PHASE-3

DEVELOPMENT-1

- Start building the IoT traffic monitoring system.

Role of IoT in Smart City Traffic Management

With the pressing demand for advanced communication & network technologies, digitalization is the driving force that stimulates the implementation of smart traffic control using IoT capabilities.

It enables them to;

- Expand the capacity of city streets without having to build new roads.
- Optimize the traffic flow and keep the drivers safe. It would include cameras, sensors, and cellular technologies that automatically adjust traffic lights, expressway lanes, speed limits, and highway exit counters.
- Transmit accurate information about available parking spaces to citizens in real-time
- Collect data on congestion and improve traffic signaling to reduce blockages and optimize commute
- Locate incidents and report them to emergency rooms immediately with road sensors and video surveillance
- Employ real-time data feeds to ensure the streetlights turn dim or brighten up per the changing weather conditions and the onset of day and night

Advantages of a Smart Traffic Management System

Cleaner, greener, safer, and more accessible roads are a few benefits of implementing IoT and intelligent technology.

It helps with the following:

- Reducing traffic jams and accidents on the streets
- Ensuring immediate clearance for emergency vehicles
- Facilitating safer and shorter commute times
- Reducing congestion & energy consumption at intersections
- Offering significant productivity benefits with real-time monitoring of crucial infrastructures
- Reducing operating costs with efficient traffic management processes
- Ensuring compliance with the regulations for reducing the carbon footprint
- Saving billions of gallons of fuel wasted every year
- Accurate tracking & quick recovery of lost and stolen vehicles

Functioning of Traffic Monitoring System Using IoT Capabilities

This intelligent system comprises several components, including wireless sensors, RFID tags, and BLE beacons installed at the traffic signals to monitor the movement of vehicles. A real-time data analytics tool connects the Geographic Information System (GIS-enabled) digital roadmap with control rooms for real-time traffic monitoring.

The smart traffic management system captures the images of vehicles at the signals using the digital image processing technique. This data is then transferred to the control room via wireless sensors. The system also leverages BLE beacons or RFID tags to track the movement of vehicles and keep traffic congestion in control, track down stolen vehicles and even clear the road for emergency vehicles that are installed with RFID readers.

Application of IoT in Traffic Management

City governments can improve their operations & infrastructure by placing IoT sensors and tracking devices on roads and highways for recording, analyzing, and sharing data in real-time.

- Deploy IoT devices (e.g., traffic flow sensors,

cameras) in strategic locations to monitor traffic conditions.

Building an IoT traffic monitoring system involves deploying IoT devices like traffic flow sensors and cameras in strategic locations to monitor traffic conditions. Below is a detailed plan to get started:

