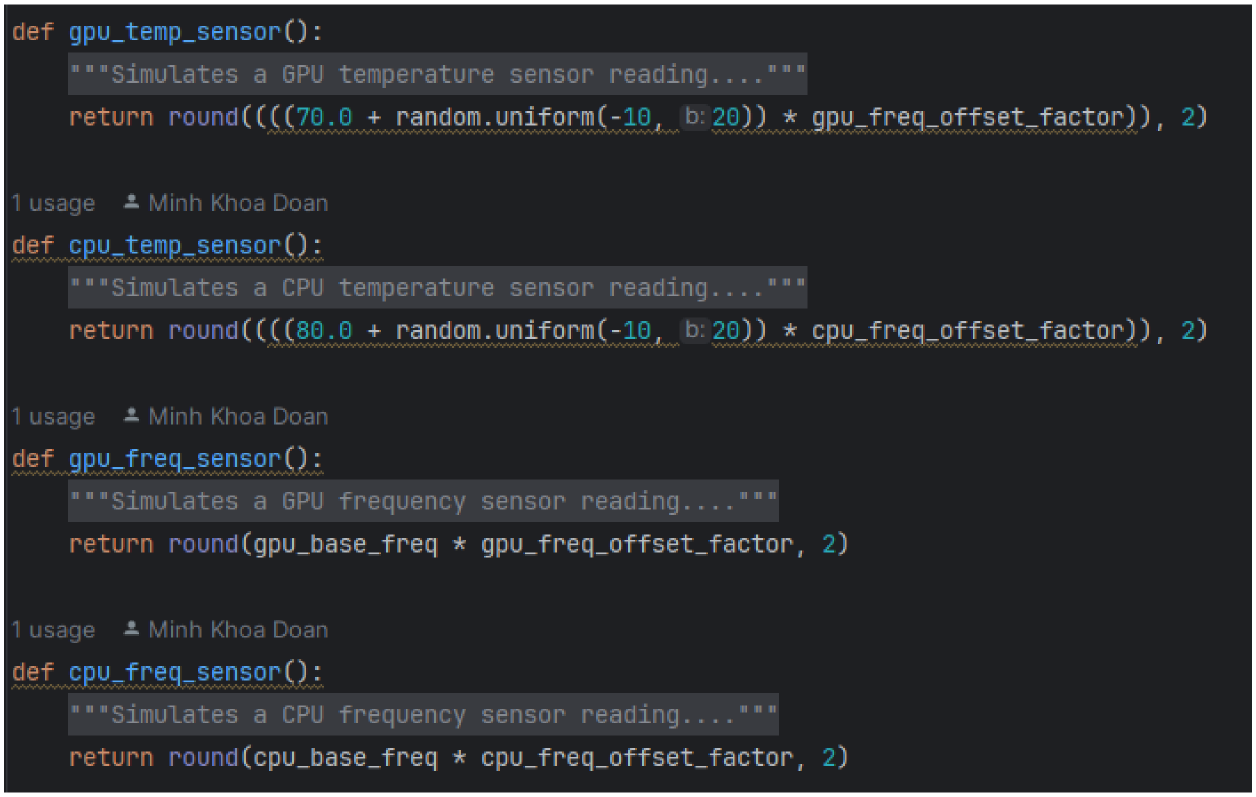
Fog Computing SoSe 2024 - Prototyping Assignment

Minh Khoa Doan 394025 – Jakub Duma 393128

The main idea of this application is a fog-based hardware monitoring tool written in Python. The local component collects (simulated) sensory data about

- GPU Frequency,
- CPU Frequency,
- GPU Temperature,
- and CPU Temperature.



