UIUC - CS 437 - Internet of Things

Lab 4 Report

Rodrigo Mendoza - rm36
Iziren Okhamafe - iokha2

Introduction

This project shows how we managed to create multiple simulated cars (or IoT things in general), connected them to a Greengrass core running on an EC2 instance, and linked a lambda function that would filter the data, and send information back to the simulated cars. Finally, we were able to visualize the data sent in a jupyter notebook, via a dataset made in AWS analytics. The video with the explanation can be found in this link. The code can be found at https://github.com/rm36/CarToCloudComms.

Part 1. Build the Cloud

1. Compared with the TCP-based client-server communication model, what are the pros and cons for MQTT-based publish/subscribe?

MQTT is a protocol that can run on TCP/IP or any other form of lossless protocol that supports bi-directional transfer (e.g. not UDP). MQTT is often seen as a lightweight protocol with little overhead, whereas in traditional TCP/IP client-server communications, there can be a huge amount of data overhead. With the publish/subscribe model, a client sends a message (the publisher) to another client or clients (subscribers). However, the connection between publisher and subscriber is often decoupled or practically non-existent. In the TCP client-server based model, there is a strong formulated connection that is done between client and server through a handshake. Finally, the publish/subscribe mechanism is quite simple to use in practice and it lets us use multiple topics as communication channels.

Because in MQTT the publisher and subscriber do not have a direct connection with one another, there is a third party known as a “broker” that handles connection(s) between the subscriber and publisher. The MQTT broker is a server itself that receives all client messages and then routes them to the publishers.

2. What are the tradeoffs in MQTT when there are a large number of clients?

The beauty of MQTT architecture is in the way where having multiple clients can automatically handle handover to automatic backup broker in case of a failure. Clients have the dual role of being a subscriber and/or publisher. Therefore, multiple clients can subscribe to a topic from a single broker. As a result, brokers may have to handle millions of concurrently connected MQTT Clients. Thus, it is important that when choosing a broker one should assess its integration and scalability as well as how it handles monitoring. Failure-resistance capabilities are important as well.

Part 1 Thoughts

We took an extremely long time getting through all of the AWS tutorials to get setup due to unforeseen issues such as:

- Long complicated set-ups in AWS.
- Having to re-do everything because the locale was set as us-west-1 by default and it didn't support the very last steps of the tutorial for greengrass.

At the end of this section we were able to have 500 devices communicating with a lambda and subscribing and publishing to different channels, with the EC2 running the Greengrass core as shown in the video. In the next parts, we changed it from 500 to 5 simulated devices to have each report a single CSV corresponding to a vehicle.

Part 2. Data Inference

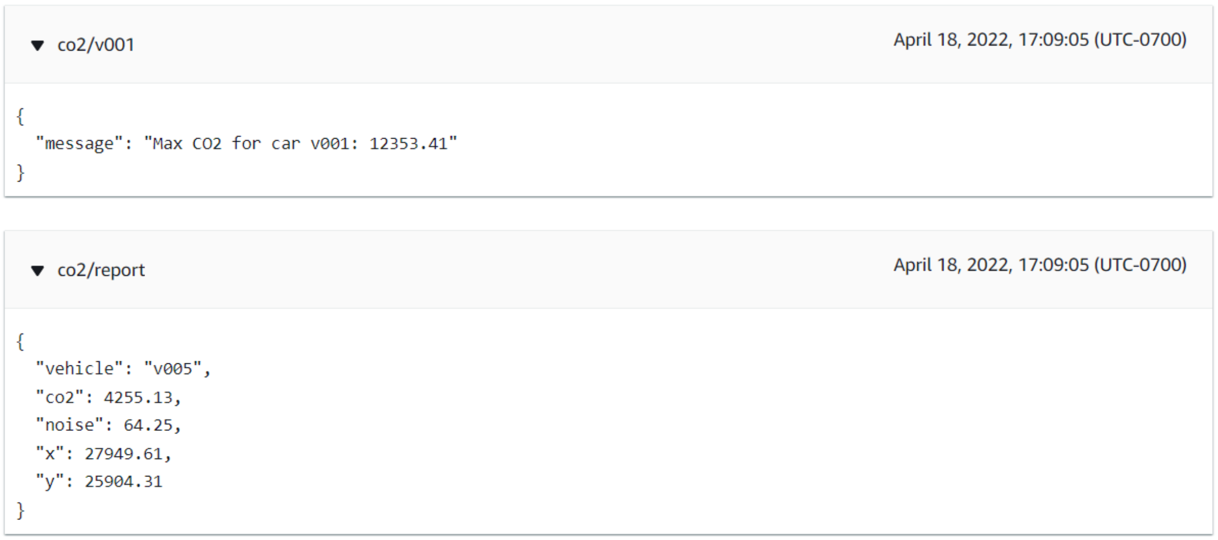
Once the setup was done, this part was relatively simple as we only updated the code as seen in the GitHub repository to subscribe and publish to different topics and calculate the maximum in the lambda.

