

CoAP Architecture and Components

CoAP Overview

- CoAP (Constrained Application Protocol) is a lightweight protocol designed for resource-constrained devices in IoT.
- Operates over UDP (User Datagram Protocol) and supports asynchronous communication.

Architecture

1. **Core Components:**
 - a. **Endpoint:** Devices or applications participating in communication.
 - b. **Message Layer:** Handles message exchange reliability and duplication.
 - c. **Request/Response Model:** Follows REST principles (GET, POST, PUT, DELETE).
2. **Key Features:**
 - a. **Resource-Oriented Design:** Resources are identified by URIs.
 - b. **Low Overhead:** Optimized for constrained environments.
 - c. **Asynchronous Messaging:** Uses CON (confirmable) and NON (non-confirmable) messages.

Communication Model

- **Client/Server:**
 - Client sends requests (e.g., GET, POST).
 - Server responds with the requested resource or acknowledgment.
- **Proxy and Cache:** Supports intermediate devices for caching and forwarding.

Message Structure

- **Header:**
 - Version, Type, Token, Code, and Message ID.
- **Options:**
 - Additional metadata like URI path and content type.
- **Payload:**

- Actual data transmitted.

CoAP Application

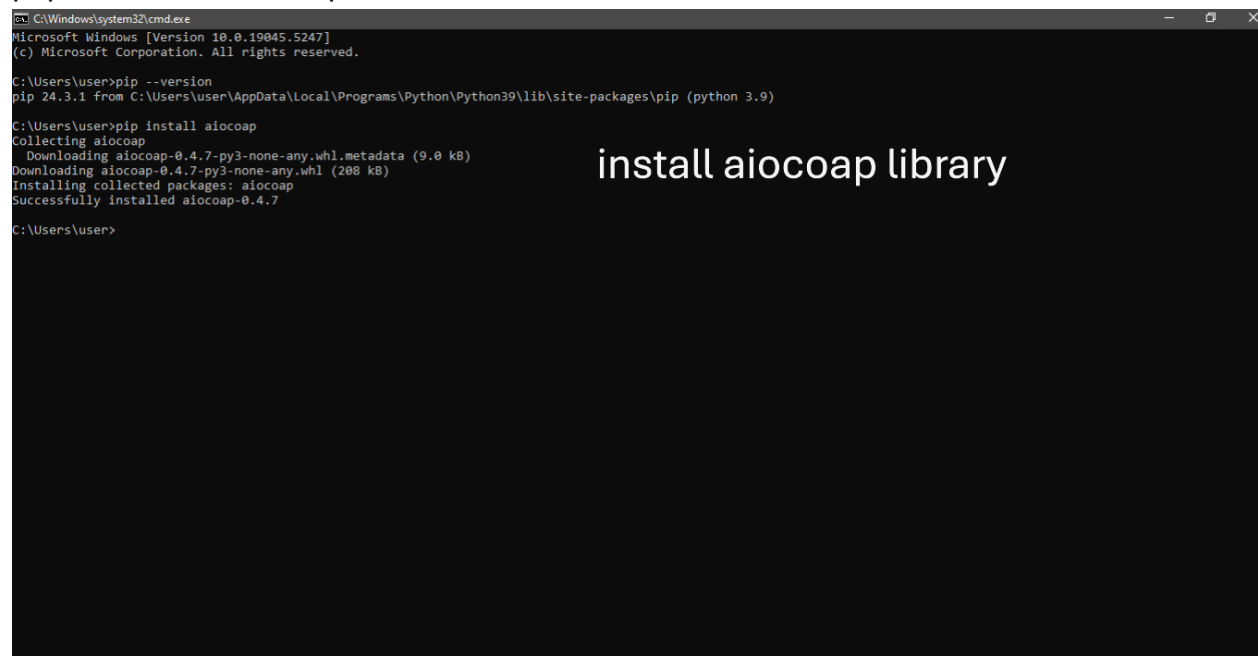
Objective

Build a simple IoT application where a sensor client sends temperature and humidity data to a CoAP server.

To develop the CoAP application, you need to install the `aiocoap` library. This library is a Python implementation of the CoAP protocol. Follow these steps:

1. Open your terminal or command prompt.
2. Run the following command to install the library:

`pip install aiocoap`



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.5247]
(c) Microsoft Corporation. All rights reserved.

