EAST WEST UNIVERSITY

Department of Computer Science and Engineering

Course Title: Internet Of Things

Code: CSE406

Section: 1

LAB-05

Submitted To:

Dr. Raihan Ul Islam

Associate Professor

Department of Computer Science & Engineering

Submitted by

Name	ID
Maisha Rahman	2022-1-60-371
Swarna Rani Dey	2022-1-60-340

Date of Submission

29.08.2025

Lab 5: Comparing HTTP, CoAP, and MQTT Protocols with ESP8266

1. Introduction:

This lab evaluates three IoT communication protocols HTTP (Basic), CoAP, and MQTT implemented on NodeMCU ESP8266 boards with supporting Python services. You will execute each protocol end-to-end, capture traffic in Wireshark, and analyze total packet size, protocol header size, and payload for common request/response cycles. The emphasis is on comparing efficiency, overhead, transport layers (TCP vs. UDP), and practical IoT suitability, with attention to protocol-specific packet structures and their behavior in constrained environments.

2. Objectives:

By completing this lab, I will be able to set up and operate HTTP, CoAP, and MQTT communication systems using ESP8266 microcontrollers with Python support, capture and analyze their network traffic in Wireshark to measure total packet size, header size, and payload size, and interpret key protocol elements such as HTTP request methods and status codes, CoAP message codes, and MQTT control packet types. You will also be able to compare these protocols in terms of efficiency, header overhead, transport layer differences, and suitability for IoT applications, and present your findings in a structured report that includes tables of measured values, annotated Wireshark screenshots, and a discussion of the advantages and disadvantages of each protocol.

3. Materials Required:

Hardware:

- 1–3 × NodeMCU ESP8266 boards.

Software & Utilities:

- Arduino IDE with libraries: ESP8266WiFi, PubSubClient, ArduinoJson, DHTesp, coap-simple.
- Python 3 with aiocoap, flask (install via pip install aiocoap flask).
- Wireshark (Wi-Fi or Ethernet capture).
- MQTT client (e.g., MQTT Explorer); curl or a web browser for HTTP tests.

Network/Broker:

- Wi-Fi network and either HiveMQ Cloud account or a local Mosquitto broker.

Provided Files :

- CSE406_mqtt.ino (MQTT client, sensor publish + device control)
- CSE406_CoapServer.ino (CoAP device-control server)
- CoapClient.py (Python CoAP client)
- CSE406_HTTPbasicClient.ino (HTTP GET client)
- main.py (Flask REST server)

Prerequisites:

- Basic knowledge of Arduino programming and ESP8266 setup.
- Install ESP8266 board support in Arduino IDE (via Boards Manager, URL: https://arduino.esp8266.com/stable/package_esp8266com_index.json).
- Familiarity with Wireshark for capturing and filtering network traffic (e.g., http, udp.port=5683, tcp.port=8883).
- Ensure all ESP8266 boards and scripts use consistent WiFi credentials and server IPs.

Background:

IoT devices typically function under strict limitations in power availability and network bandwidth, making protocol efficiency a critical consideration. HTTP operates as a text-based request/response protocol over TCP and is widely supported, but its verbose headers can introduce significant overhead. CoAP, by contrast, is a lightweight protocol running over UDP that uses compact binary headers while retaining REST-like semantics, making it better suited to constrained devices. MQTT follows a publish/subscribe model over TCP, often secured with TLS, and features extremely small headers that allow it to perform reliably in low-bandwidth or high-latency environments. In this lab, the provided code demonstrates these protocols through common IoT use cases such as device control and telemetry data transmission, enabling you to capture network packets, examine their structure, and evaluate each protocol's suitability for resource-constrained scenarios.

Lab Tasks:

After completing the tasks either individually or in groups, ensure that you document all observations, measurements, and analyses throughout the process. Your final lab report should

include detailed Wireshark screenshots, well-organized tables, and comprehensive explanations of the packet details you encountered during the experiments.