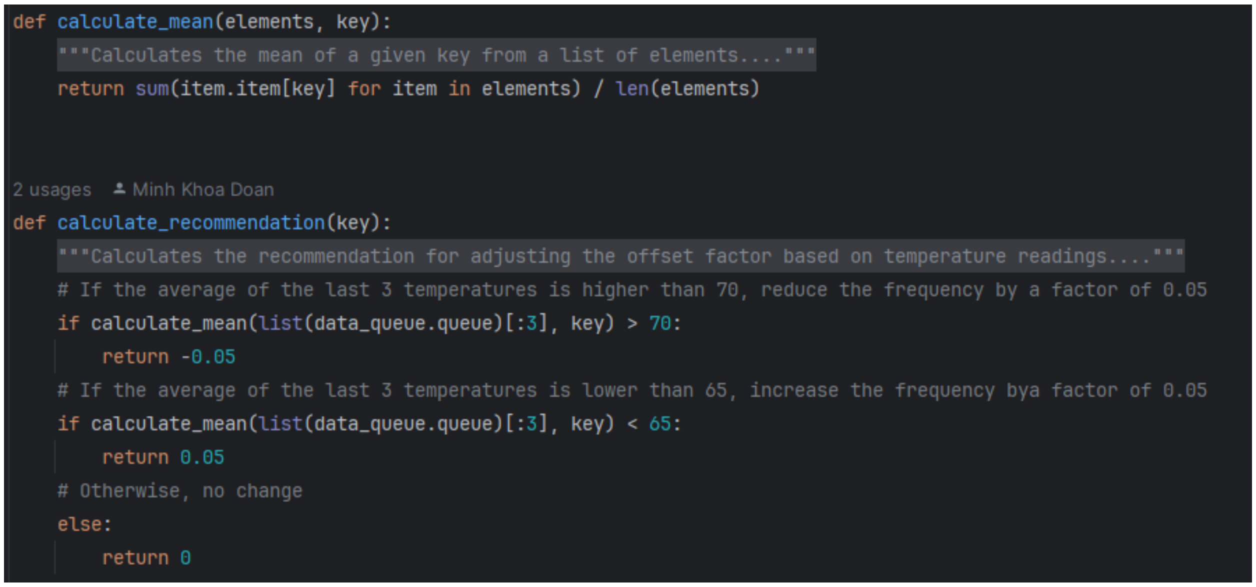
```
def gpu_temp_sensor():
    """Simulates a GPU temperature sensor reading...."""
    return round((((70.0 + random.uniform(-10,  b: 20)) * gpu_freq_offset_factor)), 2)

1 usage   Minh Khoa Doan
def cpu_temp_sensor():
    """Simulates a CPU temperature sensor reading...."""
    return round((((80.0 + random.uniform(-10,  b: 20)) * cpu_freq_offset_factor)), 2)

1 usage   Minh Khoa Doan
def gpu_freq_sensor():
    """Simulates a GPU frequency sensor reading...."""
    return round(gpu_base_freq * gpu_freq_offset_factor, 2)

1 usage   Minh Khoa Doan
def cpu_freq_sensor():
    """Simulates a CPU frequency sensor reading...."""
    return round(cpu_base_freq * cpu_freq_offset_factor, 2)
```

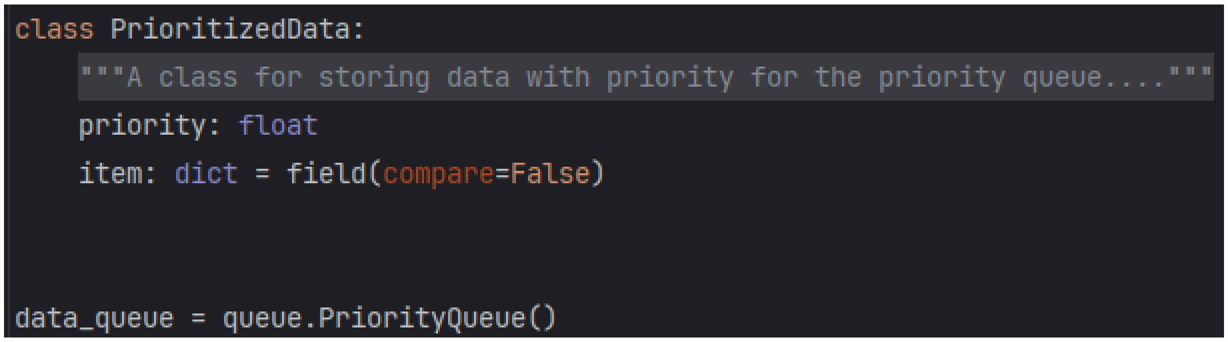
This data is regularly being sent to a cloud-based monitoring component, where the temperatures are being evaluated. If meeting a certain threshold, a recommendation is being sent back to the local component to lower/ increase/ keep the current frequency.



```python
def calculate_mean(elements, key):
    """Calculates the mean of a given key from a list of elements...."""
    return sum(item.item[key] for item in elements) / len(elements)


2 usages  Minh Khoa Doan
def calculate_recommendation(key):
    """Calculates the recommendation for adjusting the offset factor based on temperature readings...."""
    # If the average of the last 3 temperatures is higher than 70, reduce the frequency by a factor of 0.05
    if calculate_mean(list(data_queue.queue)[:3], key) > 70:
        return -0.05
    # If the average of the last 3 temperatures is lower than 65, increase the frequency bya factor of 0.05
    if calculate_mean(list(data_queue.queue)[:3], key) < 65:
        return 0.05
    # Otherwise, no change
    else:
        return 0
```