C:\Users\user>pip --version
pip 24.3.1 from C:\Users\user\AppData\Local\Programs\Python\Python39\lib\site-packages\pip (python 3.9)

C:\Users\user>pip install aiocoap
Collecting aiocoap
  Downloading aiocoap-0.4.7-py3-none-any.whl.metadata (9.0 kB)
  Downloading aiocoap-0.4.7-py3-none-any.whl (208 kB)
Installing collected packages: aiocoap
Successfully installed aiocoap-0.4.7

C:\Users\user>
```

install aiocoap library

Implementation

Server Code

1. **Use Python with the aiocoap library** to create a CoAP server.
2. **Accepts POST requests** from clients with payloads in the format:

`clientId,temperature,humidity`.

3. **Processes and evaluates the received data** to determine environmental conditions (e.g., good, too hot, too humid).
4. **Responds to clients** with a descriptive message about the received data and its condition status.
5. **Handles GET requests** to provide the last received sensor data or indicate no data is available.
6. **Uses 127.0.0.1 and port 5683** as the server's address and port.
7. **Ensures error handling** for invalid data format or non-numeric values in the payload.
8. **Relies on TEXT_PLAIN content type** for simplicity in communication.

Client Code

1. **Uses Python with the aiocoap library** to create a CoAP client.
2. **Generates unique client identifiers** using the uuid module.
3. **Randomly generates sensor data** for temperature (25.0°C–45.0°C) and humidity (40%–80%) to simulate real-world conditions.
4. **Sends POST requests** to the server with the payload in the format `clientId,temperature,humidity`.
5. **Handles server responses** by decoding and printing messages, including success or error feedback.
6. **Includes error handling** for request timeouts and other exceptions.
7. **Repeats sensor data generation and transmission** every 10 seconds in a loop.
8. **Uses 127.0.0.1 and port 5683** as the target server address and port.

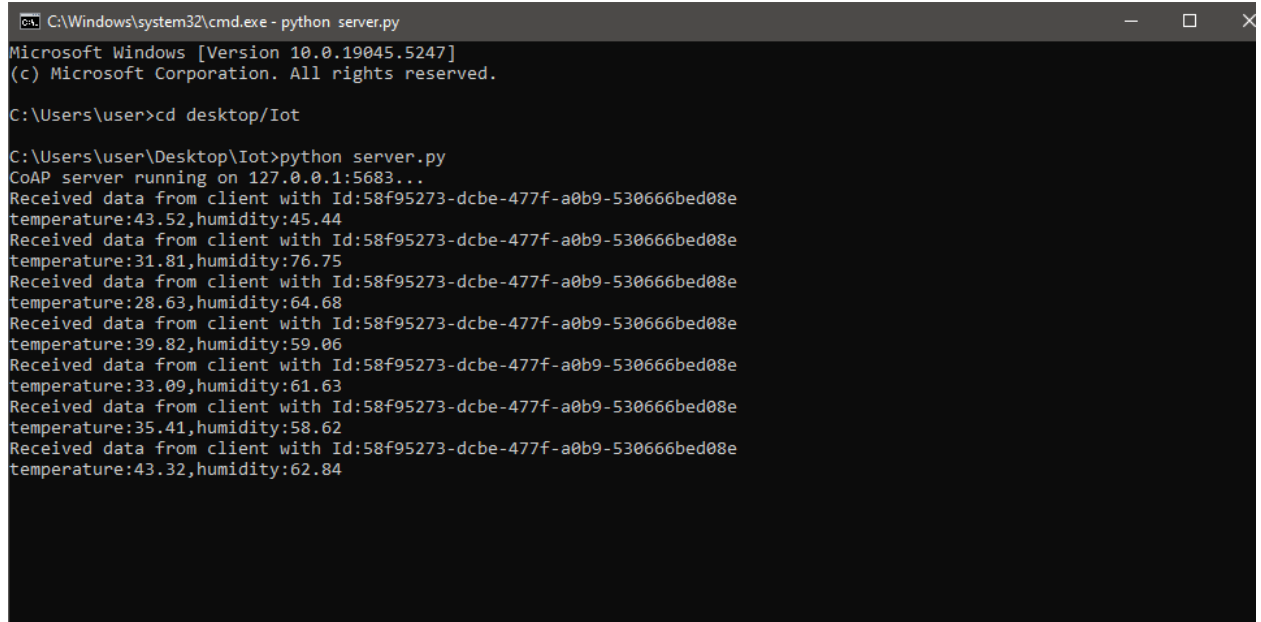
Code Overview

Include the following:

- Explanation of libraries (`aiocoap`, `asyncio`).
- Detailed comments in the code to describe functionality.

Execution Steps

- Start the server on a specific port (e.g., 5683).
- Run the client to send data periodically.
- Observe the server's response.



```
C:\Windows\system32\cmd.exe - python server.py
Microsoft Windows [Version 10.0.19045.5247]
(c) Microsoft Corporation. All rights reserved.

