Smart Light System

CSCI 43300 Internet of Things

Evin Bour-Gilson[1] & Noran Abdel-Aziz[2]

April 25, 2024

[1] ebourgil@iu.edu | 2000672545
[2] nabdelaz@iu.edu | 2000784160

I. Introduction

This project delved into creating a smart light system using a Raspberry Pi. By integrating various sensors such as an IR receiver, RGB LED, PIR motion sensor, a WS2812B light strip, and a 28BYJ-48 stepper motor, the goal is to create an IoT application capable of intelligent lighting control. The lighting control is implemented by a reprogrammed infrared remote and PUT requests over the established CoAP server running on the Raspberry Pi.

II. Development and Architecture Design

The project utilizes several key sensors/actuators to make this work:

- **IR Receiver:** Receives infrared data transmissions from a remote control for user input.
- **RGB LED:** Provides visual feedback through different color indicators for user interactions.
- **HC-SR501 PIR Motion Sensor:** Detects motion in the room to enable automatic light control, in turn enhancing energy efficiency.
- **WS2812B Light Strip:** Enables dynamic and customizable lighting effects, such as gradients, blinking, and brightness modification of the light.
- **28BYJ-48 Stepper Motor:** Physically turns on and off the light switch in the room corresponding with the WS2812B light strip

Moreover, we used these libraries in our code:

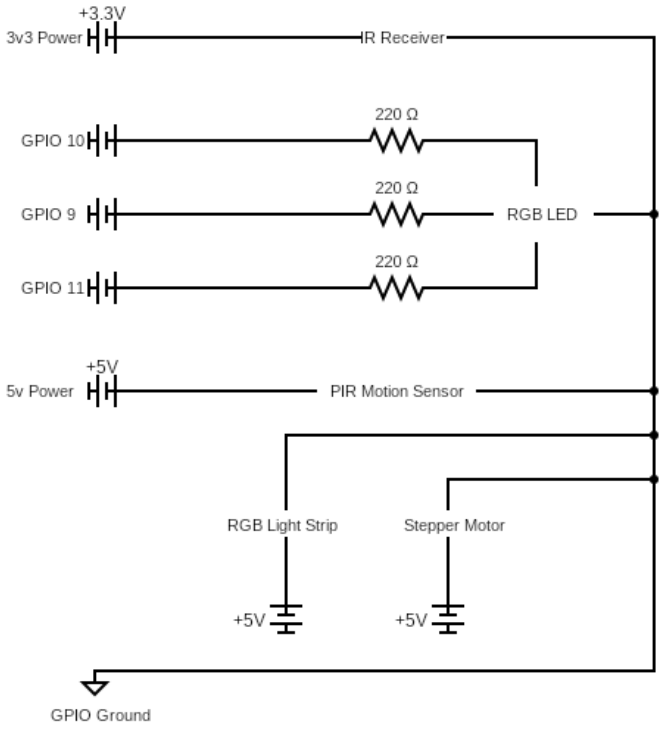
- **aiocoap:** This library was used for creating CoAP resources and handling CoAP requests and responses. This was helpful with allowing remote control of the smart light system since it made it easier for devices in the same network to work together.
- **asyncio:** This was used for asynchronous programming and to handle simultaneous tasks.
- **RPi.GPIO:** This library provided access to the GPIO pins of the Raspberry Pi, allowing control over the sensors that we used.
- **gpiozero:** We utilized this library in order to have an easier interface to work with for the PIR motion sensor
- **datetime:** This was used to handle date/time-related operations in the program (in our case, we kept track of the amount of time it took for a value to change and stored it).
- **board and neopixel:** These libraries are used to control the LED strip connected to the Raspberry Pi. This was essential in our project because it allowed us to change the visual aspects of the LED, in particular the brightness and animation effects.
- **colorsys:** This helped with converting between different color representations. In our case, we used it to convert between HSV to RGB.