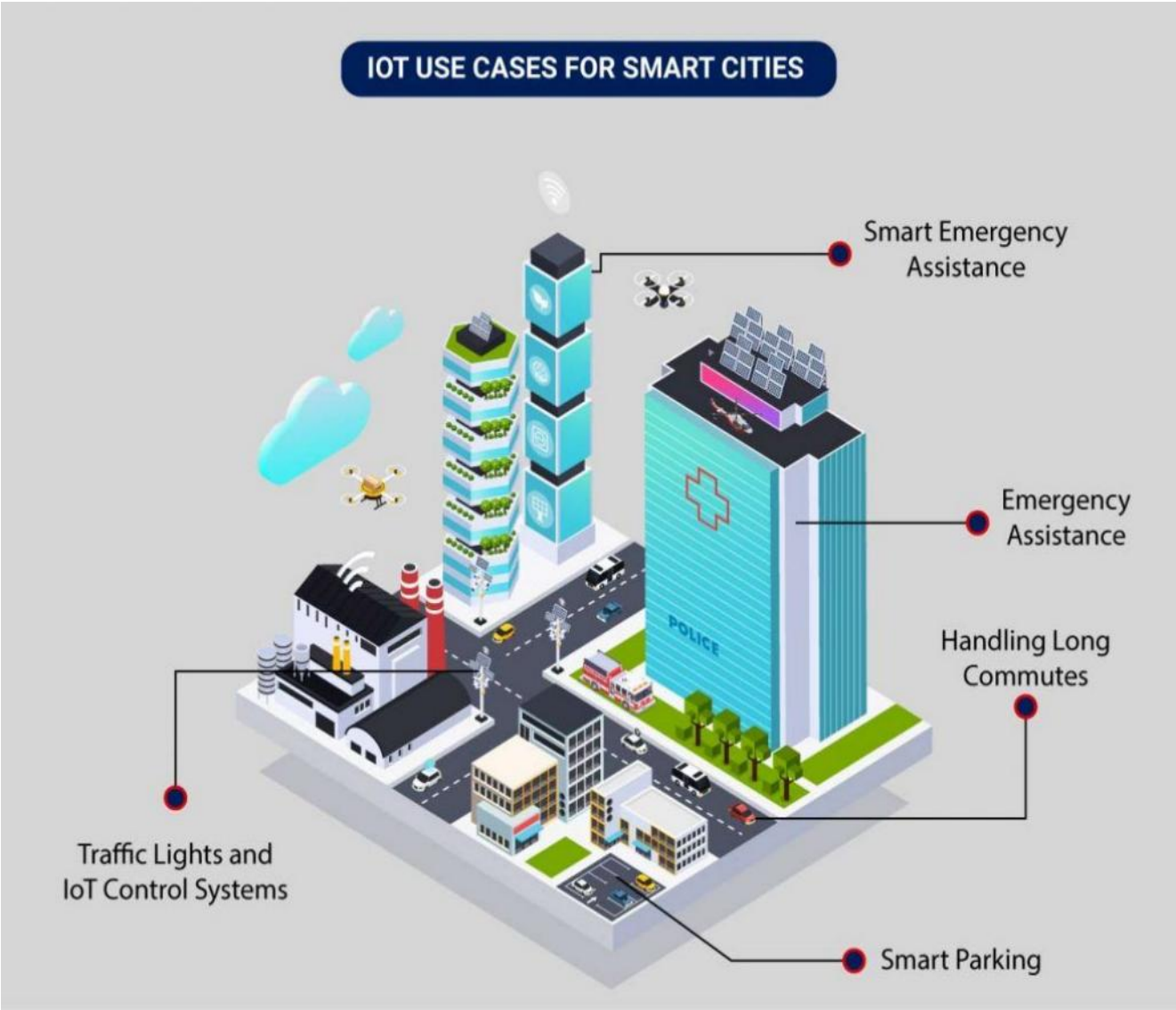


Fig: IOT use cases for smart cities.

1. Define Objectives:

- Determine the specific goals and objectives of your traffic monitoring system. This could include real-time traffic data collection, congestion detection, traffic management, or security.

2. Select IoT Devices:

- Choose the appropriate IoT devices for your project. For traffic monitoring, consider the following:

  - Traffic flow sensors: These can be inductive loop sensors, ultrasonic sensors, or infrared sensors to detect vehicle presence and count.

  - Cameras: High-definition cameras with video analytics capabilities for monitoring traffic conditions.

3. Choose Strategic Locations:

- Identify key locations where you will deploy the IoT devices. These locations should provide valuable data and insights. Common deployment areas include intersections, highways, toll booths, parking lots, and urban centers.

4. Secure Power and Connectivity:

- Ensure that IoT devices have a stable power source. This may involve wiring to the electrical grid, using solar panels, or battery solutions.

- Establish reliable internet connectivity for data transmission. Options include Wi-Fi, Ethernet, or cellular connections.

5. Data Collection and Sensing:

- Set up the traffic flow sensors and cameras at the selected locations.

- Configure the sensors to collect relevant data, such as vehicle count, speed, and direction.

- Cameras should capture images or videos that can be analyzed for traffic conditions.

6. Data Processing and Analysis:

- Implement data processing and analysis components on the IoT devices or a centralized server.

- Use computer vision and image processing techniques to analyze camera data.

- Apply algorithms to interpret sensor data and generate meaningful traffic information.

7. Real-time Communication:

- Develop communication protocols for the IoT devices to send real-time data to a central server or cloud platform. MQTT or HTTP may be used for this purpose.

- Ensure data is sent securely, and implement mechanisms to handle data loss or network interruptions.