We chose to use the co2/# topic to publish everything:

- Each simulated vehicle published 1. Their vehicle ID (v001, …, v005) along with their CO2 measurements and other information (noise, x and y position) to the topic **co2/report**.
- The lambda was linked only to the co2/report channel, so it would process the CO2 per vehicle, store it in a vehicle-keyed dictionary and return a message with

the maximum CO2 for that vehicle. The topic for it would be co2/VEHICLE-ID, for example **co2/v001**.

We can see an example of one report published by the 5th simulated vehicle and one message published by the lambda by subscribing to co2/# in the MQTT test client.


▼ co2/v001 April 18, 2022, 17:09:05 (UTC-0700)

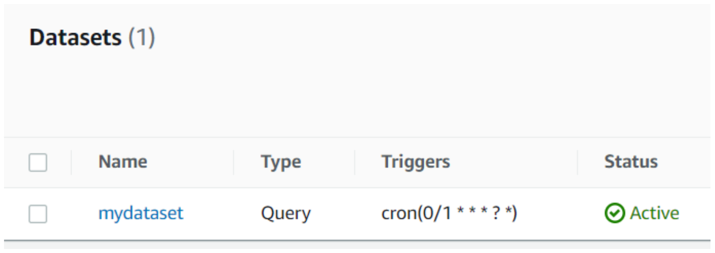
```
{
  "message": "Max CO2 for car v001: 12353.41"
}
```

▼ co2/report April 18, 2022, 17:09:05 (UTC-0700)

```
{
  "vehicle": "v005",
  "co2": 4255.13,
  "noise": 64.25,
  "x": 27949.61,
  "y": 25904.31
}
```

Part 3. IoT Analytics

For Part 3, we created a dataset that would SELECT * FROM co2, where co2 is the data store where all the publishings to co2/report would go. We set it to update every minute with a cron job as shown in the screenshot below.


Datasets (1)

	Name	Type	Triggers	Status
☐	mydataset	Query	cron(0/1 * * * ? *)	✓ Active

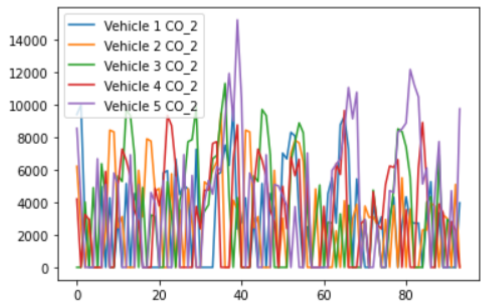
With all the reports in the dataset, we ran the code (in GitHub as jupyter_python_code.py) to produce the following plots:

- The CO2 for each vehicle plotted together.
- The noise of each vehicle plotted together.
- The Trajectories of each vehicle in a 2D plot.

The code is shown right before the plots in the screenshots.

