Checkpoint 3: CoAP

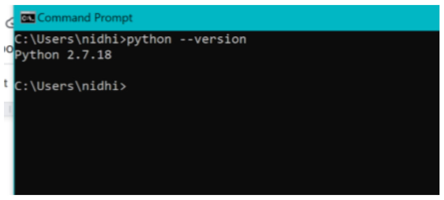
Constrained Application Protocol is based on the client-server architecture. It provides methods such as GET, PUT, POST, DELETE to access and modify resources on the server using a URL. The two important CoAP methods used in IoT devices are OBSERVE and DISCOVER. The aim of this checkpoint is to give students hands-on experience in message exchanges between IoT devices using CoAP.

Students are free to choose any programming language and CoAP libraries to complete the checkpoint. However, use the following setup guide if you are using Python with the CoAPthon library.

Part 1: Environment set up.

Tools needed:

1. **Python 2.7**
   Download and install Python 2.7 if you do not have it on your computer. To check type "python --version" in the command line. [1]
   successful installation returns the version of python that's currently running like in the image below;

   ```
   C:\Users\nidhi>python --version
   Python 2.7.18

   C:\Users\nidhi>
   ```

   In the scenarios where you may need to have both the python versions follow [2] and [3] to successfully download and install both 2.7 and 3.X versions of python in your system.

2. **Pycharm** (not necessary, any other IDE will work as long as you specify the python interpreter correctly)
   To download and install pycharm follow [5]

3. **CoAPthon**
   Using the following command in the command line install the library in the python's libraries:

   ```
   pip install CoAPthon
   ```

   Note: Downloading a library in a linux based machine by default requires running the command as follows:

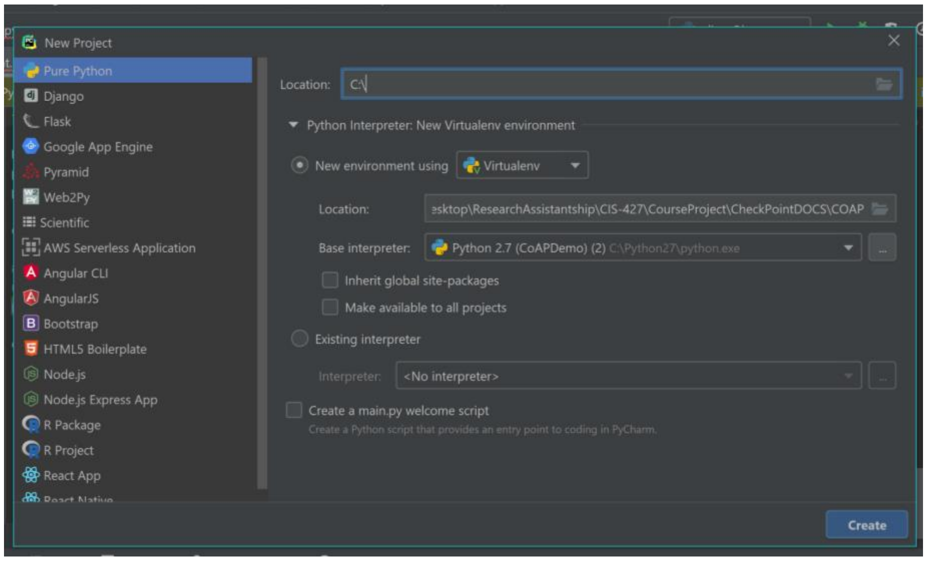
pip2 install CoAPthon

Testing the setup:

For creating both client and server follow:

Open pycharm -> new project -> choose a location for creating the project ->choose your base interpreter and create the project.

Choose the right “Base interpreter”, depending upon the version of python in your system, the list would differ. For coapthon choose “Python 2.7”. Refer to the image below.



Location: C:\

Python Interpreter: New Virtualenv environment

New environment using Virtualenv

Location: esktop\ResearchAssistantship\CIS-427\CourseProject\CheckPointDOCS\COAP

Base interpreter: Python 2.7 (CoAPDemo) (2) C:\Python27\python.exe

Inherit global site-packages

Make available to all projects

Existing interpreter

Interpreter: <No interpreter>

Create a main.py welcome script

Create a Python script that provides an entry point to coding in PyCharm.