C:\Users\user>cd desktop/Iot

C:\Users\user\Desktop\Iot>python server.py
CoAP server running on 127.0.0.1:5683...
Received data from client with Id:58f95273-dcbe-477f-a0b9-530666bed08e
temperature:43.52,humidity:45.44
Received data from client with Id:58f95273-dcbe-477f-a0b9-530666bed08e
temperature:31.81,humidity:76.75
Received data from client with Id:58f95273-dcbe-477f-a0b9-530666bed08e
temperature:28.63,humidity:64.68
Received data from client with Id:58f95273-dcbe-477f-a0b9-530666bed08e
temperature:39.82,humidity:59.06
Received data from client with Id:58f95273-dcbe-477f-a0b9-530666bed08e
temperature:33.09,humidity:61.63
Received data from client with Id:58f95273-dcbe-477f-a0b9-530666bed08e
temperature:35.41,humidity:58.62
Received data from client with Id:58f95273-dcbe-477f-a0b9-530666bed08e
temperature:43.32,humidity:62.84
```

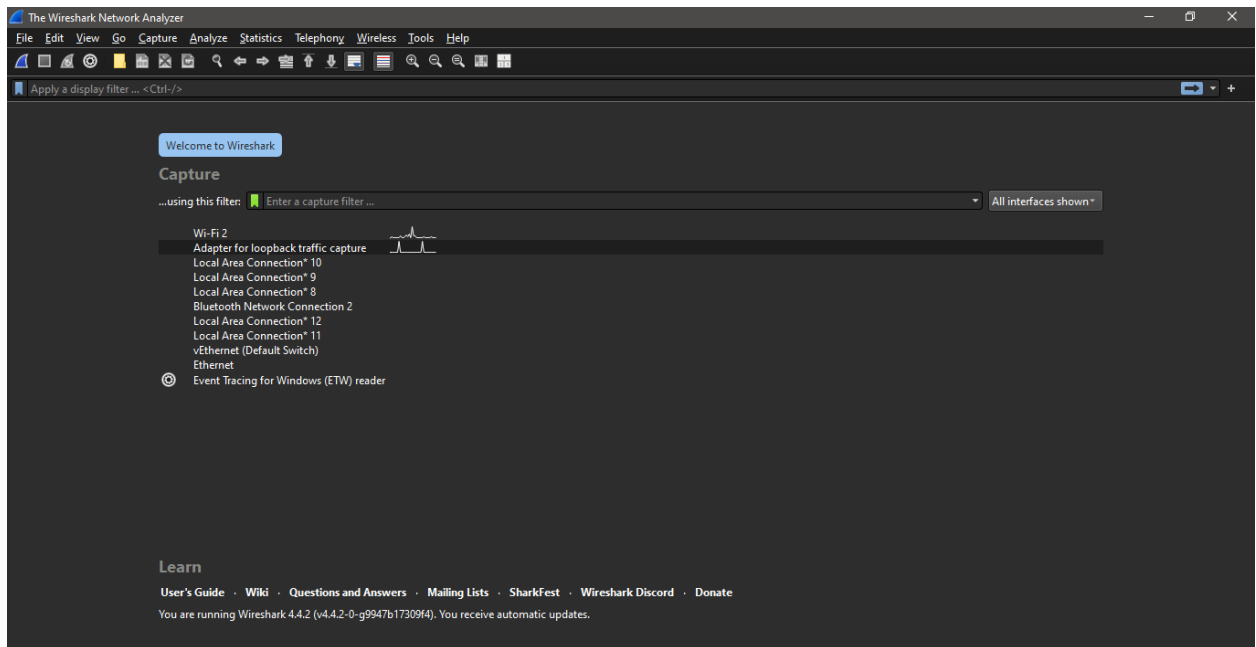
```
C:\Windows\system32\cmd.exe - python client.py