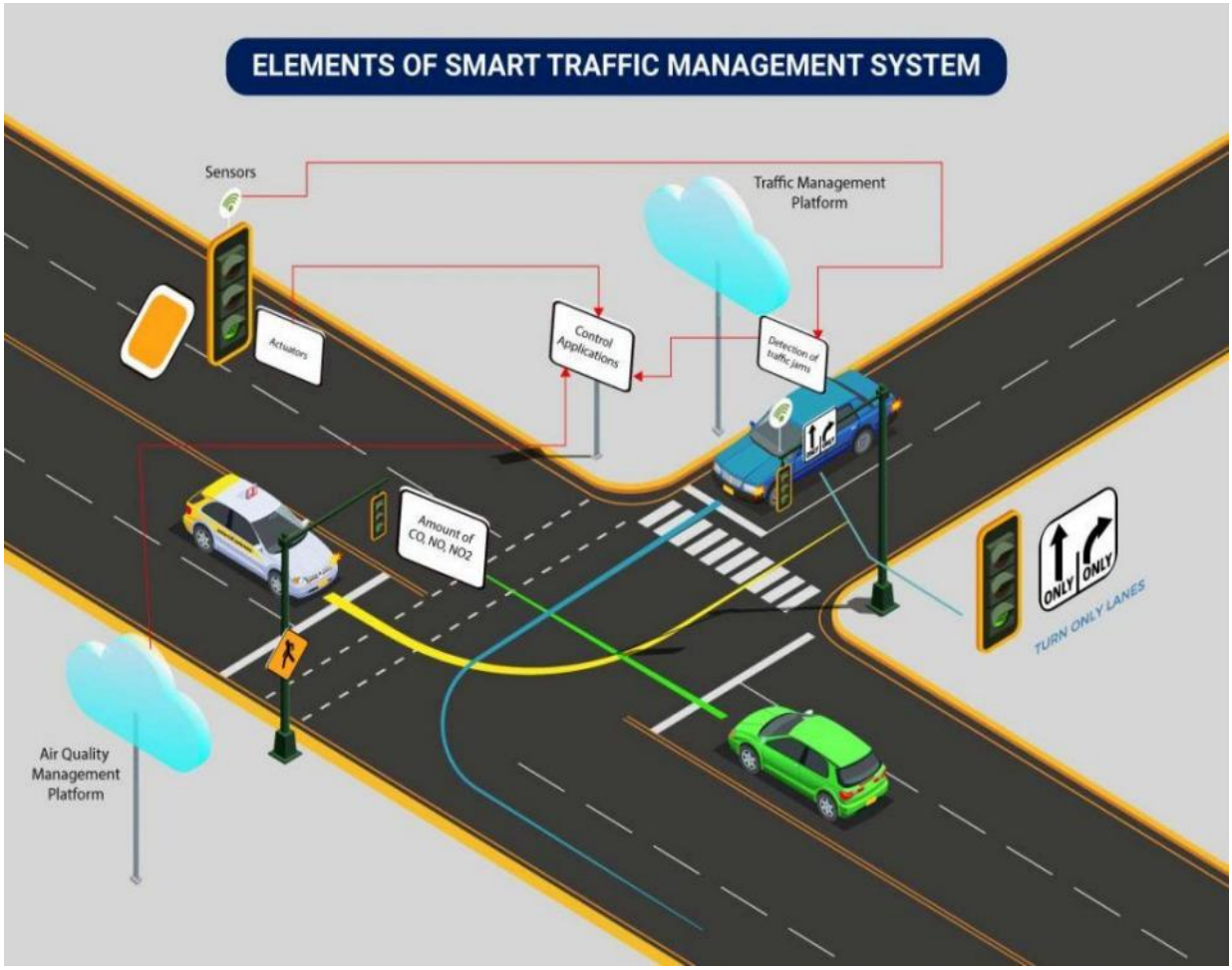


Fig: Elements of smart traffic management system.

8. Data Storage and Visualization:

- Set up a database to store historical traffic data.

- Create a dashboard or visualization platform to display real-time and historical traffic information.

- Use tools like Grafana, Tableau, or custom web

applications for visualization.

9. Alarms and Notifications:

- Implement alerts and notifications based on predefined thresholds or anomalies detected in traffic data.

- Notifications could be sent to traffic management personnel, law enforcement, or the public through various channels.

10. Maintenance and Calibration:

- Regularly maintain and calibrate the IoT devices to ensure accuracy.

- Check for wear and tear, sensor accuracy, and camera functionality.

11. Security and Privacy:

- Implement security measures to protect data and devices from unauthorized access.

- Ensure privacy compliance, especially if cameras are used, by anonymizing data.

12. Scalability and Future Expansion:

- Design the system to be scalable so that more IoT devices can be added as needed.

- Consider future expansions and technologies, such as 5G connectivity or edge computing.

13. Documentation and Training:

- Document the setup, configuration, and maintenance procedures.

- Provide training to staff responsible for operating and maintaining the system.