To ensure that recommendations are being made based on the most recent three sensory data sets, we have implemented a PriorityQueue which uses the negative timestamp value of each dataset to sort the incoming data.

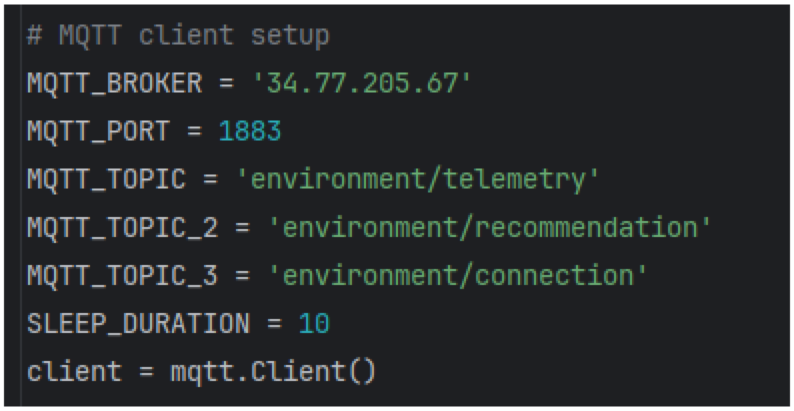


```python
class PrioritizedData:
    """A class for storing data with priority for the priority queue...."""
    priority: float
    item: dict = field(compare=False)


data_queue = queue.PriorityQueue()
```