GPIO Pins Used:

- **IR Receiver:** 3v3 Power (1), GPIO 17 (11), Ground (9)

- **RGB LED:** GPIO 10 (19), GPIO 9 (21), GPIO 11 (23), Ground (9)
- **HC-SR501 PIR Motion Sensor:** GPIO 4 (7), 5v Power (2), Ground (9)
- **WS2812B Light Strip:** GPIO 21 (40), Ground (14)
- **28BYJ-48 Stepper Motor:** GPIO 26 (37), GPIO 19 (35), GPIO 13 (33), GPIO 6 (31), Ground (9)

Circuit Diagram:



III. Implementation & Analysis

The program can be roughly divided into 5 parts:

1) GPIO & Sensor Setup (lines 12-88):
    a) This segment initializes the GPIO pins of the Raspberry Pi.
    b) Each GPIO pin used is configured to interact with all the sensors (IR receiver, PIR motion sensor, RGB LED, WS2812B LED strip, and 28BYJ-48 stepper motor).
    c) By setting up the pins, the program establishes the physical interface between the Raspberry Pi and the external components, which is vital for it to work.
    d) Proper setup of the sensors ensures that the Raspberry Pi can accurately receive the input signals and control the output devices.

2) Remote Control Handling (lines 127-179):

a) This segment defines how the script interacts with signals received from an IR remote control. Specifically, it allows users to control the smart light using an infrared remote that has been reprogrammed to execute the device's functions.
b) It maintains a list of button codes corresponding to specific actions, such as changing colors, adjusting brightness, or toggling power.
c) Upon receiving an IR signal, the script compares it to the predefined button codes and executes the respective action.
d) The getBinary() function reads data from the IR receiver and converts it to binary.
e) The convertHex() function converts the binary data to hexadecimal.
f) From this, we can take away that the remote control handling enhances user accessibility as well as convenience.

3) CoAP Server Setup (lines 94-121 & 399-404):
   a) In this part of the program, the CoAP server is set up to enable remote control of the smart light over a network.
   b) The server exposes resources that can be accessed through CoAP requests.
   c) By defining resources such as HelloWorldResource, RemoteResource, and PowerResource, the server allows clients to perform basic communication tests and control operations.

4) Sensor Specific Handling (lines 185-230):
   a) Here, motion detection and stepper motor functionality is implemented.
   b) The checkMotion() function asynchronously monitors the motion sensor to check if there are any changes.
   c) When motion is detected or stopped, corresponding actions are executed. In this case, the light is either turned on or off, respectively.
   d) The moveMotor() function asynchronously rotates the stepper motor according to the program's overall status of the light being on and off.
   e) The stepper motor works in correspondence with the LED strip so that the room lights turn on and off when the LED strip does.

5) Lighting Color Transition and Animation (lines 236-388):
   a) This segment implements the functionalities needed for the lighting effects of the program to run correctly.
   b) The setColor() function handles various color, brightness, and speed changes based on the IR remote button presses, CoAP PUT requests, and PIR motion sensor detections.
   c) The runRgbTransition() function handles the asynchronous execution of the RGB color transition so that it works simultaneously with other operations.
   d) As mentioned earlier, the colorsys library is used to convert between HSV and RGB to get a smooth RGB color transition.

IV. WiFi Protocol and Wireshark Analysis

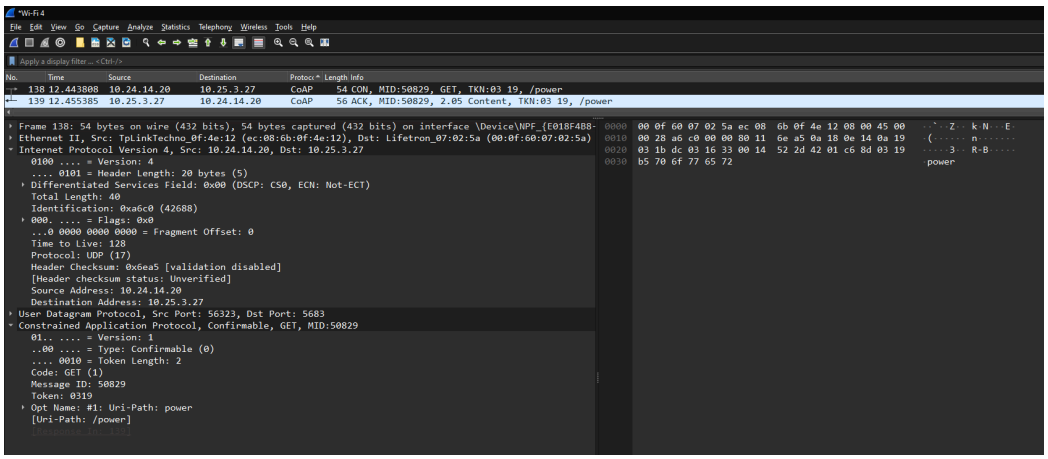


Figure 2.1: The client IP sends a GET request to the server's IP (Raspberry Pi) for the power resource

