A close-up of a logo

Description automatically generated

|  |  |
| --- | --- |
| A person holding a tablet in a greenhouse  Description automatically generated  AgriSmart House: IoT-Driven Greenhouse Automation | **TECHNICAL REPORT**  Mary Hannoush & Sami Sulaiman |

**Abstract**

Greenhouse management is crucial for agricultural efficiency, yet traditional methods often lack real-time control and automation. AgriSmart House is a smart greenhouse system leveraging IoT technologies to monitor environmental parameters such as temperature, humidity, and light, automating control through actuators. Using a Raspberry Pi as an MQTT broker and integrating Arduino, ESP8266, and XBee modules, the system provides remote monitoring via HiveMQ and mobile MQTT applications. This project showcases a scalable, energy-efficient solution to modernize greenhouse operations.

**Introduction**

The Internet of Things (IoT) is transforming agriculture by introducing intelligent systems that enhance efficiency, precision, and scalability. Among the sectors benefitting most from IoT technologies, greenhouse management stands out as a key area where real-time environmental monitoring and control can optimize crop production and resource utilization. Traditional greenhouse operations, which often rely on manual monitoring and adjustment, are not only labor-intensive but also prone to errors, making them unsuitable for large-scale or precision agriculture.

Effective greenhouse management involves the continuous regulation of temperature, humidity, light intensity, and other environmental factors to ensure optimal conditions for crop growth. However, achieving this balance manually is time-consuming and often results in resource inefficiencies. IoT-based systems offer a transformative solution by automating these tasks through interconnected devices and sensors.

This project, AgriSmart House, addresses the challenges of traditional greenhouse management by integrating IoT technologies into a smart automation system. The solution uses sensors to monitor environmental parameters, actuators to respond to changes, and reliable communication protocols to ensure seamless data exchange and control. Centralized data aggregation and processing occur via a Raspberry Pi acting as an MQTT broker, while XBee and ESP8266 modules facilitate device communication. The system’s capabilities extend to real-time data visualization and remote control through HiveMQ and mobile MQTT applications.

By combining automation, scalability, and energy efficiency, AgriSmart House minimizes human intervention while optimizing greenhouse operations. The project highlights the potential of IoT to revolutionize agricultural practices, providing a sustainable, modern approach to greenhouse management.

* **General Topology:**

A computer chip with different components

Description automatically generated with medium confidence

As shown in our topology, the AgriSmart House system consists of several microcontrollers that communicate through MQTT and ZigBee protocols to manage and exchange sensor data. We have ESP8266 modules, Arduino Uno, and a Raspberry Pi that work together to monitor environmental parameters and control actuators.

The first ESP8266 is connected to a PIR sensor and an HC-SR04 proximity sensor. This microcontroller collects motion and distance data, which it publishes to the MQTT broker on the Raspberry Pi. The data is sent to the topics .Once the data is published, the Raspberry Pi subscribes to these topics and uses the information to control actuators, such as activating a fan when motion is detected or open the light when there is motion detected and the temperature is normal ( in our case let the relay on).

The second ESP8266 is linked to a DHT22 sensor, which measures temperature and humidity. This data is sent to the MQTT topics .Additionally, this ESP8266 subscribes to the topic of the relay to activate a relay and buzzer when specific threshold values are met, providing an alert if environmental conditions go out of range.

An Arduino Uno is used in the system to manage actuators connected to it. The Arduino is connected to an MH light intensity sensor and sends its data to the Raspberry Pi using ZigBee communication. This communication is facilitated by two XBee modules—one acting as a coordinator connected to the Raspberry Pi and the other as an end device connected to the Arduino. The Arduino can also activate actuators like relays or buzzers based on the sensor data processed by the Raspberry Pi.

The Raspberry Pi functions as the central server in the system, acting as the MQTT broker using Mosquitto. It is responsible for processing the data from the ESP8266 modules and Arduino, storing it in a database, and subscribing to relevant topics to control actuators. Additionally, the Raspberry Pi connects to a cloud MQTT broker, allowing remote access to sensor data and actuator control from anywhere. This cloud connection enables seamless integration and monitoring of the greenhouse system.