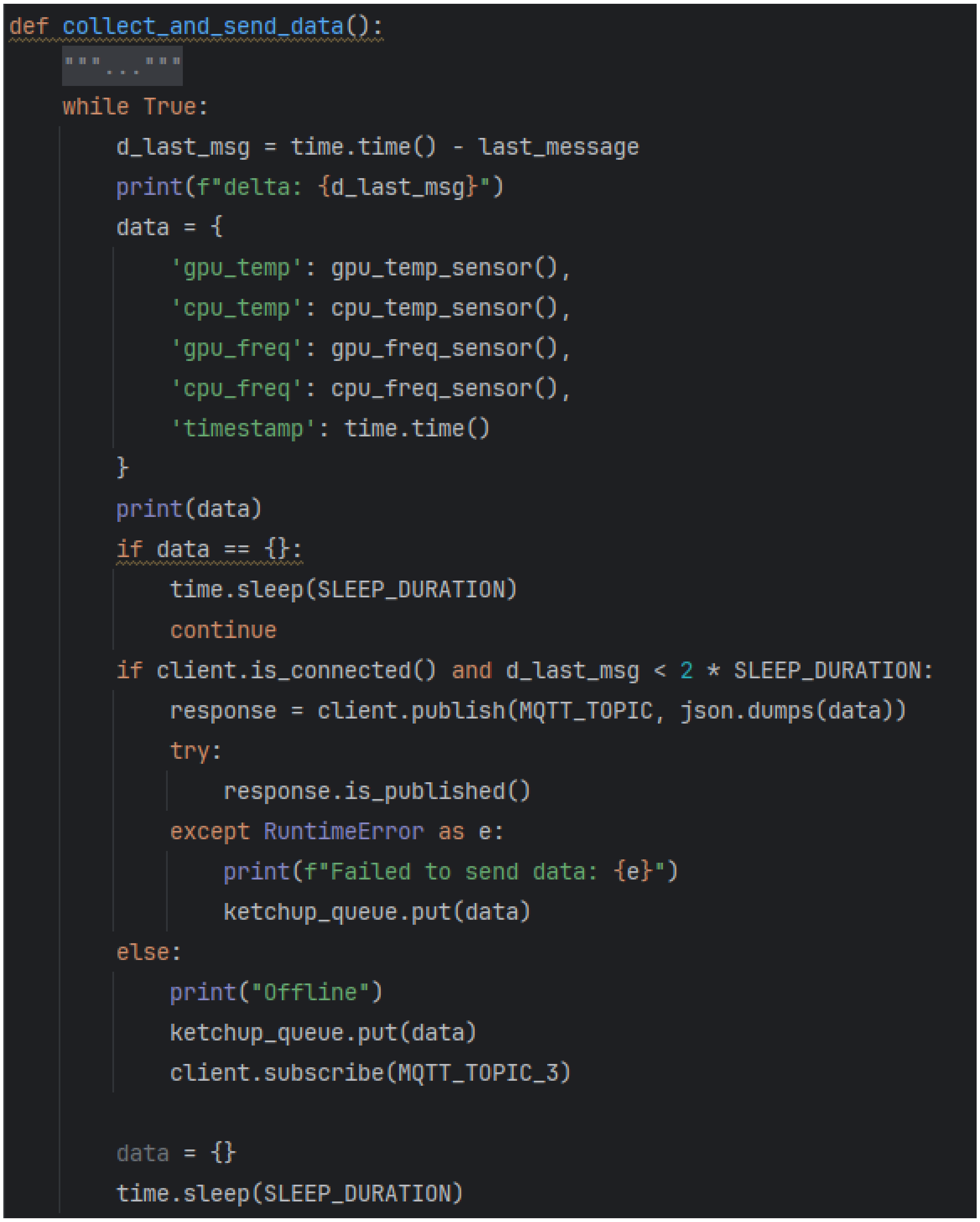
We have decided to use MQTT (a pub/sub model) for communication purposes. This allows us to publish data over the MQTT broker from the sensors which the cloud can subscribe to and vice versa. We have three pub/sub channels used for

1. sensory data,
2. recommendations calculated by the monitoring tool,
3. connection status.



```python
# MQTT client setup
MQTT_BROKER = '34.77.205.67'
MQTT_PORT = 1883
MQTT_TOPIC = 'environment/telemetry'
MQTT_TOPIC_2 = 'environment/recommendation'
MQTT_TOPIC_3 = 'environment/connection'
SLEEP_DURATION = 10
client = mqtt.Client()
```

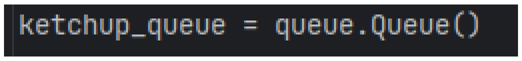
The following screenshot represents the data collection and sending mechanism of the client



```python
def collect_and_send_data():
    """..."""
    while True:
        d_last_msg = time.time() - last_message
        print(f"delta: {d_last_msg}")
        data = {
            'gpu_temp': gpu_temp_sensor(),
            'cpu_temp': cpu_temp_sensor(),
            'gpu_freq': gpu_freq_sensor(),
            'cpu_freq': cpu_freq_sensor(),
            'timestamp': time.time()
        }
        print(data)
        if data == {}:
            time.sleep(SLEEP_DURATION)
            continue
        if client.is_connected() and d_last_msg < 2 * SLEEP_DURATION:
            response = client.publish(MQTT_TOPIC, json.dumps(data))
            try:
                response.is_published()
            except RuntimeError as e:
                print(f"Failed to send data: {e}")
                ketchup_queue.put(data)
        else:
            print("Offline")
            ketchup_queue.put(data)
            client.subscribe(MQTT_TOPIC_3)

        data = {}
        time.sleep(SLEEP_DURATION)
```