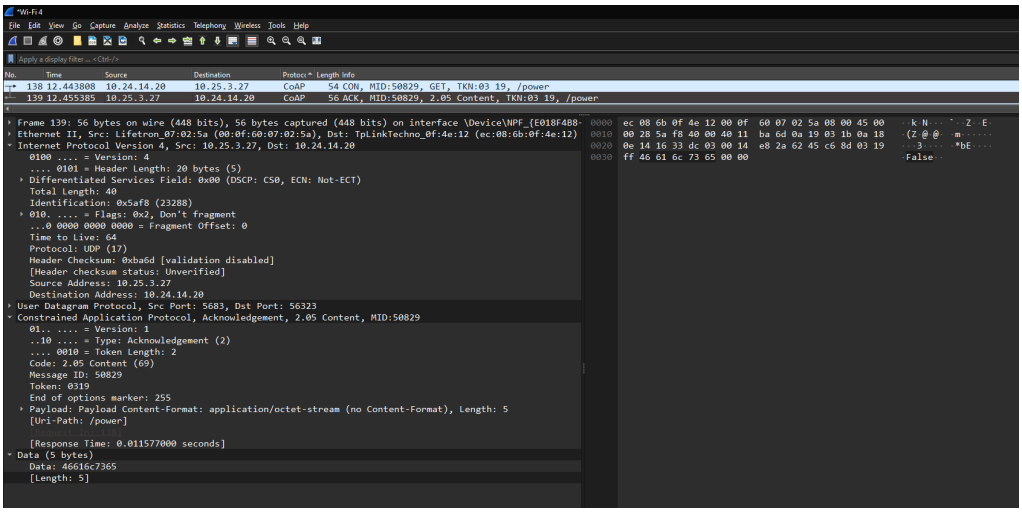


Figure 2.2: The server's IP sending the payload for the GET request back to the client's IP

As you can see in the pictures of the captured packets in Wireshark, our IoT application system utilizes the CoAP protocol for data transmission over WiFi. We chose to use the CoAP protocol for the fact that it is well suited for this IoT application due to its ability to easily interface with HTTP for integration with the web, very low overhead, and simplicity for constrained environments.[3] In Figure 2.1 we see that the client requests information from the "power" resource. This resource is dedicated to showing the power status of lights (if they are on or off). In Figure 2.2 we see that the server (the Raspberry Pi) sends the current status of the lights' power in response to the request. Next, we will analyze a PUT request and see how the CoAP protocol handled it for this program.

[3] Zach Shelby, Klaus Hartke, and Carsten Bormann, "RFC 7252: The Constrained Application Protocol (CoAP)," IETF Datatracker, accessed April 7, 2024, https://datatracker.ietf.org/doc/html/rfc7252.

