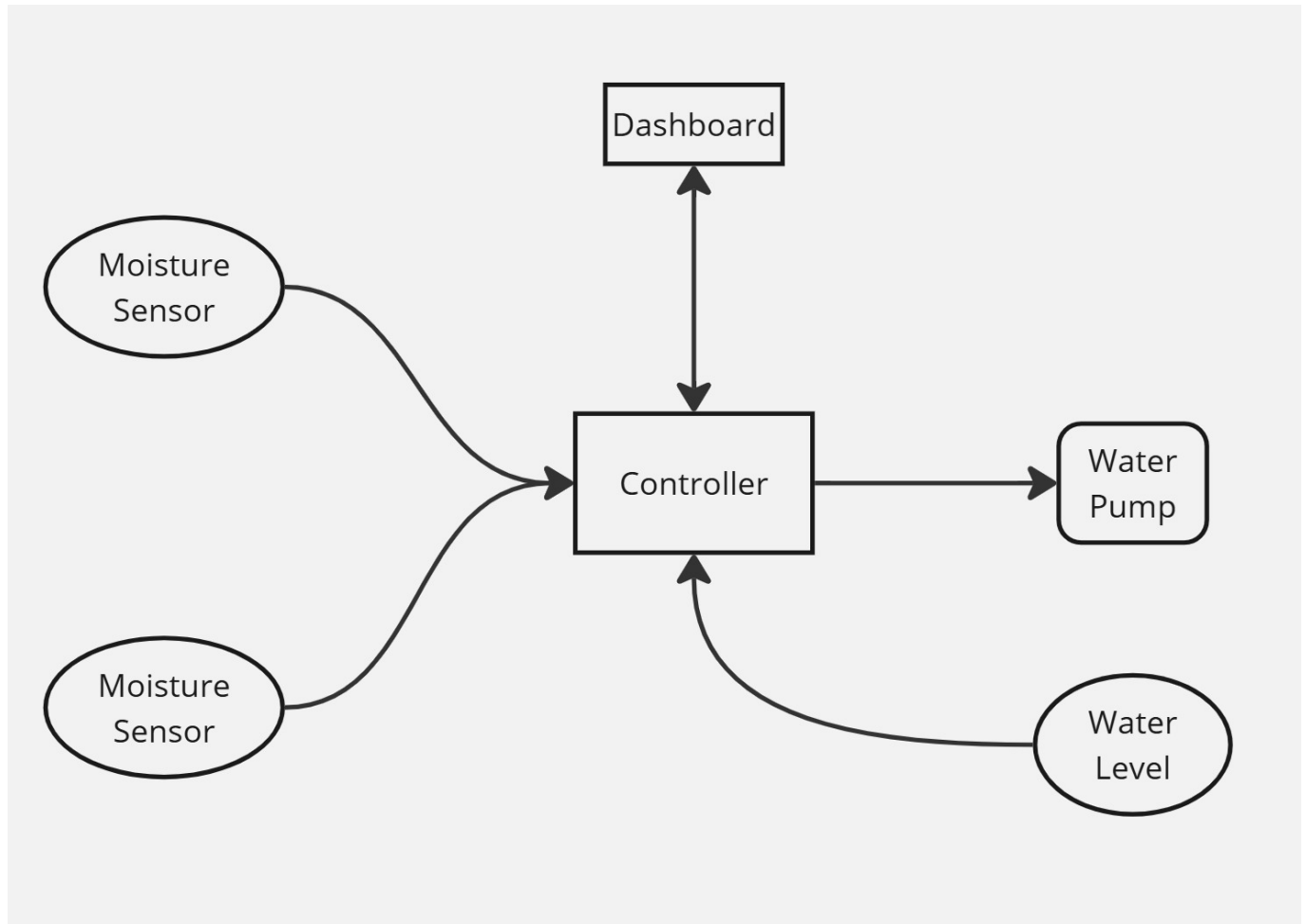


Figure 29: System Architecture

6.1 Controller Code

The controller code serves as the central unit of the Smart Plant Watering System. It is responsible for subscribing to MQTT topics to receive data from the soil moisture sensor and water level sensor. The controller then processes this data to decide whether to activate or deactivate the water pump. Additionally, the controller publishes status updates and commands to relevant MQTT topics, allowing other components to respond accordingly. The primary goal of this code is to ensure that the plant receives adequate water without depleting the water supply.

```

1 from paho.mqtt import client as mqtt_client
2 import paho.mqtt.client as mqtt
3 import time
4 import os
5 from dotenv import load_dotenv
6
7 # Callback when the client connects to the MQTT broker
8 def on_connect(client, userdata, flags, rc):
9     if rc == 0:
10         print("Connected to MQTT broker\n")
11         client.subscribe("Sensor_1", qos)
12         client.subscribe("Water_level", qos)
13     else:
14         print("Connection failed with code {rc}")
15
16 def on_message(client, userdata, msg):
17     global sensor_data
18     global water_level
19     topic = msg.topic
20     if topic == "Sensor_1":
21         sensor_data = str(msg.payload.decode("utf-8"))
22     elif topic == "Water_level":
23         water_level = str(msg.payload.decode("utf-8"))
24     else:
25         print("Unknown topic")
26
27 # Create an MQTT client instance
28 client = mqtt.Client(client_id="controller", clean_session=True)
29
30 # Set the callback function
31 client.on_connect = on_connect
32 client.on_message = on_message
33
34 # Using a Mosquitto broker hosted on a server
35 broker_address = "137.184.9.146" # broker's address
36 broker_port = 1883
37 keepalive = 5
38 qos = 1
39
40 publish_topic = "RPS_client_to_server"
41 subscribe_topic = "RPS_server_to_client"
42
43 load_dotenv() # Load the environment variables
44 username = os.environ['MQTT_USERNAME']
45 password = os.environ['MQTT_PASSWORD']
46
47 client.username_pw_set(username, password)
48
49 # Connect to the MQTT broker
50 client.connect(broker_address, broker_port, keepalive)
51
52 # Start the MQTT loop to handle network traffic
53 client.loop_start()

```

```
54
55 # Publish loop
56 time.sleep(2) # Wait for the connection to establish
57 sensor_data = "High"
58 water_level = "High"
59
60 try:
61     while True:
62         if sensor_data == "Low":
63             if water_level == "Low":
64                 print("Water level is low, please refill the tank")