* **Implementation**
* **Hardware Implementation**
* Arduino connected to MH sensor and XBee module

|  |  |  |  |
| --- | --- | --- | --- |
| Device | pin | Connected Component | Purpose/Function |
| |  | | --- | | **Arduino Uno** |  |  | | --- | |  | | A0 | MH Light Sensor (AO pin) | Reads analog value from the light sensor. |
| TX | XBee Shield (DIN pin) | Transmits sensor data to Raspberry Pi via XBee. |
| RX | XBee Shield (DOUT pin) | Receives messages from Raspberry Pi via XBee. |
| 5V, GND | XBee Shield | Provides power and ground for XBee shield. |
| 5V, GND | MH Light Sensor | Provides power and ground to the light sensor. |
| USB Port | PC | Debugging and programming the Arduino. |

* ESP8266 devices interfaced with PIR, HC, DHT22, relay, and buzzer:

|  |  |  |  |
| --- | --- | --- | --- |
| Device | pin | Connected Component | Purpose/Function |
| |  | | --- | | **ESP8266 (Module 1)** |  |  | | --- | |  | | GPIO2 (D4) | PIR Motion Sensor (OUT pin) | Detects motion and publishes to MQTT topic. |
|  | GPIO12 (D6) | HC-SR04 Ultrasonic Sensor (Trig pin) | Sends trigger pulse for distance measurement. |
|  | GPIO14 (D5) | HC-SR04 Ultrasonic Sensor (Echo pin) | Reads echo pulse to calculate distance. |
|  | 5V, GND | HC-SR04 Ultrasonic Sensor | Provides power and ground to ultrasonic sensor |
|  | 5V, GND | PIR Motion Sensor | Provides power and ground to PIR motion sensor. |
| ESP8266 (Module 2) | GPIO2 (D4) | DHT22 Temperature Sensor (DATA pin) | Reads temperature and humidity data. |
|  | |  | | --- | | 3.3V, GND |  |  | | --- | |  | | Relay | Provides power and ground to relay. |
| Relay | Control Signal Pin | ESP8266 GPIO5 (D1) | Activates or deactivates connected devices based on control signals from MQTT. |
|  | 3.3V, GND | ESP8266 | |  | | --- | |  |  |  | | --- | | Powers the relay for controlling external electrical appliances. | |
|  | Relay NO | Buzzer VCC | Connects relay to the power supply to drive external devices (e.g., buzzer). |
|  | Relay COM | 3.3V or 5V Power Supply | Connects relay to the power supply to drive external devices (e.g., buzzer). |
| Buzzer | GND | ESP8266 GND | Connects the buzzer to ground for stable operation. |

* Raspberry Pi connected to XBee for wireless data collection:

|  |  |  |  |
| --- | --- | --- | --- |
| Device | pin | Connected Component | Purpose/Function |
| |  | | --- | | **Raspberry pi 3** |  |  | | --- | |  | | 5V (GPIO Pin 2 or 4) | XBee USB Adapter (VCC) | Provides power to the XBee USB adapter. |
| GND (GPIO Pin 6 or 9) | XBee USB Adapter (GND) | Establishes a common ground connection for proper communication. |
| TXD (GPIO Pin 8) | XBee USB Adapter (RX) | Transmits data from the Raspberry Pi to the XBee USB adapter. |
| RXD (GPIO Pin 10) | XBee USB Adapter (TX) | Receives data from the XBee USB adapter to the Raspberry Pi. |

* **Temperature and Humidity Monitoring (DHT22 Sensor)**

An ESP8266 module is connected to a DHT22 sensor to monitor temperature and humidity within the greenhouse. The configuration involves connecting to the MQTT broker hosted on the Raspberry Pi. The following steps detail the setup:

**MQTT Configuration** & **Wi-Fi Configuration**:

* The IP address and port of the Raspberry Pi’s Mosquitto broker are specified.
* The ESP8266 connects to the local Wi-Fi network using the provided SSID and password, enabling seamless communication with the broker.

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

**Publishing Data**:

* The client.publish method is used to encapsulate the sensor data into MQTT messages.
* Temperature data is published to the topic /greenhouse/temperature, while humidity data is published to /greenhouse/humidity.

A computer screen shot of text

Description automatically generated

**Subscription for Actuation**:

* The ESP subscribes to the /greenhouse/relay topic, which carries actuation commands to activate or disactivate the relay.

**A screenshot of a computer program

Description automatically generated**

* **Light Intensity Monitoring (MH Sensor)**

An Arduino Uno equipped with an XBee module collects light intensity data from an MH sensor. The configuration includes:

**XBee Setup**:

To configure the XBEE modules, we need to set how each one will operate using the XCTU software. The first module is configured as a coordinator, and this module will be connected to the Raspberry Pi. To configure the module, a PAN ID is set, which must be the same for XBEE modules; in this case, we configured it as 1410. Additionally, the CE field is configured, which allows the module to be the coordinator, and this field must be enabled."

