PW 5 : CoAP Lab Exercise Using aiocoap on Localhost

Dr. Mohamed Amine Ferrag
Guelma University

10/04/2025

Objective

- Understand the basics of the Constrained Application Protocol (CoAP).
- Set up and run a CoAP server locally using `aiocoap`.
- Use a CoAP client script to demonstrate GET, POST, PUT, and DELETE operations.
- Verify and analyze the CoAP message exchanges between the client and the server.

Lab Requirements

- A computer running Linux, macOS, or Windows.
- Python 3 installed.
- `aiocoap` library installed in a virtual environment.
- Terminal or command prompt access.

Part 1: Setting Up the Environment

1. **Create a Virtual Environment:**
   Open a terminal, navigate to your project directory, and execute:

```
python3 -m venv venv
```

Listing 1: Create Virtual Environment

2. **Activate the Virtual Environment:**
   On macOS/Linux, run:

```
source venv/bin/activate
```

Listing 2: Activate Virtual Environment

3. **Install aiocoap:**
   With the virtual environment activated, install the package:

```
pip install aiocoap
```

Listing 3: Install aiocoap

Part 2: Running the CoAP Server

1. Prepare your CoAP server using the solution code provided in Section **??** below.
2. Open a terminal, ensure the virtual environment is activated, and run your server with:

```
python3 coap_server.py
```

Listing 4: Start CoAP Server

3. The server should start and display the available CoAP endpoints:
   - `GET coap://127.0.0.1/sensors/temp`
   - `POST coap://127.0.0.1/alarms`
   - `PUT coap://127.0.0.1/configuration/light1`
   - `DELETE coap://127.0.0.1/sensors/obsolete_sensor`

Part 3: Using the CoAP Client

Step 3.1: Testing the GET Operation

1. Open a new terminal (with the virtual environment activated).
2. Run the client to perform a GET request to the temperature sensor endpoint:

```
python3 coap_client.py GET
```

Listing 5: GET Request

3. Verify that the response contains the expected sensor data.

Step 3.2: Testing the POST Operation

1. Open another terminal, and run:

```
python3 coap_client.py POST
```

Listing 6: POST Request

2. Check that the response indicates the successful creation of a new subordinate resource.

Step 3.3: Testing the PUT Operation

1. Run the following command to test updating or creating a configuration:

```
python3 coap_client.py PUT
```

Listing 7: PUT Request

2. The client should receive a response confirming the update.

Step 3.4: Testing the DELETE Operation

1. Run the client with the DELETE command to remove a resource:

```
python3 coap_client.py DELETE
```

Listing 8: DELETE Request

2. Verify that the resource is marked as deleted by performing a subsequent GET request.

Part 4: Additional Testing and Options

- Experiment with different payloads and observe the response codes.
- Test error conditions by sending malformed requests.
- Discuss security considerations and constrained network characteristics in CoAP.

Troubleshooting Tips

- **Server Not Running:** Ensure the CoAP server is up and running in a separate terminal.
- **Incorrect URIs:** Double-check that the URIs in the client match those registered in the server.
- **Environment Issues:** Ensure you are working within the virtual environment and that there are no network restrictions.

Conclusion

This lab exercise introduces students to the CoAP protocol by:

- Setting up a local CoAP server using `aiocoap`.
- Running a client to execute GET, POST, PUT, and DELETE requests.
- Troubleshooting common issues in IoT communication scenarios.

This hands-on lab provides a practical foundation for working with CoAP in resource-constrained environments, similar to real-world IoT applications.