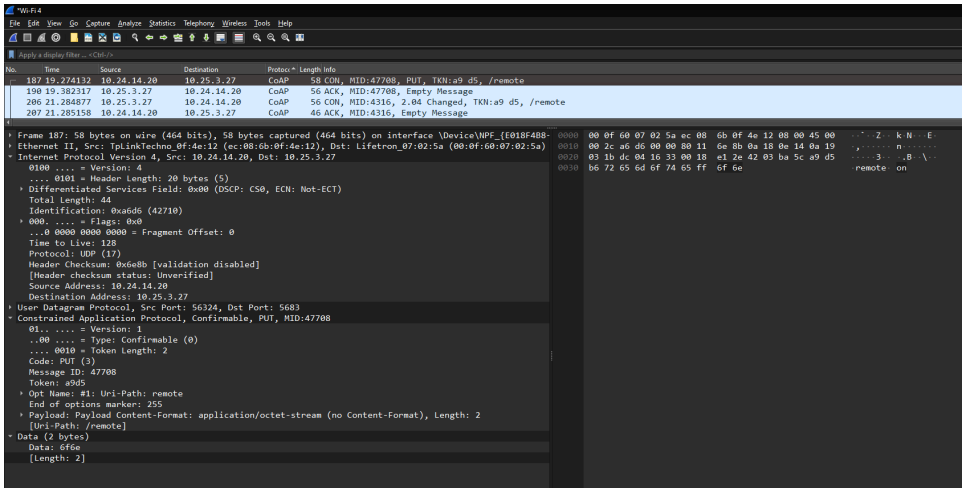


Figure 2.3: The client sends a PUT request to the server for the “remote” resource to receive the data “on”

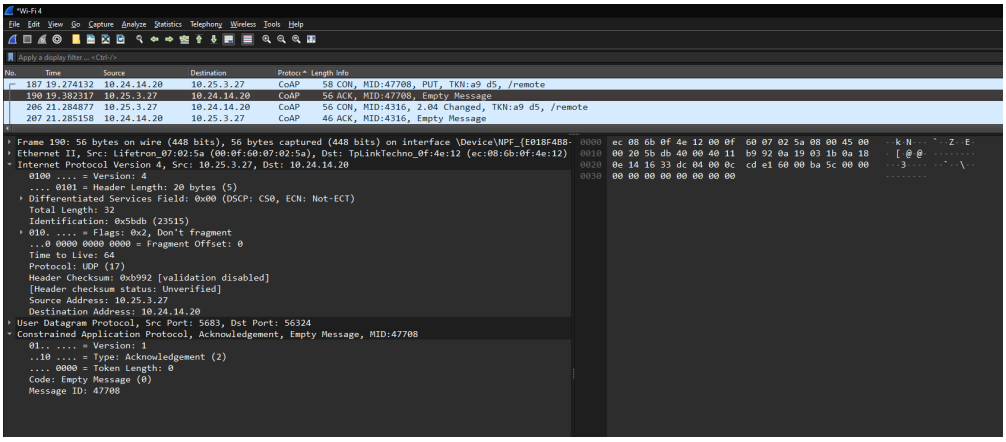


Figure 2.4: The server sends an acknowledgment packet to the client to let the client know that it received the request

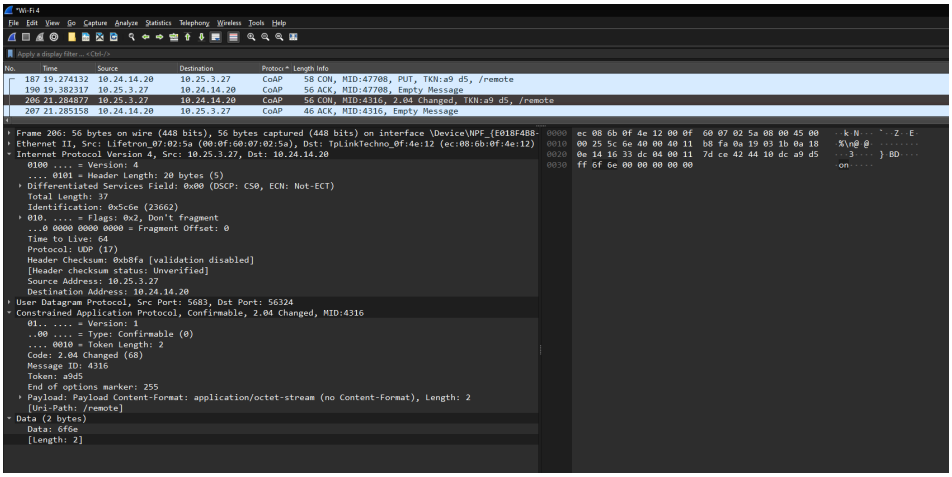


Figure 2.5: The server sends a packet to the client letting the client know that the data they requested to be changed in the “remote” resource has been changed to “on”