* **Coordinator (Raspberry Pi)**: Configured to receive data from the XBee module connected to the Arduino. The Raspberry Pi processes the received data and publishes it to /greenhouse/light.

A screenshot of a computer

Description automatically generated

* **End Device (Arduino Uno)**: Transmits light intensity readings via XBee to the Raspberry Pi.

A screenshot of a computer

Description automatically generated

**Publishing Data**:

The light intensity data is formatted and sent to the Raspberry Pi in real-time. The Raspberry Pi then publishes this data to the MQTT broker for remote access.

A screenshot of a computer program

Description automatically generated

* **Motion and Proximity Detection (PIR and Ultrasonic Sensors)**

**Motion Detection:**

An ESP8266 module connected to a PIR sensor monitors motion within the greenhouse:

**Wi-Fi and MQTT Configuration**:

* The ESP is set up to connect to the Raspberry Pi’s MQTT broker over Wi-Fi.
* the motion data is published to /greenhouse/motion.

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

**Data Publishing**

When motion is detected, the PIR sensor sends a signal to the ESP, which encapsulates the message as MOTION\_DETECTED and publishes it to the topic.

If no motion is detected, a NO\_MOTION message is sent.

A screen shot of a computer code

Description automatically generated

**Proximity Detection:**

The HC-SR04 ultrasonic sensor, also connected to an ESP8266, monitors proximity:

**Wi-Fi and MQTT Setup**:

Similar to the PIR configuration, the ESP connects to the MQTT broker and publishes data to /greenhouse/distance.

**Data Processing**:

The ESP calculates the distance from the HC-SR04 readings and publishes this value.

A computer screen shot of text

Description automatically generated

**Subscription and Actuation**:

Based on proximity data, commands can be sent to /greenhouse/relay to trigger specific actions, such as turning on lights or activating a buzzer.

**Actuator Control**

The relay and buzzer are controlled via the /greenhouse/relay topic:

**Relay Operation**:

* When the Raspberry Pi publishes RELAY\_ON, the subscribed ESP activates the relay to power devices like fans or pumps.
* For RELAY\_OFF, the relay deactivates.

**Buzzer Alerts**:

* The buzzer is triggered based on specific conditions, such as motion detection or critical temperature thresholds.
* Example: If temperature less than the min temperature , the Raspberry Pi publishes BUZZER\_ON to warn users.

**Data Visualization and Remote Access**

**HiveMQ Integration**

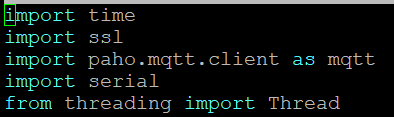
The system integrates HiveMQ to provide remote monitoring via mobile applications:

**Publishing to HiveMQ Cloud**:

* The Raspberry Pi acts as a client to the HiveMQ broker, replicating sensor data from local MQTT topics to corresponding cloud topics.
* Users can subscribe to topics like /greenhouse/temperature, /greenhouse/motion, and /greenhouse/light via web client MQTT to monitor real-time data.
* **Raspberry Configuration:**

**1. Imports and Configuration**

This section imports the essential libraries and modules necessary for the program's operation. The time module is used to introduce delays in the execution where necessary. The ssl library provides secure communication by enabling the use of TLS (Transport Layer Security) for connecting to the cloud MQTT broker securely. The paho.mqtt.client library is a widely used MQTT client that allows the program to publish and subscribe to MQTT topics. The serial module facilitates communication with an Arduino via a serial port. Lastly, the threading.Thread class is used to run multiple processes (such as MQTT communication and serial reading) concurrently, ensuring smooth execution of all tasks.



**2. MQTT Broker Configuration**

In this section, two MQTT brokers are configured: a local MQTT broker and a cloud-based broker. The local broker, identified by its IP address (192.168.59.157) and port (1883), handles communication within the local network. The cloud broker configuration includes a host address, a secure port (8883), and credentials (username and password) for authentication. By connecting to both brokers, the system ensures that data is accessible locally while also being transmitted to a cloud-based server for remote monitoring.

A computer screen with green text

Description automatically generated.

**3. Global Variables Initialization**

Global variables are defined here to store the current states of various sensors. current\_temperature and current\_humidity hold the latest temperature and humidity readings, respectively. temp\_state and humid\_state store the qualitative state of these values (normal, low, or high). motion\_state keeps track of whether motion has been detected, and distance\_state stores the reading from a distance sensor. previous\_relay\_state is initialized to track the last relay state, preventing unnecessary MQTT messages when the relay state has not changed.