65                 client.publish("Dashboard", "Water level is low, please refill
66                     the tank", qos)
67             else:
68                 print("Water level is normal. Water Pump Started")
69                 client.publish(publish_topic, "Start", qos)
70         else:
71             print("Soil moisture is sufficient")
72             client.publish(publish_topic, "Stop", qos)
73         time.sleep(3) # Wait for 3 seconds
74
75 except KeyboardInterrupt:
76     # Disconnect from the MQTT broker
77     pass
78 client.loop_stop()
79 client.disconnect()
80
81 print("Disconnected from the MQTT broker")
```

Listing 3: Controller Code

6.2 Moisture Sensor Code

The moisture sensor code reads the soil moisture levels and communicates this data to the MQTT broker. This code is designed to operate continuously, monitoring the soil's moisture content and reporting it in real time. Based on the moisture level, the controller decides whether the plant requires watering. The simplicity of this code lies in its focused task—accurately reporting soil moisture to enable efficient water management.

```

1 from paho.mqtt import client as mqtt_client
2 import paho.mqtt.client as mqtt
3 import time
4 import os
5 from dotenv import load_dotenv
6
7 # Callback when the client connects to the MQTT broker
8 def on_connect(client, userdata, flags, rc):
9     if rc == 0:
10         print("Connected to MQTT broker\n")
11         client.subscribe("Pump", qos)
12     else:
13         print("Connection failed with code {rc}")
14
15 def on_message(client, userdata, msg):
16     message = str(msg.payload.decode("utf-8"))
17     global pump
18     if message == "Start":
19         pump = "start"
20     elif message == "Stop":
21         pump = "stop"
22
23 # Create an MQTT client instance
24 client = mqtt.Client(client_id="pump", clean_session=True)
25
26 # Set the callback function
27 client.on_connect = on_connect
28 client.on_message = on_message
29
30 # Using a Mosquitto broker hosted on a server
31 broker_address = "137.184.9.146" # broker's address
32 broker_port = 1883
33 keepalive = 5
34 qos = 1
35
36 load_dotenv() # Load the environment variables
37 username = os.environ['MQTT_USERNAME']
38 password = os.environ['MQTT_PASSWORD']
39 client.username_pw_set(username, password)
40
41 # Connect to the MQTT broker
42 client.connect(broker_address, broker_port, keepalive)
43
44 # Start the MQTT loop to handle network traffic
45 client.loop_start()
46
47 # Publish loop
48 time.sleep(2) # Wait for the connection to establish
49
50 def pump_start():
51     print("Water Pump Started")
52
53 def pump_stop():
54     print("Water Pump Stopped")
55