Once you have created the project, test if everything is set up correctly.

Build the server with the following lines of code:
Enter your IP address instead of “0.0.0.0”

```
from coapthon.server.coap import CoAP
from exampleresources import BasicResource

class CoAPServer(CoAP):
    def __init__(self, host, port):
        CoAP.__init__(self, (host, port))
        self.add_resource('basic/', BasicResource())

def main():
    server = CoAPServer("0.0.0.0", 5683)
    try:
        server.listen(10)
    except KeyboardInterrupt:
        print "Server Shutdown"
        server.close()
        print "Exiting..."

if __name__ == '__main__':
    main()
```

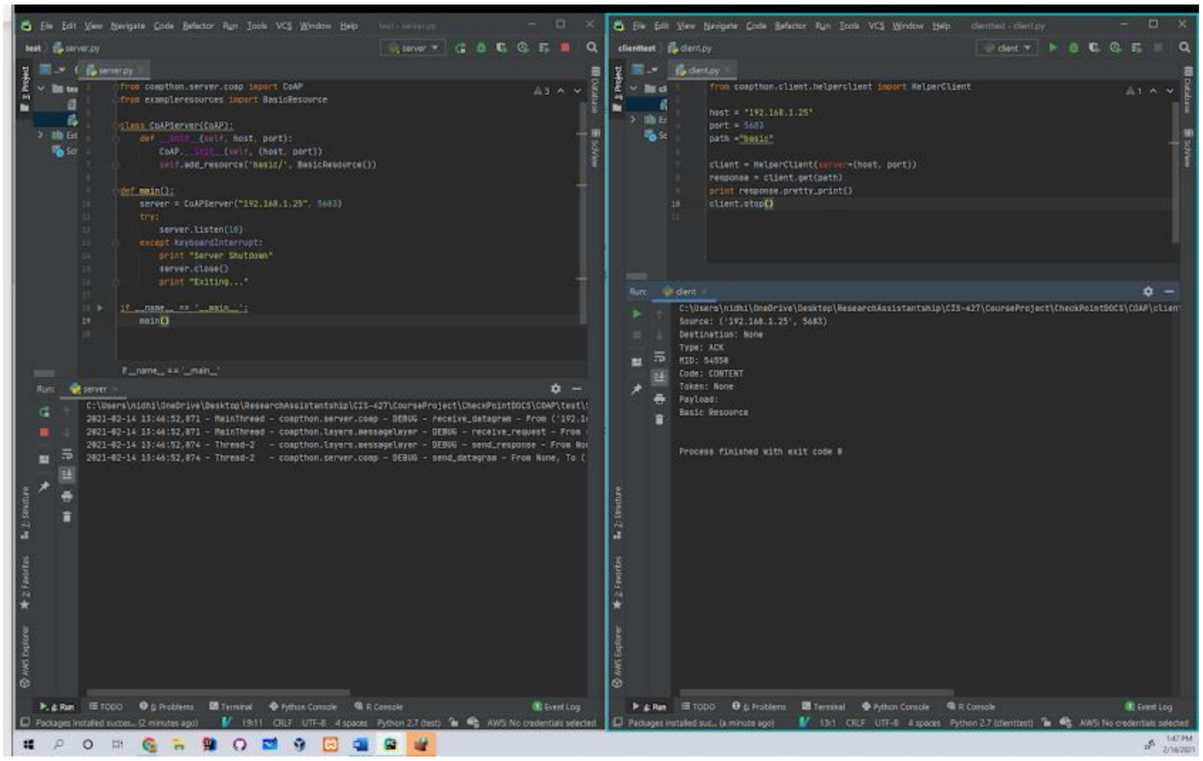
In a separate project build your client using the code:
Set the host variable to the IP address of the machine where the server is located.

```
from coapthon.client.helperclient import HelperClient

host = "127.0.0.1"
port = 5683
path ="basic"

client = HelperClient(server=(host, port))
response = client.get(path)
print response.pretty_print()
client.stop()
```