A computer screen shot of text

Description automatically generated

**4. MQTT Topics Definition**

This section defines the MQTT topics that the system uses for publishing and subscribing to messages. For example, the temperature\_topic is set to "greenhouse/temperature", and it will be used for sending or receiving temperature data. Similarly, humidity\_topic, motion\_topic, and distance\_topic are defined for other sensor data. The arduino\_topic handles messages received from the Arduino, while relay\_topic is used to control and monitor the relay's state.

A computer screen shot of text

Description automatically generated

**5. Thresholds for Environmental Parameters**

Threshold values are set to determine acceptable ranges for temperature and humidity. The system defines a minimum temperature (mintemp) of 25°C and a maximum (maxtemp) of 40°C. Similarly, the acceptable humidity range is between 30% (minhumid) and 70% (maxhumid). These thresholds help the system categorize the current environmental conditions as normal, low, or high, influencing relay control decisions.

A computer screen shot of text

Description automatically generated

**6. Serial Communication Setup**

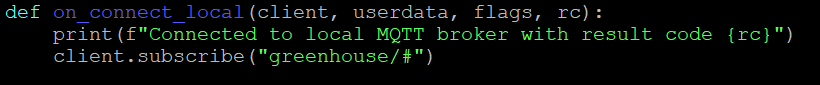
In this section, the program initializes communication with the Arduino using the serial.Serial method. It specifies the serial port (/dev/ttyAMA0) and a baud rate of 9600 for data transmission. A short delay (time.sleep(2)) ensures that the connection stabilizes before data is read from the Arduino. This setup allows the program to receive sensor data directly from the Arduino.



**7. MQTT Callbacks**

**a. Local Broker Connection Callback**

The on\_connect\_local function is a callback triggered when the program connects to the local MQTT broker. It prints a confirmation message and subscribes to all topics under the "greenhouse/#" hierarchy. This ensures that the program listens for all messages related to greenhouse conditions.

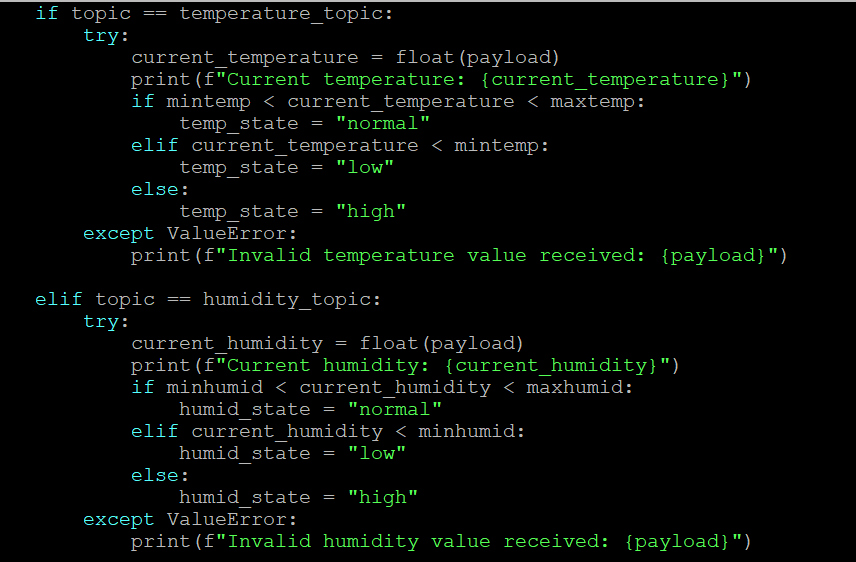


**b. Message Handling Callback for Local Broker**

The on\_message\_local function processes incoming messages from the local broker. It first relays all messages to the cloud broker using client\_cloud.publish(). Then, it evaluates the data based on the message topic. For example, temperature and humidity data are checked against defined thresholds, updating the corresponding state variables (temp\_state, humid\_state). Motion and distance data are similarly handled. Finally, the function decides the relay's state based on a combination of sensor states and publishes the relay state to the MQTT topic only if it changes from the previous state.

A black background with white text

Description automatically generated



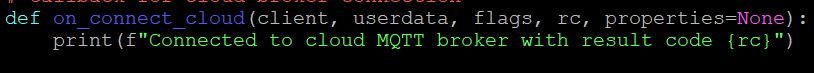
We use some conditions to control our system remotely and easily :

A computer screen with text on it

Description automatically generated

**c. Cloud Broker Connection Callback**

The on\_connect\_cloud function is triggered when the program connects to the cloud MQTT broker. It prints a message confirming the connection. This function ensures that the system is ready to send data to the cloud for remote monitoring.