Task 1 — Setup & Packet Capture:

A. Environment Setup

- Install Arduino libs & Python pkgs.
- Set Wi-Fi credentials in `CSE406_mqtt.ino`, `CSE406_CoapServer.ino`, `CSE406_HTTPbasicClient.ino`.
- For MQTT, set `mqtt_server`, `mqtt_username`, `mqtt_password` (HiveMQ Cloud/local).
- Update `CoapClient.py` (ESP8266 IP) and `CSE406_HTTPbasicClient.ino` (Flask server IP).
- Configure Wireshark (Wi-Fi monitor mode if needed, or Ethernet if local).

B. Hardware Setup

- Use NodeMCU ESP8266 boards. No extra hardware is required; comment unused pins or hardcode sensor values (e.g., `humidity=45.0`, `temperature=27`) in `CSE406_mqtt.ino` to stabilize payloads for measurement.

C. Start Captures, Per Protocol

- **HTTP:** Upload `CSE406_HTTPbasicClient.ino`. Run `main.py` (Flask). Filter: `http`.
- **CoAP:** Upload `CSE406_CoapServer.ino`. Run `CoapClient.py` and send `"1"/"0"` via PUT. Filter: `udp.port==5683`.
- **MQTT:** Upload `CSE406_mqtt.ino`. Use MQTT Explorer to publish to `led_state` (“1”/“0”) and subscribe to `esp8266_data`. Filter: `tcp.port==8883`.
- Save each capture as a separate `.pcapng`.

Task 2 — Packet Detail Analysis:

In Wireshark, perform the following for each protocol:

A. Identify the Key Packets:

- **HTTP**: Client **GET** request and server **200 OK** response.
- **CoAP**: **PUT** request (Code **0.03**) and **2.04 Changed** response.
- **MQTT**: **CONNECT/CONNACK**, **SUBSCRIBE/SUBACK**, **PUBLISH** (your data topic).

B. Measure Sizes:

- **Total Packet Size**: From the Wireshark "Length" column (includes IP + TCP/UDP + protocol headers).
- **Protocol Header Size** (extract from the protocol dissection pane):
    - **HTTP**: request/status line + headers (Host, Accept, Content-Length…).
    - **CoAP**: 4-byte fixed header + Options (e.g., Uri-Path, Content-Format).
    - **MQTT**: Fixed header (1–2 B) + Variable header (topic length, packet ID).
- **Payload Size**: Application data (e.g., JSON for HTTP/MQTT; `"0"/"1"` for CoAP).

C. Explain Packet Fields (What to annotate in output):

- **HTTP**: Method (GET), Status (200), key headers (Host, Content-Length), and response payload (e.g., `{"message":"GET request received"}`).
- **CoAP**: Version (1), Type (CON/ACK), Code (0.03 PUT, 2.04 Changed), Options (Uri-Path `"light"`), Token, Payload (`"0"/"1"`).
- **MQTT**: Control packet type (1=CONNECT, 3=PUBLISH), flags (QoS, Retain), Topic length/name, Payload (e.g., JSON in `esp8266_data`).

Task 3 — Protocol Comparison:

Construct a table using one representative transaction per protocol (HTTP GET, CoAP PUT, MQTT PUBLISH), then compute and discuss efficiency, overhead, transport, and IoT suitability. Include annotated Wireshark screenshots for each row.

Protocol	Request/Response	Total on-wire (full)	Total on-wire (headers-only)	Header bytes	Payload bytes	Notes
HTTP	GET /rest → 200 OK	535 B	446 B	338 B	35 B	TCP; text headers (Host,