```

```
56 pump = "stop"
57 try:
58     while True:
59         time.sleep(3)
60         if pump == "start":
61             pump_start()
62             pump = "stop" # Reset the pump status in case of controller failure
63         elif pump == "stop":
64             pump_stop()
65
66 except KeyboardInterrupt:
67     # Disconnect from the MQTT broker
68     pass
69 client.loop_stop()
70 client.disconnect()
71
72 print("Disconnected from the MQTT broker")
```

Listing 4: Moisture Sensor Code

6.3 Water Level Sensor Code

The water level sensor code is responsible for monitoring the water level in the tank. It continuously checks the tank's water level and publishes this information to the MQTT broker. The controller uses this data to determine if there is sufficient water to activate the water pump. This code is critical in ensuring that the pump does not run dry, thereby preventing potential damage and ensuring the system's reliability.

```

1 from paho.mqtt import client as mqtt_client
2 import paho.mqtt.client as mqtt
3 import time
4 import os
5 from dotenv import load_dotenv
6
7 # Callback when the client connects to the MQTT broker
8 def on_connect(client, userdata, flags, rc):
9     if rc == 0:
10         print("Connected to MQTT broker\n")
11         client.subscribe("Water_level", qos)
12     else:
13         print("Connection failed with code {rc}")
14
15 def on_message(client, userdata, msg):
16     water_level = str(msg.payload.decode("utf-8"))
17     global level
18     level = water_level
19
20 # Create an MQTT client instance
21 client = mqtt.Client(client_id="water_level_sensor", clean_session=True)
22
23 # Set the callback function
24 client.on_connect = on_connect
25 client.on_message = on_message
26
27 # Using a Mosquitto broker hosted on a server
28 broker_address = "137.184.9.146" # broker's address
29 broker_port = 1883
30 keepalive = 5
31 qos = 1
32
33 load_dotenv() # Load the environment variables
34 username = os.environ['MQTT_USERNAME']
35 password = os.environ['MQTT_PASSWORD']
36 client.username_pw_set(username, password)
37
38 # Connect to the MQTT broker
39 client.connect(broker_address, broker_port, keepalive)
40
41 # Start the MQTT loop to handle network traffic
42 client.loop_start()
43
44 # Publish loop
45 time.sleep(2) # Wait for the connection to establish
46 level = "High"
47
48 try:
49     while True:
50         client.publish("Water_level", level, qos)
51         time.sleep(3) # Publish water level every 3 seconds
52
53 except KeyboardInterrupt:
54     # Disconnect from the MQTT broker
55     pass

```

```
56 client.loop_stop()  
57 client.disconnect()  
58  
59 print("Disconnected from the MQTT broker")
```

Listing 5: Water Level Sensor Code

6.4 Water Pump Code

The water pump code controls the activation and deactivation of the water pump based on commands received from the controller via MQTT. When the controller sends a "Start" command, the pump is activated to water the plant. Conversely, a "Stop" command deactivates the pump. This code ensures that the pump operates only when necessary, conserving water and power while maintaining the health of the plant.

```

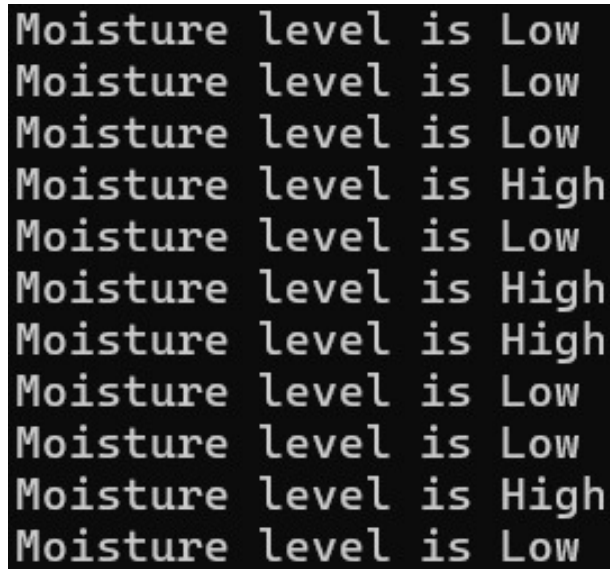
1 from paho.mqtt import client as mqtt_client
2 import paho.mqtt.client as mqtt
3 import time
4 import os
5 from dotenv import load_dotenv
6
7 # Callback when the client connects to the MQTT broker
8 def on_connect(client, userdata, flags, rc):
9     if rc == 0:
10         print("Connected to MQTT broker\n")
11         client.subscribe("Pump", qos)
12     else:
13         print("Connection failed with code {rc}")
14
15 def on_message(client, userdata, msg):
16     pump_command = str(msg.payload.decode("utf-8"))
17     global pump
18     pump = pump_command
19
20 # Create an MQTT client instance
21 client = mqtt.Client(client_id="water_pump", clean_session=True)
22
23 # Set the callback function
24 client.on_connect = on_connect
25 client.on_message = on_message
26
27 # Using a Mosquitto broker hosted on a server
28 broker_address = "137.184.9.146" # broker's address
29 broker_port = 1883
30 keepalive = 5
31 qos = 1
32
33 load_dotenv() # Load the environment variables
34 username = os.environ['MQTT_USERNAME']
35 password = os.environ['MQTT_PASSWORD']
36 client.username_pw_set(username, password)
37
38 # Connect to the MQTT broker
39 client.connect(broker_address, broker_port, keepalive)
40
41 # Start the MQTT loop to handle network traffic
42 client.loop_start()
43
44 # Publish loop
45 time.sleep(2) # Wait for the connection to establish
46 pump = "stop"
47
48 try:
49     while True:
50         if pump == "start":
51             print("Water Pump Started")
52             # Code to activate the water pump hardware
53             pump = "stop" # Reset pump status
54         elif pump == "stop":
55             print("Water Pump Stopped")