**8. Serial Communication with Arduino**

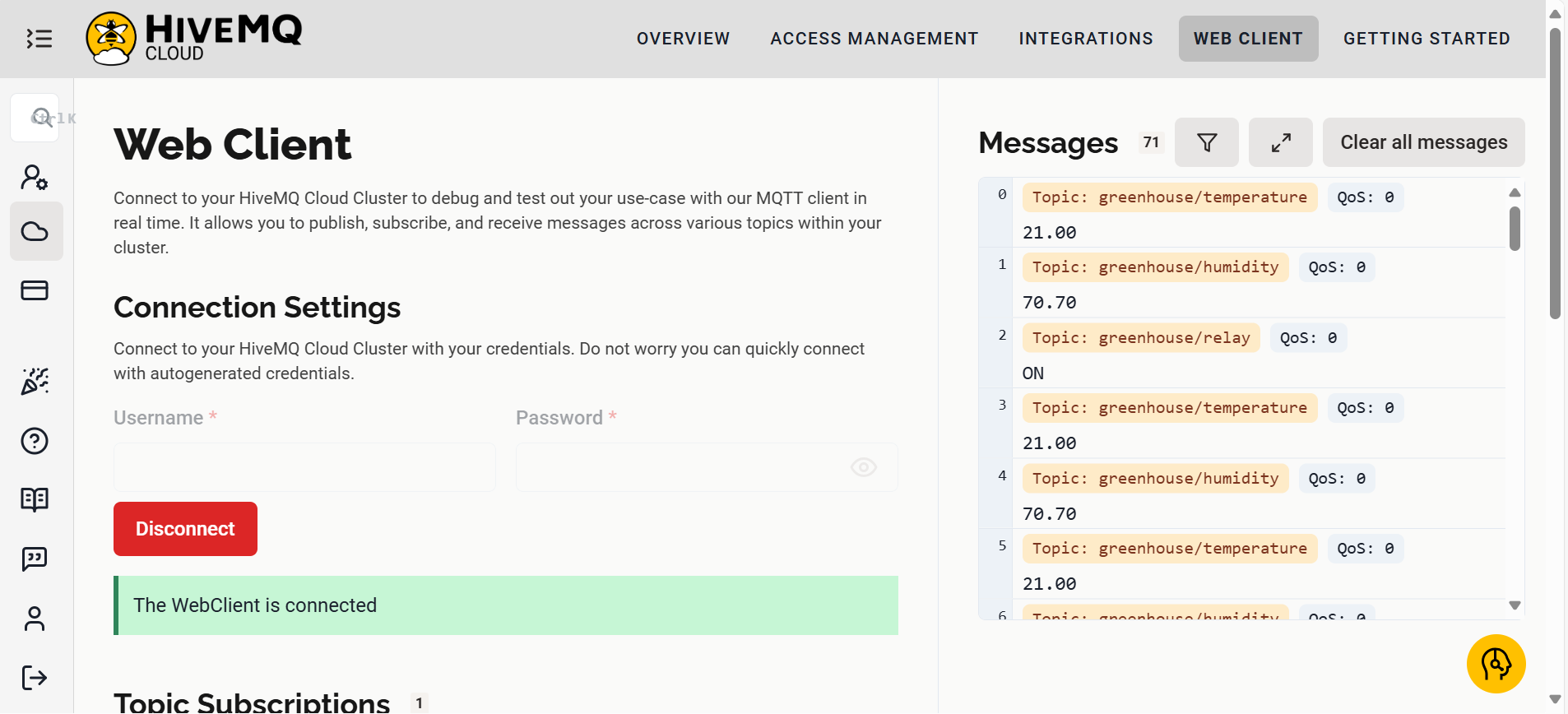
The serial\_communication function continuously reads data from the Arduino using the serial connection. If data is available (ser.in\_waiting > 0), it reads and decodes the message, then publishes it to the cloud MQTT broker under the arduino\_topic. This loop ensures real-time data transmission from the Arduino to the cloud.

A computer screen with text on it

Description automatically generated

**HiveMQ Web Client for Monitoring MQTT Topics:**This figure illustrates the HiveMQ Cloud Web Client interface, which is used to monitor and debug real-time communication within the AgriSmart House system. The Web Client is connected to the HiveMQ MQTT broker, enabling bidirectional communication between IoT devices and the server.