C:\Users\user>cd desktop/Iot

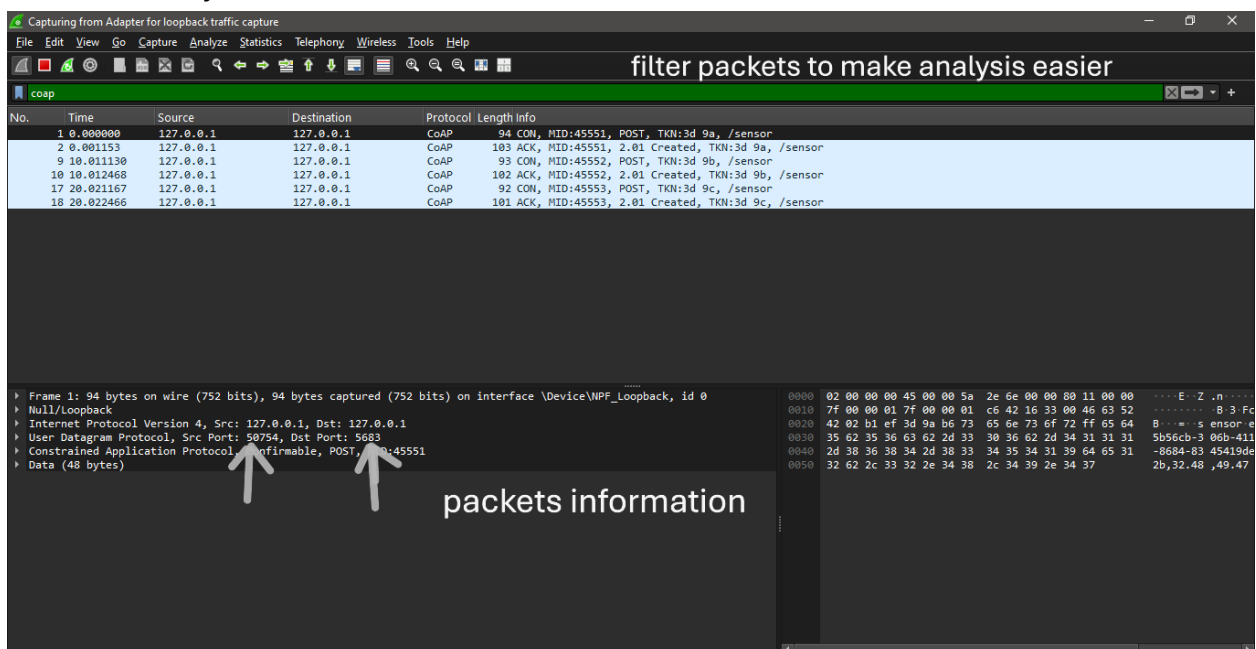
C:\Users\user\Desktop\Iot>python client.py
Client ID: 58f95273-dcbe-477f-a0b9-530666bed08e ,Generated Data: Temperature = 43.52°C, Humidity = 45.44%
Response from server: Received Temperature: 43.52°C, Humidity: 45.44%. It is too hot.
Client ID: 58f95273-dcbe-477f-a0b9-530666bed08e ,Generated Data: Temperature = 31.81°C, Humidity = 76.75%
Response from server: Received Temperature: 31.81°C, Humidity: 76.75%. It is too hot.
Client ID: 58f95273-dcbe-477f-a0b9-530666bed08e ,Generated Data: Temperature = 28.63°C, Humidity = 64.68%
Response from server: Received Temperature: 28.63°C, Humidity: 64.68%. It is too humid.
Client ID: 58f95273-dcbe-477f-a0b9-530666bed08e ,Generated Data: Temperature = 39.82°C, Humidity = 59.06%
Request timed out.
Client ID: 58f95273-dcbe-477f-a0b9-530666bed08e ,Generated Data: Temperature = 33.09°C, Humidity = 61.63%
Response from server: Received Temperature: 33.09°C, Humidity: 61.63%. It is too hot.
Client ID: 58f95273-dcbe-477f-a0b9-530666bed08e ,Generated Data: Temperature = 35.41°C, Humidity = 58.62%
Response from server: Received Temperature: 35.41°C, Humidity: 58.62%. It is too hot.
Client ID: 58f95273-dcbe-477f-a0b9-530666bed08e ,Generated Data: Temperature = 43.32°C, Humidity = 62.84%
Response from server: Received Temperature: 43.32°C, Humidity: 62.84%. It is too hot.
```

Capturing Packets Using Wireshark

1. **Install Wireshark:** Ensure Wireshark is installed on your system to monitor network traffic.
2. **Start Packet Capture:** Launch Wireshark, start capturing packets, and select the appropriate network adapter for loopback traffic (e.g., lo for localhost) to monitor communication between the client and server.



3. **Apply a Filter:** Use the filter `coap` in Wireshark to display only CoAP-related packets for easier analysis.



Here, we apply a CoAP filter to display only relevant packets. The source port matches the server's configured port (5683), while the destination port varies for the client, indicating successful communication between the client and server.

Advantages and Disadvantages of CoAP

Advantages

1. **Lightweight:**
 - a. Designed for low-bandwidth and resource-constrained environments.
2. **RESTful Architecture:**
 - a. Easy integration with web services.
3. **Asynchronous Communication:**
 - a. Supports real-time updates.
4. **Efficient Message Encoding:**
 - a. Small header size reduces overhead.
5. **Support for Discovery:**
 - a. Resources can be discovered using multicast.

Disadvantages

1. **Reliability:**
 - a. UDP-based; no inherent guarantee of message delivery.
2. **Limited Security:**
 - a. DTLS (Datagram TLS) provides security, but it adds complexity.
3. **Scalability:**
 - a. Less suitable for large-scale applications compared to HTTP/2 or MQTT.
4. **Complexity in Implementation:**
 - a. Requires careful handling of message retransmission and duplication.