```

```
56         # Code to deactivate the water pump hardware
57         time.sleep(3) # Loop every 3 seconds
58
59 except KeyboardInterrupt:
60     # Disconnect from the MQTT broker
61     pass
62 client.loop_stop()
63 client.disconnect()
64
65 print("Disconnected from the MQTT broker")
```

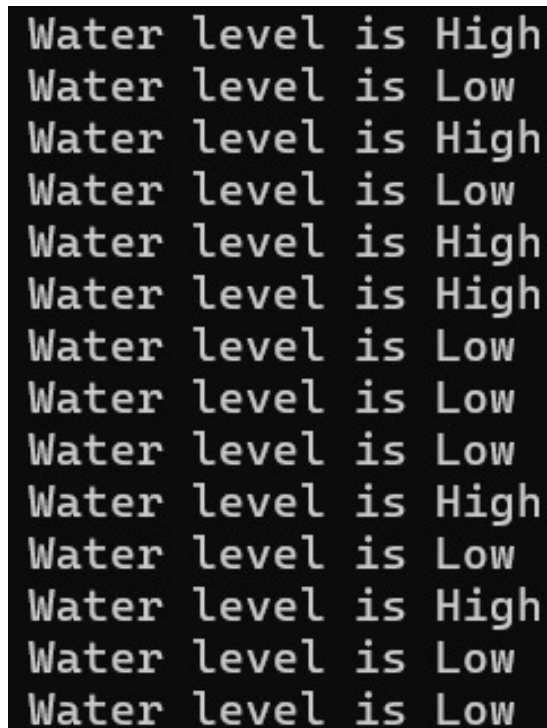
Listing 6: Water Pump Code

6.5 Outputs



```
Moisture level is Low
Moisture level is Low
Moisture level is Low
Moisture level is High
Moisture level is Low
Moisture level is High
Moisture level is High
Moisture level is Low
Moisture level is Low
Moisture level is High
Moisture level is Low
```

Figure 30: Moisture Sensor



```
Water level is High
Water level is Low
Water level is High
Water level is Low
Water level is High
Water level is High
Water level is Low
Water level is Low
Water level is Low
Water level is High
Water level is Low
Water level is High
Water level is Low
Water level is Low
```

Figure 31: Water Level

```
Water Pump Started
Water Pump Stopped
Water Pump Started
Water Pump Stopped
Water Pump Stopped
Water Pump Started
Water Pump Started
Water Pump Stopped
Water Pump Started
Water Pump Stopped
Water Pump Stopped
Water Pump Started
Water Pump Stopped
Water Pump Started
Water Pump Started
Water Pump Started
Water Pump Started
```

Figure 32: Water Pump

```
Moisture level is high, Water Pump Stopped
Water level is normal.Water Pump Started
Moisture level is high, Water Pump Stopped
Water level is normal.Water Pump Started
Moisture level is high, Water Pump Stopped
Moisture level is high, Water Pump Stopped
Water level is normal.Water Pump Started
Water level is normal.Water Pump Started
Moisture level is high, Water Pump Stopped
Water level is normal.Water Pump Started
Moisture level is high, Water Pump Stopped
Moisture level is high, Water Pump Stopped
Water level is normal.Water Pump Started
Moisture level is high, Water Pump Stopped
```

Figure 33: Controller