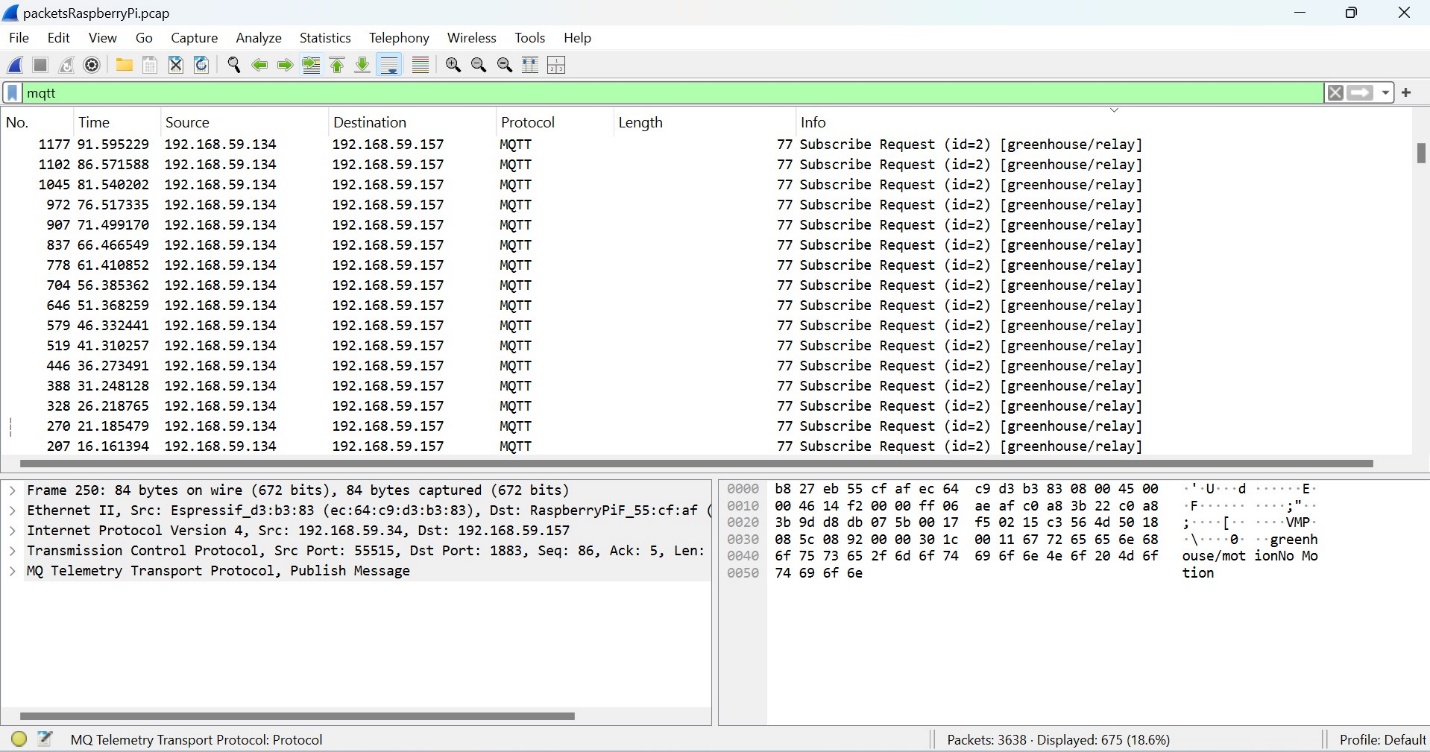
On the right, the **Messages** panel displays the topics and their corresponding payloads published by the system's sensors and actuators.