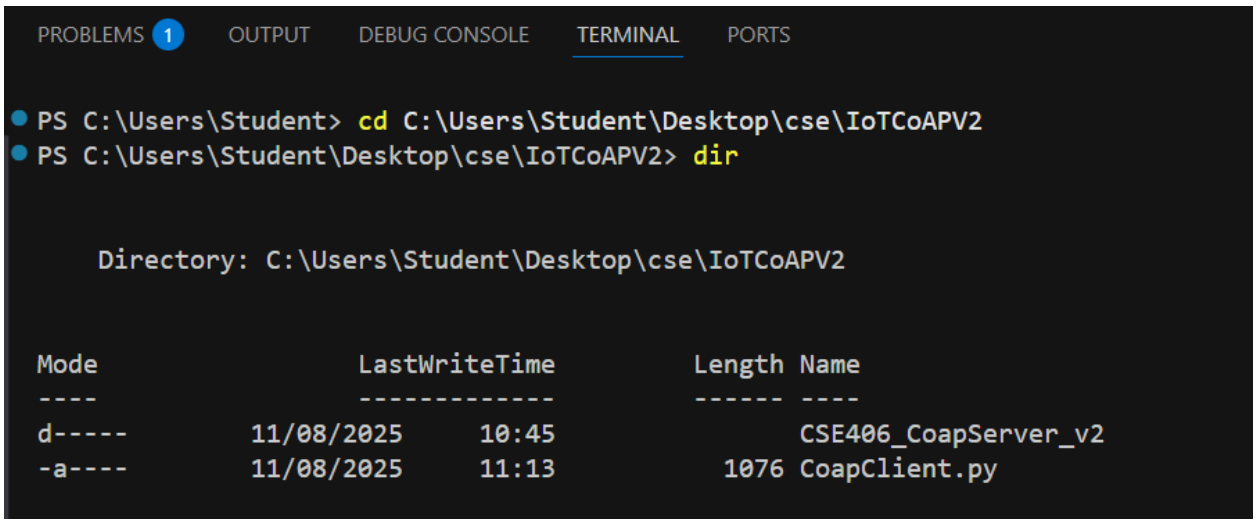
						User-Agent, Content-Length); JSON reply
CoAP	PUT /light “1” → 2.04 Changed	112 B	111B	27 B	1B	UDP; CoAP v1 CON; token; Uri-Path “light”; response empty payload

Conclusion:

This lab clearly showed the differences between HTTP and CoAP on the ESP8266. Capturing packets in Wireshark made the contrast obvious: an HTTP GET/200 OK moved ~535 B to send a 35 B JSON message, with most of the cost in headers, while a CoAP PUT/2.04 Changed achieved the same task with only ~112 B, including ~27 B of protocol overhead. The gap comes from HTTP’s TCP-based, text-heavy design versus CoAP’s compact UDP format. For frequent, lightweight IoT updates, CoAP is more efficient and faster, whereas HTTP remains convenient for integration with existing web tools despite its overhead.

Outputs:

Output 1:



Output 2:

```
PS C:\Users\Student\Desktop\cse\IoTCoAPV2> python -m venv .env
PS C:\Users\Student\Desktop\cse\IoTCoAPV2> .\.env\Scripts\activate
(.env) PS C:\Users\Student\Desktop\cse\IoTCoAPV2> python .\CoapClient.py
Traceback (most recent call last):
  File "C:\Users\Student\Desktop\cse\IoTCoAPV2\CoapClient.py", line 3, in <module>
    import aiocoap
ModuleNotFoundError: No module named 'aiocoap'
(.env) PS C:\Users\Student\Desktop\cse\IoTCoAPV2> pip install aiocoap
Collecting aiocoap
  Using cached aiocoap-0.4.14-py3-none-any.whl.metadata (9.6 kB)
Using cached aiocoap-0.4.14-py3-none-any.whl (241 kB)
Installing collected packages: aiocoap
Successfully installed aiocoap-0.4.14
```

Output 3:

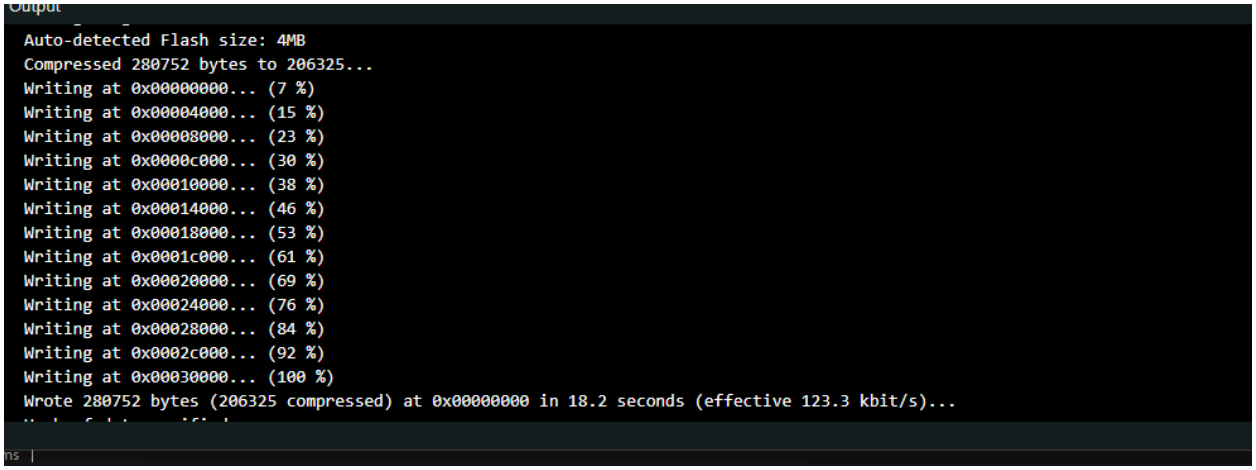
```
    import aiocoap
ModuleNotFoundError: No module named 'aiocoap'
(.env) PS C:\Users\Student\Desktop\cse\IoTCoAPV2> pip install aiocoap
Collecting aiocoap
  Using cached aiocoap-0.4.14-py3-none-any.whl.metadata (9.6 kB)
Using cached aiocoap-0.4.14-py3-none-any.whl (241 kB)
Installing collected packages: aiocoap
Successfully installed aiocoap-0.4.14

[notice] A new release of pip is available: 25.0.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
(.env) PS C:\Users\Student\Desktop\cse\IoTCoAPV2> python .\CoapClient.py
DEBUG:asyncio:Using proactor: IocpProactor
DEBUG:coap:Sending request - Token: 8bdf, Remote: <_Connection at 0x197507202f0 on transport <_ProactorDatagramTransport fd=416 read=<_OverlappedFuture pending cb=
[_ProactorDatagramTransport._loop_reading()]>>, active>
DEBUG:coap:Sending message <aiocoap.Message at 0x197506c7230: CON PUT (MID 42401, token 8bdf) remote <_Connection at 0x197507202f0 on transport <_ProactorDatagramT
ransport fd=416 read=<_OverlappedFuture pending cb=[_ProactorDatagramTransport._loop_reading()]>>, active>, 1 option(s), 1 byte(s) payload>
DEBUG:coap:Exchange added, message ID: 42401.
DEBUG:coap:Incoming error [WinError 1232] The network location cannot be reached. For information about network troubleshooting, see Windows Help from <_Connection
 at 0x197507202f0 on transport <_ProactorDatagramTransport fd=416>, active>
Failed to send PUT request: Network error: NetworkError
(.env) PS C:\Users\Student\Desktop\cse\IoTCoAPV2>
```

Output 4:

```
Using library CoAP simple library at version 1.3.28 in folder: C:\Users\Student\Documents\Arduino\libraries\CoAP_simple_library
"C:\Users\Student\AppData\Local\Arduino15\packages\esp8266\tools\python3\3.7.2-post1/python3" -I "C:\Users\Student\AppData\Local\Arduino15\packages\esp8266\hardware
esptool.py v3.0
Serial port COM6
Connecting.....
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: 34:5f:45:56:17:ba
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 280752 bytes to 206325...
Writing at 0x00000000... (7 %)
```

Output 5:



Github link:
Maisha:
https://github.com/RahMaisha/Comparing-HTTP-CoAP-and-MQTT-Protocols-with-ESP8266.git

Swarna:
https://github.com/SwarnaDey127/Comparing-HTTP-CoAP-and-MQTT-Protocols-with-ESP8266.git