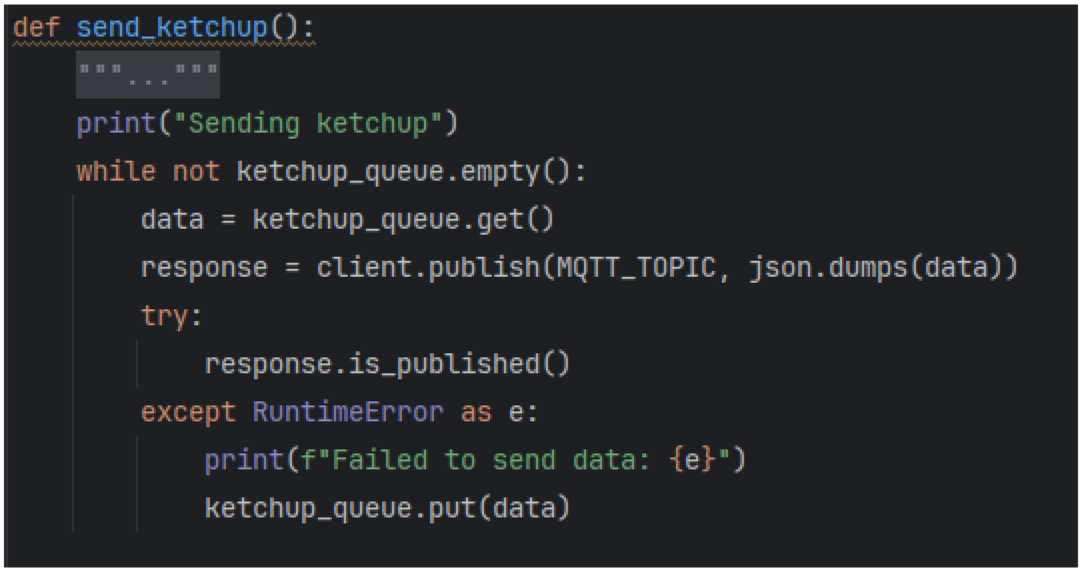
To ensure integrity and prevent data loss, we have implemented a catchup queue for both the client and cloud component. From the client perspective, it is used to store sensory data that has not yet been sent to the cloud for evaluation. This is relevant in any case of connection loss to the broker/ cloud.



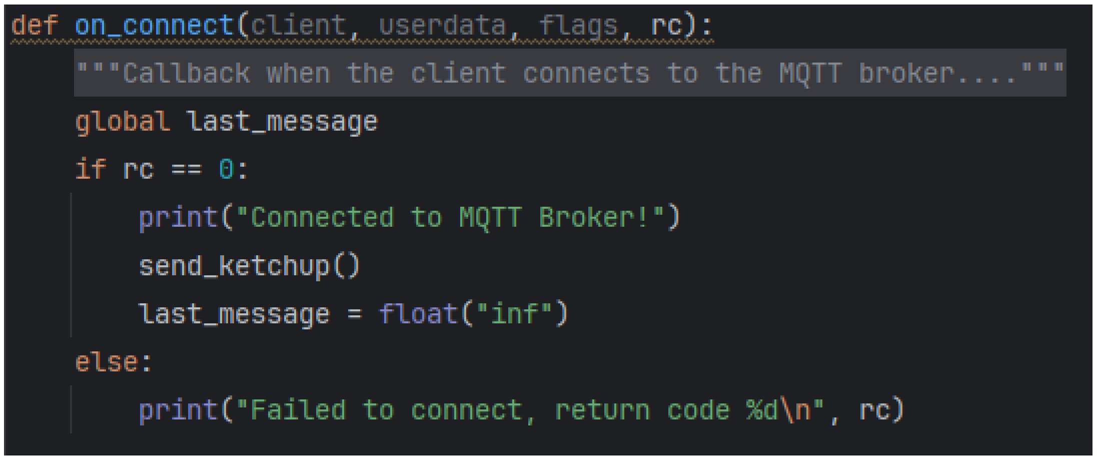
```python
ketchup_queue = queue.Queue()
```

The cloud uses the catchup queue to store its recommendations based on the most recent sensory data.

On reconnect, the respective component publishes stored data in this catchup queue. The following represents the client catchup queue mechanism.