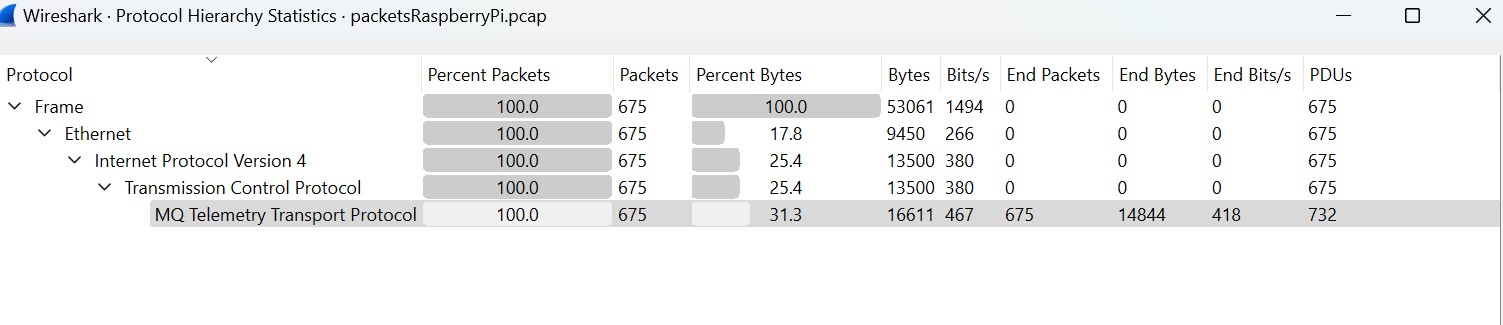
This visualization validates the successful transmission of data from ESP8266 modules and Arduino to the broker. It also allows for the real-time monitoring of key environmental parameters in the greenhouse, ensuring proper system operation. The use of HiveMQ's Web Client simplifies debugging and confirms that the MQTT topics are correctly configured and functioning as intended.

By subscribing to relevant topics, users can observe the system's behavior and verify that sensor data and control commands are being processed correctly. This tool plays a crucial role in ensuring the reliability and efficiency of the AgriSmart House system.

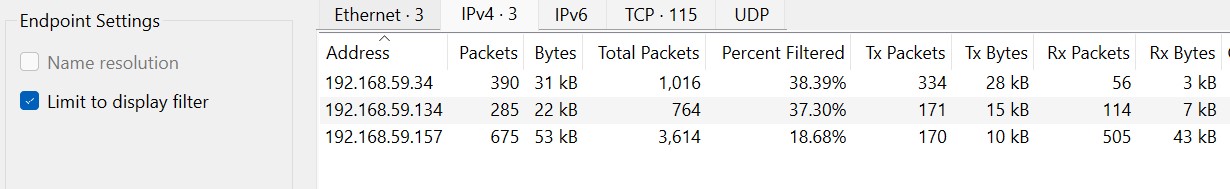
* **Results (the Performance):**  
  After implementing our sensor network, we analyzed various metrics to evaluate its performance and behavior using the Wireshark tool. To achieve this, we examined the traffic generated by the sensors over a 5-minute period, during which the sensor conditions were adjusted to enable actuator operation.



Over a 5-minute period, our brokers recorded a total of 675 packets. These packets consisted of publish and subscribe messages exchanged between the broker and the microcontrollers.



For the MQTT local broker, 170 packets are transmitted, and 505 packets are received. Meanwhile, the first node transmits 334 packets and receives 56 packets. And the second node transmits 171 packets and receives 114 packets. Adding these together gives a total of 675 packets.



To assess the latency in the network, we analyzed the MQTT Publish messages visible in the provided figure These messages pertain to topics such as greenhouse/temperature and greenhouse/humidity. The latency was determined by calculating the difference in timestamps between two consecutive packets, as follows:

* The timestamp for the first packet is **51.375487 m seconds**.
* The timestamp for the subsequent packet is **56.392917 m seconds**.

**Latency Calculation**:  
The latency is obtained by subtracting the earlier timestamp from the later one:

Latency = 56.392917 – 51.375487 = 5.01743 ms

A screenshot of a computer

Description automatically generated

Using the same way to calculate the latency , we can observe this for the ESP connected to distance sensors :