14. Compliance:

- Ensure compliance with local regulations and privacy laws related to data collection, storage, and sharing.

15. Testing and Optimization:

- Thoroughly test the system's functionality and performance.

- Continuously optimize algorithms and data collection strategies based on real-world feedback.

Building an IoT traffic monitoring system is a complex and ongoing process that requires careful planning,

hardware and software integration, and ongoing maintenance to provide valuable traffic insights and improve road safety and efficiency.

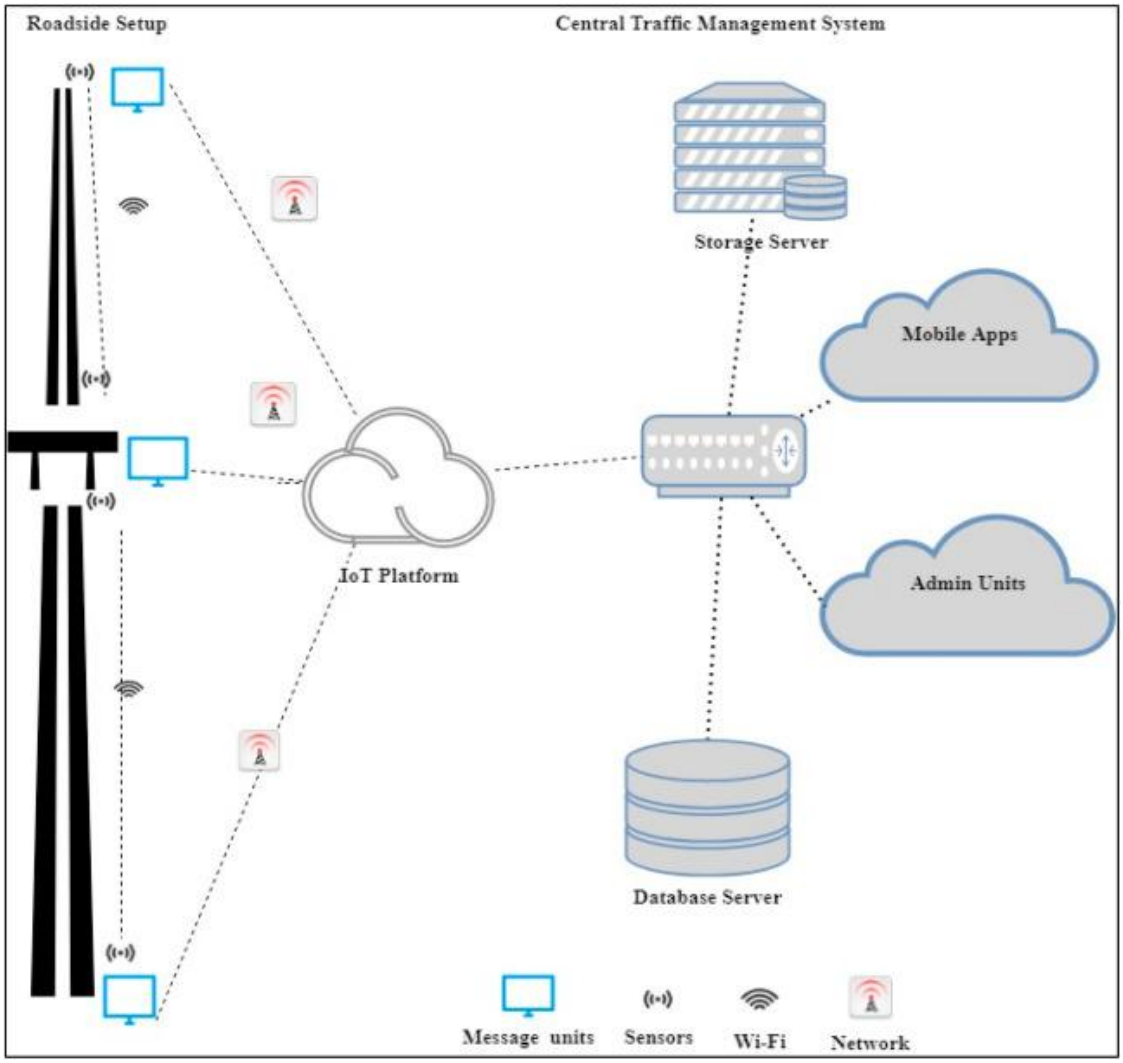


Fig: Internet through various IOT resource work together.