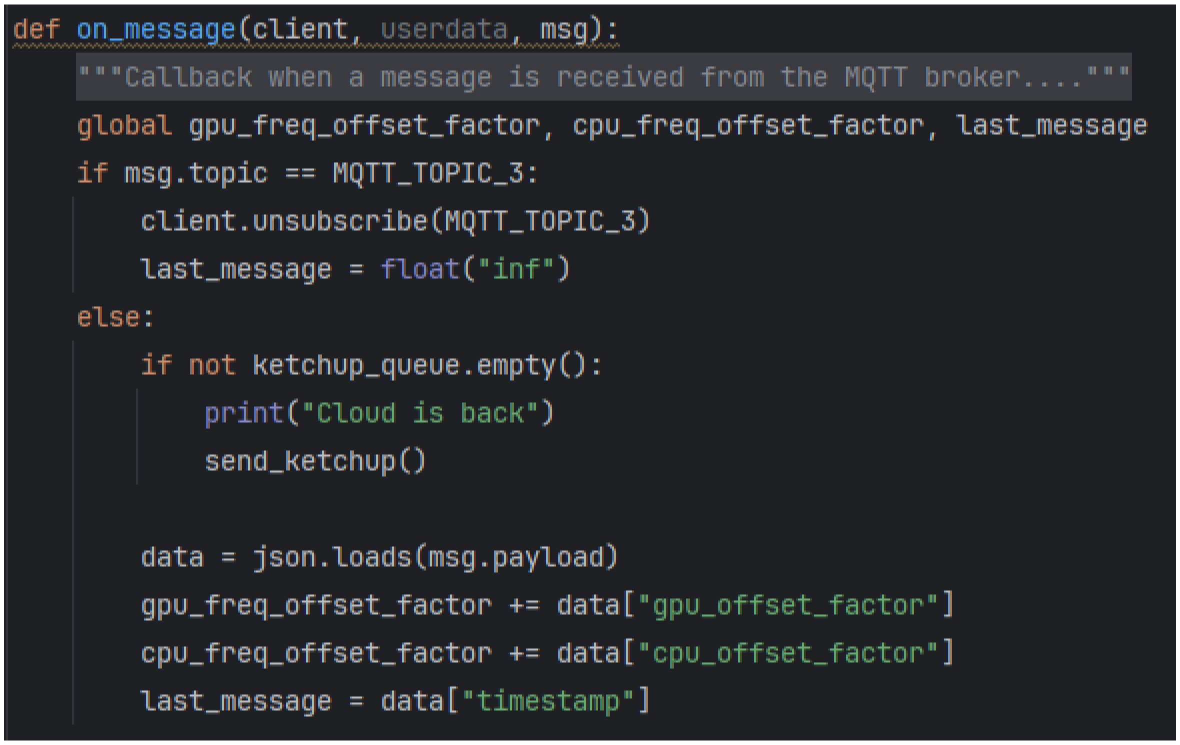


```python
def send_ketchup():
    """..."""
    print("Sending ketchup")
    while not ketchup_queue.empty():
        data = ketchup_queue.get()
        response = client.publish(MQTT_TOPIC, json.dumps(data))
        try:
            response.is_published()
        except RuntimeError as e:
            print(f"Failed to send data: {e}")
            ketchup_queue.put(data)
```



```python
def on_connect(client, userdata, flags, rc):
    """Callback when the client connects to the MQTT broker...."""
    global last_message
    if rc == 0:
        print("Connected to MQTT Broker!")
        send_ketchup()
        last_message = float("inf")
    else:
        print("Failed to connect, return code %d\n", rc)
```

When the client receives a recommendation back from the cloud it updates the respective frequencies accordingly.



```python
def on_message(client, userdata, msg):
    """Callback when a message is received from the MQTT broker...."""
    global gpu_freq_offset_factor, cpu_freq_offset_factor, last_message
    if msg.topic == MQTT_TOPIC_3:
        client.unsubscribe(MQTT_TOPIC_3)
        last_message = float("inf")
    else:
        if not ketchup_queue.empty():
            print("Cloud is back")
            send_ketchup()

        data = json.loads(msg.payload)
        gpu_freq_offset_factor += data["gpu_offset_factor"]
        cpu_freq_offset_factor += data["cpu_offset_factor"]
        last_message = data["timestamp"]
```