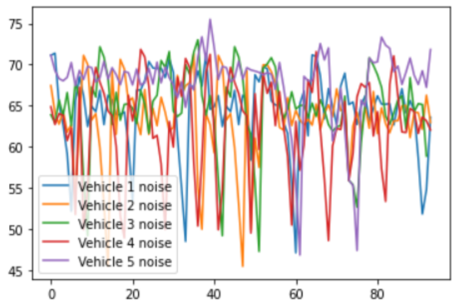
[57]:

```
# Get CO2 data from dataset
v1_co2 = data[data["vehicle"]=="v001"]["co2"].to_numpy()
v2_co2 = data[data["vehicle"]=="v002"]["co2"].to_numpy()
v3_co2 = data[data["vehicle"]=="v003"]["co2"].to_numpy()
v4_co2 = data[data["vehicle"]=="v004"]["co2"].to_numpy()
v5_co2 = data[data["vehicle"]=="v005"]["co2"].to_numpy()

# plot lines
plt.plot(v1_co2, label = "Vehicle 1 CO_2")
plt.plot(v2_co2, label = "Vehicle 2 CO_2")
plt.plot(v3_co2, label = "Vehicle 3 CO_2")
plt.plot(v4_co2, label = "Vehicle 4 CO_2")
plt.plot(v5_co2, label = "Vehicle 5 CO_2")
plt.legend()
plt.show()
```



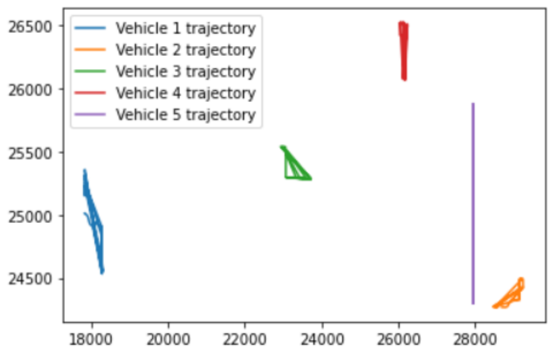
[58]:

```
# Get CO2 data from dataset
v1_noise = data[data["vehicle"]=="v001"]["noise"].to_numpy()
v2_noise = data[data["vehicle"]=="v002"]["noise"].to_numpy()
v3_noise = data[data["vehicle"]=="v003"]["noise"].to_numpy()
v4_noise = data[data["vehicle"]=="v004"]["noise"].to_numpy()
v5_noise = data[data["vehicle"]=="v005"]["noise"].to_numpy()

# plot lines
plt.plot(v1_noise, label = "Vehicle 1 noise")
plt.plot(v2_noise, label = "Vehicle 2 noise")
plt.plot(v3_noise, label = "Vehicle 3 noise")
plt.plot(v4_noise, label = "Vehicle 4 noise")
plt.plot(v5_noise, label = "Vehicle 5 noise")
plt.legend()
plt.show()
```



[59]:

```
# Get CO2 data from dataset
v1_x = data[data["vehicle"]=="v001"]["x"].to_numpy()
v2_x = data[data["vehicle"]=="v002"]["x"].to_numpy()
v3_x = data[data["vehicle"]=="v003"]["x"].to_numpy()
v4_x = data[data["vehicle"]=="v004"]["x"].to_numpy()
v5_x = data[data["vehicle"]=="v005"]["x"].to_numpy()
v1_y = data[data["vehicle"]=="v001"]["y"].to_numpy()
v2_y = data[data["vehicle"]=="v002"]["y"].to_numpy()
v3_y = data[data["vehicle"]=="v003"]["y"].to_numpy()
v4_y = data[data["vehicle"]=="v004"]["y"].to_numpy()
v5_y = data[data["vehicle"]=="v005"]["y"].to_numpy()

# plot lines
plt.plot(v1_x, v1_y, label = "Vehicle 1 trajectory")
plt.plot(v2_x, v2_y, label = "Vehicle 2 trajectory")
plt.plot(v3_x, v3_y, label = "Vehicle 3 trajectory")
plt.plot(v4_x, v4_y, label = "Vehicle 4 trajectory")
plt.plot(v5_x, v5_y, label = "Vehicle 5 trajectory")
plt.legend()
plt.show()
```


Vehicle 1 trajectory
Vehicle 2 trajectory
Vehicle 3 trajectory
Vehicle 4 trajectory
Vehicle 5 trajectory