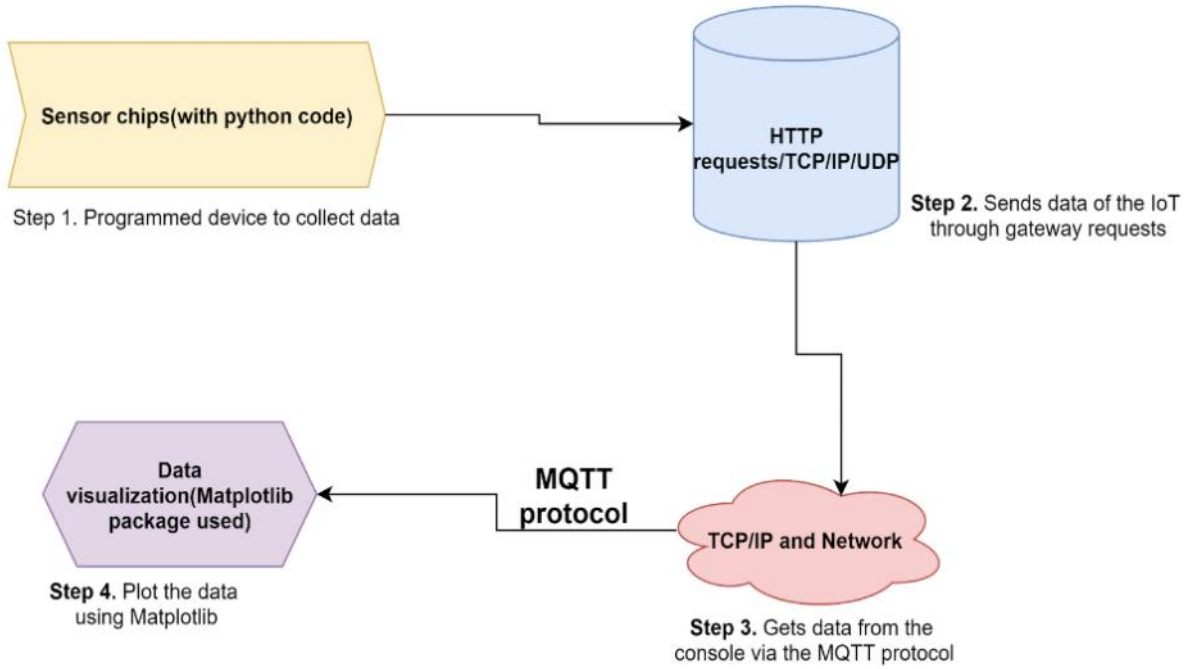
- Develop a Python script on the IoT devices to send real-time traffic data to the traffic information platform.

Here's a basic script structure to get you started using Python with the MQTT protocol, which is a common choice for IoT communication.

IOT SENSORS SIMULATORS USED IN PYTHON PROGRAMMING INCLUDE:

MQ TELEMETRY TRANSPORT (MQTT) SENSOR SIMULATOR

MQTT protocol for the IoT in Python enables high-speed data exchange with low payload communication between the devices. User-friendly requests of MQTT are made directly in Python. Data is collected in real-time and easily analyzed in mathematical computation libraries like matplotlib. The diagram below shows the steps used for the data flow:



Here is the complete Python code for basic implementation of the IoT Sensor Data Processing.

Program.

```
# Sample sensor data from various IoT devices

sensor_data = [

    {'type': 'temperature', 'value': 20},

    {'type': 'temperature', 'value': 22},

    {'type': 'humidity', 'value': 55},

    {'type': 'temperature', 'value': 24},

    {'type': 'humidity', 'value': 60},

    {'type': 'temperature', 'value': 25},

    {'type': 'humidity', 'value': 50},]
```

```
# Group sensor data by type
grouped_data = {}
for data in sensor_data:
    sensor_type = data['type']
    if sensor_type not in grouped_data:
        grouped_data[sensor_type] = []
    grouped_data[sensor_type].append(data['value'])

# Calculate average values for each sensor type and store them in a list
average_values = []
for sensor_type, values in grouped_data.items():
    average_value = sum(values) / len(values)
    average_values.append({'type': sensor_type, 'average_value': average_value})

# Sort the list of average values
average_values.sort(key=lambda x: x['average_value'])

# Print the sorted list of average sensor values
print(average_values)
```

Output:

```
[{'type': 'temperature', 'average_value': 22.75}, {'type': 'humidity', 'average_value': 55.0}]
```