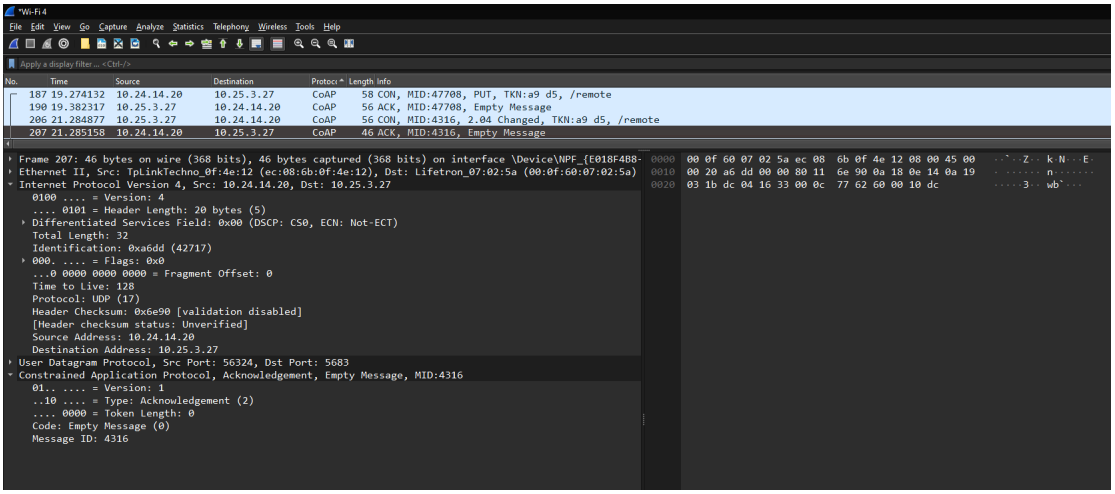


Figure 2.6: The client sends an acknowledgement packet to the server letting them know that they received the packet relating to the changed data

As we can see in the Wireshark packets for this PUT request, the CoAP protocol behaves differently when there is a PUT request compared to a GET request. When the client requested the “remote” resource to receive the data “on”, there were four packets involved in the exchange, whereas in the GET request there were only 2. As we can see in the details of the packets in Figures 2.4 and 2.6, these two new packets are acknowledgment packets.

V. Conclusion

We learned a lot through this project. The big takeaway from it was incorporating everything we learned from the previous projects to create something new. In particular, we learned how to use new sensors that we hadn’t used previously, and with them came new libraries to learn how to utilize. It is safe to conclude that this project was a great way to showcase what we learned throughout the entire semester while taking this course.

References

"28BYJ-48 Datasheet." ALLDATASHEET.COM. Accessed April 24, 2024. https://pdf1.alldatasheet.com/datasheet-pdf/view/1132391/ETC1/28BYJ-48.html.

"HC-SR501 Datasheet." ALLDATASHEET.COM. Accessed April 24, 2024. https://pdf1.alldatasheet.com/datasheet-pdf/view/1131987/ETC2/HC-SR501.html.

Lime-Parallelogram. "Lime-Parallelogram/IR-Code-Referencer: This Repo Contains the Code That Was Written in My Video about Using and IR Sensor in Python." GitHub. Accessed April 24, 2024. https://github.com/Lime-Parallelogram/IR-Code-Referencer.

Nuttall, Ben. "Gpiozero." gpiozero. Accessed April 25, 2024. https://gpiozero.readthedocs.io/en/stable/index.html.

Shelby, Zach, Klaus Hartke, and Carsten Bormann. "RFC 7252: The Constrained Application Protocol (CoAP)." IETF Datatracker. Accessed April 24, 2024. https://datatracker.ietf.org/doc/html/rfc7252.

Tim. "Control Multiple Fully-Addressable WS2812B RGB Led Strips with a Raspberry Pi Single Board Computer - Tutorial Australia." Core Electronics, February 15, 2023. https://core-electronics.com.au/guides/raspberry-pi/fully-addressable-rgb-raspberry-pi/.

"TL1838 Datasheet." ALLDATASHEET.COM. Accessed April 24, 2024. https://pdf1.alldatasheet.com/datasheet-pdf/view/1132045/ETC1/TL1838.html.