A screenshot of a computer

Description automatically generated

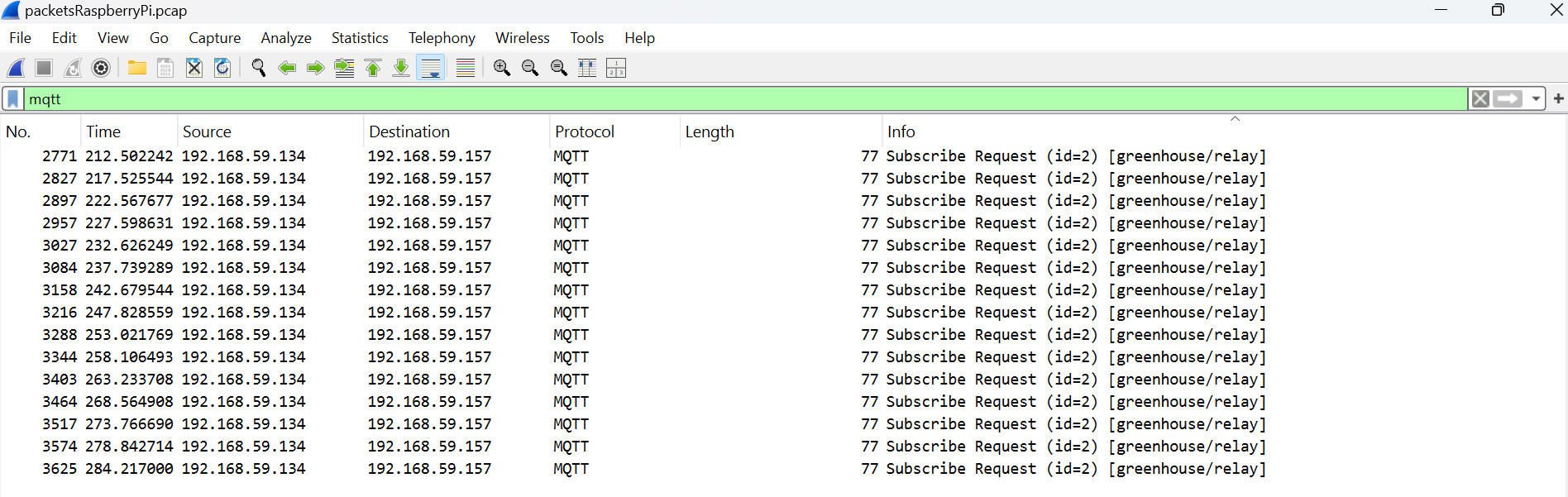
Latency = 34.403516 – 32.440099 = 1.963417 ms

A screenshot of a computer

Description automatically generatedAnd for the same ESP we calculate the latency in publish motion sensor state :

Latency = 32.395831 – 30.373087 = 2.022744 ms

If we analyze the latency in the subscription messages between the Raspberry Pi and the ESP we will also got this :

 latency in the subscription = 284.21700 – 278.842714 = 5.374286ms

To analyze the behavior of the transmitted and received packets over some time, we analyzed the Throughput of the MQTT packets only. over 5 minutes we have a variation in the number of packets.

This graph illustrates the network activity of the AgriSmart House system, focusing on MQTT protocol traffic. The data was captured and analyzed using Wireshark, with MQTT packets filtered for visualization. Peaks in the graph represent moments of heightened communication, such as when environmental data was published to the broker or actuator commands were issued

A screen shot of a graph

Description automatically generated

* **Criticism and Improvements**

**Limitations:**

Dependency on Wi-Fi for ESP8266 communication.:

* The ESP8266 (which is a Wi-Fi-enabled microcontroller) relies on a Wi-Fi connection to communicate with the MQTT broker or other devices. This can be a limitation if the Wi-Fi network is unreliable or unavailable.
* The improvement that can be made is introduce a local buffering system on the ESP8266. If the Wi-Fi connection is lost, the device can temporarily store sensor data and transmit it when connectivity is restored.

Single point of failure in the Raspberry Pi MQTT broker.

* The MQTT broker is running on a Raspberry Pi. If the Raspberry Pi fails or encounters issues, it could disrupt the entire communication system, as it's central to the message flow between devices.
* The improvement will be Deploy a secondary MQTT broker in the cloud or on another device (e.g., another Raspberry Pi or a local server). Then use MQTT's High Availability (HA) feature to synchronize data between the primary and secondary brokers.

**Future Improvements**

Introduce redundant brokers or fog computing.

* + Redundant brokers would involve having multiple MQTT brokers (either on different Raspberry Pis or servers) running simultaneously. This would reduce the risk of a system failure since communication could be routed to another broker if one goes down.
  + Fog computing involves processing data closer to the edge devices (like ESP8266 or Raspberry Pi) rather than relying entirely on centralized cloud or server resources. It could help improve latency and reduce dependency on a single server.

Implement LoRaWAN for larger-scale applications.

* LoRaWAN (Long Range Wide Area Network) is a low-power, long-range wireless protocol used for IoT (Internet of Things) devices. It's suitable for large-scale, wide-area applications where Wi-Fi might not be sufficient. LoRaWAN could help to extend the communication range and provide more reliable data transfer in large-scale systems or remote locations.

**Conclusion :**  
The AgriSmart House project integrates IoT technologies to create an automated agricultural monitoring system. Using ESP8266, Arduino Uno with XBee, and a Raspberry Pi MQTT broker, the system monitors environmental parameters like temperature, humidity, motion, and light, and controls actuators in real time. MQTT ensures efficient data exchange, and the HiveMQ cloud broker enables remote control. This solution demonstrates the potential of IoT in enhancing agricultural productivity, efficiency, and sustainability.