Now compile and run the server and then the client. Successful message exchange would display the payload on the client’s console and details of the exchange on the server side. Refer to the image below.



```python
from coapthon.server.coap import CoAP
from exampleresources import BasicResource

class CoAPServer(CoAP):
    def __init__(self, host, port):
        CoAP.__init__(self, (host, port))
        self.add_resource('basic/', BasicResource())

def main():
    server = CoAPServer("192.168.1.25", 5683)
    try:
        server.listen(10)
    except KeyboardInterrupt:
        print "Server Shutdown"
        server.close()
        print "Exiting..."

if __name__ == '__main__':
    main()
```

```
C:\Users\nidhi\OneDrive\Desktop\ResearchAssistantship\CIS-427\CourseProject\CheckPointDOCS\COAP\test\!
2021-02-14 13:46:52,871 - MainThread - coapthon.server.coap - DEBUG - receive_datagram - From ('192.1
2021-02-14 13:46:52,871 - MainThread - coapthon.layers.messagelayer - DEBUG - receive_request - From (
2021-02-14 13:46:52,874 - Thread-2   - coapthon.layers.messagelayer - DEBUG - send_response - From No
2021-02-14 13:46:52,874 - Thread-2   - coapthon.server.coap - DEBUG - send_datagram - From None, To (
```

```python
from coapthon.client.helperclient import HelperClient

host = "192.168.1.25"
port = 5683
path ="basic"

client = HelperClient(server=(host, port))
response = client.get(path)
print response.pretty_print()
client.stop()
```

```
C:\Users\nidhi\OneDrive\Desktop\ResearchAssistantship\CIS-427\CourseProject\CheckPointDOCS\COAP\clien
Source: ('192.168.1.25', 5683)
Destination: None
Type: ACK
MID: 54558
Code: CONTENT
Token: None
Payload:
Basic Resource


Process finished with exit code 0
```

Part 2:

Setup the server on the host machine and setup the clients on the virtual machine. You can implement the clients on the virtual machine by using the Thonny IDE. PyCharm may make the machine slower.

Step 1: Add the following resources on the server:

a. **temp**: stores the current numeric temperature of the said device.
b. **weather**: provides a descriptive weather condition based on temperature readings.

Step 2: The server should monitor the change in ``temp” and update the ``weather”. The weather has three status, cold, normal, and hot, which represents a temperature of (-inf, 0], (0-25], (25, +inf).

Step 3: Setup the following clients (*they are programs or processes, not devices)*:

a. **User1**: use the "observe'' method on the resource "weather".
b. **Thermostat**: use the "put" method to update the resource "temp" with a given number.
c. **Smart Home Control Unit**: "observe" both resources.
d. **Guest**: use the Resource Discovery feature to find the resources available; use the "get" method on the resource "weather".

Each time the thermostat client is executed, it takes in a numeric parameter as simulated temperature and updates the resource ``temp" on the server.

Step 4: Test the clients by changing the temperature multiple times (preferably 3 to 6 times). Observe the control packets in the communication between each client and the server. Answer the following:

1. Type of message - confirmable or non confirmable
2. Message ID in the request and the response packet
3. Payload size in the request and the response packet
4. Will the ``Guest'' client get persistent updates of the resource "temp''? Why / why not?

References:
Python 2.7 download
[1] https://www.python.org/download/releases/2.7/

Managing multiple versions of python
[2] https://spapas.github.io/2017/12/20/python-2-3-windows/
[3]https://www.freecodecamp.org/news/installing-multiple-python-versions-on-windows-using-virtualenv/

In case you have python 2.7 and pip installed but the command line still doesn't recognise the command "pip", you may have to check that the required variables are defined in the PATH environment variable. Follow the answer in this stack overflow question.
[4]https://stackoverflow.com/questions/23708898/pip-is-not-recognized-as-an-internal-or-external-command

Pycharm:
[5] https://www.jetbrains.com/pycharm/

https://tools.ietf.org/html/rfc7252#page-31
[6] https://www.opensourceforu.com/2016/09/coap-get-started-with-iot-protocols/

Coapthon (With python 2.7)

[7] https://github.com/Tanganelli/CoAPthon

Coapthon documentation
[8] https://coapthon.readthedocs.io/en/latest/