Here is the complete Python code for advanced implementation of the IoT Sensor Data Processing:

Program.

```
import time

from random import choice, uniform


# Function to simulate real-time sensor data collection

def collect_sensor_data():

    sensor_types = ['temperature', 'humidity', 'pressure']

    return {'type': choice(sensor_types), 'value': round(uniform(10, 100), 2)}


# Function to calculate the average values of sensor data

def calculate_average(sensor_data_list):

    if not sensor_data_list:

        return None
```

```
    total_value = sum(sensor_data_list)
    average_value = total_value / len(sensor_data_list)
    return round(average_value, 2)

# Initialize empty sensor data and average values dictionaries
sensor_data = {}
average_values = {}

# Continuously collect sensor data and update average values
while True:
    # Collect real-time sensor data
    data = collect_sensor_data()

    # Update the sensor_data dictionary
    sensor_type = data['type']
    if sensor_type not in sensor_data:
        sensor_data[sensor_type] = []
    sensor_data[sensor_type].append(data['value'])

    # Calculate the updated average value for the sensor type
```

```
    average_value = calculate_average(sensor_data[sensor_type])

    average_values[sensor_type] = average_value



    # Sort the average values dictionary

    sorted_average_values = sorted(average_values.items(),
key=lambda x: x[1])



    # Print the sorted average values

    print("Sorted Average Sensor Values:", sorted_average_values)



    # Wait for 1 second before collecting the next sensor data

    time.sleep(1)
```

Output (for first 17 seconds):

```
Sorted Average Sensor Values: [('temperature', 12.52)]
Sorted Average Sensor Values: [('temperature', 12.52), ('humidity', 32.46)]
Sorted Average Sensor Values: [('temperature', 12.52), ('humidity', 32.46),
('pressure', 37.19)]
Sorted Average Sensor Values: [('temperature', 17.22), ('humidity', 32.46),
('pressure', 37.19)]
Sorted Average Sensor Values: [('temperature', 17.22), ('humidity', 32.46),
('pressure', 36.58)]
Sorted Average Sensor Values: [('temperature', 17.22), ('humidity', 32.46),
('pressure', 54.14)]
Sorted Average Sensor Values: [('temperature', 17.22), ('humidity', 32.46),
('pressure', 56.6)]
Sorted Average Sensor Values: [('temperature', 31.9), ('humidity', 32.46),
('pressure', 56.6)]
Sorted Average Sensor Values: [('humidity', 32.46), ('temperature', 47.19),
('pressure', 56.6)]
Sorted Average Sensor Values: [('humidity', 32.46), ('temperature', 50.14),
```

('pressure', 56.6)]
Sorted Average Sensor Values: [('humidity', 32.46), ('temperature', 50.14), ('pressure', 58.25)]
Sorted Average Sensor Values: [('humidity', 32.46), ('temperature', 50.14), ('pressure', 53.2)]
Sorted Average Sensor Values: [('humidity', 26.76), ('temperature', 50.14), ('pressure', 53.2)]
Sorted Average Sensor Values: [('humidity', 26.76), ('temperature', 50.14), ('pressure', 58.08)]
Sorted Average Sensor Values: [('humidity', 26.76), ('temperature', 46.12), ('pressure', 58.08)]
Sorted Average Sensor Values: [('humidity', 26.76), ('temperature', 43.06), ('pressure', 58.08)]
Sorted Average Sensor Values: [('humidity', 26.76), ('temperature', 43.06), ('pressure', 57.05)]

Program.

```
import paho.mqtt.client as mqtt
import json

mqtt_broker_address = "mqtt.example.com"
mqtt_port = 1883
mqtt_topic = "traffic_data"

client = mqtt.Client("TrafficDataPublisher")

def collect_traffic_data():
    data = {
        "location": "Intersection A",
        "traffic_flow": "Heavy",
```

```
        "timestamp": "2023-10-16 14:30:00"
    }
    return json.dumps(data)

def on_connect(client, userdata, flags, rc):
    print("Connected with result code " + str(rc))
    client.subscribe(mqtt_topic)

client.on_connect = on_connect
client.connect(mqtt_broker_address, mqtt_port, 60)

while True:
    traffic_data = collect_traffic_data()
    client.publish(mqtt_topic, traffic_data)
    print("Published: " + traffic_data)
```

Make sure to replace the MQTT broker details and implement the collect_traffic_data() function to fetch real-time traffic data from your devices.