**Intro**

For Project 2, I used a Raspberry PI along with a temperature sensor and accessories to create a Python implementation of a CoAP server. My Raspberry PI 3B+ runs Raspberry PI OS (previously called Raspbian), and my project code was written in Python 3. I connected my Raspberry PI to a solderless breadboard using a 26-pin ribbon cable and T-Cobbler breakout, used to ease GPIO pin access. For my sensor, I used a DS18B20 temperature probe. For my CoAP client, I used the Copper for Chrome browser extension/app. Finally, I used Wireshark to analyze packs sent between the server and client.

**Design and Implementation**

A close up of a device

Description automatically generated

(Circuit diagram, image also included separately in project folder)

My project design centers around reading a temperature sensor and communicating those readings from server to client using the CoAP protocol. My code (.py files included in project folder) consists mostly of the CoAPthon3 library with two custom scripts added. One script (project2.py) extends the server class, starting up a server on port 5683 with a custom resource imported from my second custom script. This script is quite simple and does not need any real UI since that will be handled on the client end. The resource imported into project2.py comes from project2resource.py which is an extension of the resource class. Project2resource.py contains some standard libraries for string formatting, some setups to access the temperature probe, and an initialization and GET function. The initialization sets up the resource and has a default payload that should never actually get sent saying “no temperature readings yet.” When the CoAP client calls the GET function, the script reads the current temperature, duplicates the standard Celsius reading, and converts the duplicate to Fahrenheit. Finally, it takes the Celsius and Fahrenheit readings, formats the strings to shorten the decimals since this is not a scientific application and readability is more important, and assigns these two values to the payload before sending them to the client. The resource is discoverable by the client and shows up as “Office Temperature” since the sensor is in my home office. Payloads have the format “(24.29, 78.72).”

For step 3 of the project, I used Wireshark to analyze packets sent between my CoAP server and client. I captured the packets on my desktop computer that was running the Copper for Chrome CoAP client. Searching the Wireshark capture for “coap” returns the following sample of frames:

Table

Description automatically generated

The first frame in the list, frame 1192, represents the client discovering the server. 192.168.1.73 is the IP of my desktop computer running the CoAP client while 192.168.1.135 is the IP of my Raspberry Pi running the CoAP server. Additional details such as the MAC addresses, port number, and protocol information can be seen below:

Graphical user interface, text, application

Description automatically generated

A subsequent GET can also be seen below in frame 1602. The resource has been discovered at this point and is now labelled “office\_temperature.” Here the source is 192.168.1.73 (client) and the destination is 192.168.1.135 (server).

Graphical user interface, text, application

Description automatically generated

Finally, frame 1800 shows an example of a delivered payload. Here the source is 192.168.1.135 (server) and the destination is 192.168.1.73 (client). The payload data is highlighted in the screenshot, showing that the temperature in my office was 28.15 degrees Celsius.

Graphical user interface, text, application

Description automatically generated

My project mainly uses one Python 3 library called CoAPthon3 (<https://github.com/Tanganelli/CoAPthon3>). As this is a port of the original CoAPthon library, I also used the documentation section for the original CoAPthon (<https://github.com/Tanganelli/CoAPthon>) library which was much more thorough. Other than CoAPthon, I used some standard Python libraries to access features of the Raspberry Pi, temperature sensor, and string formatting functions. These other libraries include time, os, glob, and math.

**Difficulties and Lessons Learned**

The greatest challenge of this project was getting the GET function of the CoAP resource to read my sensor data. I kept running into an issue where my script would say that the functions called from GET were not defined. I had originally written two functions to pull and format my sensor data, and these functions works perfectly in a test script I made to test my temperature sensor. When I added these functions to my TemperatureResource class in project2resources.py, the above error occurred each time I called them even though I had all the same dependencies added from my test script. To resolve this issue, I copied the code from the two functions into the GET function so that the GET function was running the code itself rather than calling other functions. I originally tried to attend office hours to look more into this but managed the solve the issue myself using the above workaround. I am not completely sure why this worked or why the problem occurred in the first place, but it resolved my issue.

The other challenge I had was learning how to use a temperature sensor. Since I received a faulty temperature sensor for project 1, I ordered a new, nicer sensor for this second project. The DS18B20 temperature probe, with an included pull up resistor board, was very easy to use. I was able to get readings with very little effort and uncomplicated code. It did take some troubleshooting to get the 1 wire functionality working on my Raspberry Pi, but overall, it was a great experience and I wish I had this sensor for project 1.

As I mentioned in my project 1 report, I have rarely used Python and enjoyed writing code in a language with which I am less practiced. If I were to extend this project, I would implement the other REST API calls not outlined in this project such as POST, PUT, DELETE. It would be interesting to implement POST/PUT to control resources remotely on the Pi as well.