Project title: Traffic management system using IOT

OBJECTIVE:

An IoT-based Traffic Management System (TMS) utilizes connected devices and data analytics to monitor and improve traffic flow in urban and suburban areas. By deploying tools such as traffic cameras, GPS trackers, and various sensors at key locations like intersections and highways, the system collects real-time data on vehicle count, speed, traffic density, and environmental conditions.

This data is securely stored and processed using advanced algorithms and Python-based analysis to detect congestion, estimate travel times, and generate timely alerts. The insights are then presented to commuters through user-friendly platforms or mobile applications, offering up-to-date traffic information and suggesting alternative routes.

Integrating IoT devices with these platforms ensures seamless communication and continuous data exchange, enhancing the system's efficiency. Overall, this approach leads to reduced traffic congestion, improved road safety, better-informed travel decisions, and contributes to more effective urban planning and transportation management.

IOT DEVICE DESIGN:

To achieve real-time traffic monitoring, a strategically planned deployment of IoT sensors is essential. Few sensors and components which are used include:
Serial Peripheral Interface (SPI):- It allows us to communicate with the peripheral devices quickly over short distances.
RFID: - A library for interfacing RFID readers with Arduino board UART.
Servo: - This library allows an Arduino board to control rotation of Servo motors.Software Serial: - This library allows serial communication on other digital pins of the Arduino using software to replicate the functionality.
MFRC 522:- Read and write different types of Radio Frequency Identification cards on the Arduino board.

ARDUINO DESIGN:

Arduino UNO which is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins. The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins. These two arduino boards are combined to form the central controller of the entire system which is configured by python. The sensors and components are then properly connected to the configured aurdino which make up the entire system. This IoT system offers good traffic flow and congestion control.

RFID's role in the project:

- Emergency Vehicle Identification:
  - RFID tags on emergency vehicles contain unique identification information.
  - RFID readers detect and read these tags as vehicles approach the junction.
- Automated Traffic Signal Control:

- RFID reader communicates with the traffic control system (Arduino MEGA) upon tag detection.
- Traffic signals are adjusted automatically to clear the way for emergency vehicles.

- Congestion Prevention:
  - RFID technology differentiates emergency vehicles from regular traffic.
  - This prevents unnecessary congestion by adjusting signals and barricades.
- Reduced Manual Intervention:
  - Human intervention in traffic management is minimized.
  - Replaces manual inspections, reducing traffic delays.
- Manual Override Option:
  - Includes a manual operation feature for exceptional situations.
  - Remotes can be used to open barricades when needed.
- Servo Motors for Barricade Control:
  - Servo motors control barricades.
  - RFID-triggered actions activate servo motors to facilitate emergency vehicle passage.

Role of Arduino in the project:

Arduino MEGA:
- Controls traffic signals (LEDs) and barricades.
- Manages eight servo motors for barricade control.
- Handles timing and sequencing of traffic signals.
- Establishes inter-Arduino communication with Arduino UNO.
- Prioritizes emergency vehicles by adjusting traffic signals and barricades when needed.

Arduino UNO:
- Manages RFID readers for each road.
- Receives data from RFID readers regarding RFID tag presence on vehicles.- Communicates with Arduino MEGA in real-time for traffic signal and barricade control.
- Facilitates the efficient passage of emergency vehicles through the junction.

Both Arduino boards work together to create an intelligent traffic management system that optimizes
traffic flow, reduces congestion, and prioritizes emergency vehicle movement at the road junction.

Working:

The methodology for real-time traffic signal control in this project aims to minimize traffic congestion and prioritize emergency service vehicles. It involves the use of two Arduino circuit boards: the Uno board and the Mega board, each with its own algorithm. Here is a summary of the proposed methodology:

1. Arduino Uno Algorithm:

   - The Uno board is responsible for handling the RFID readers at roadside to detect and scan RFID cards placed on emergency vehicles.
   - The algorithm on the Uno board is designed to detect and scan an emergency vehicle's RFID card only if it arrives from the left lane of any road.
   - It utilizes the functions provided by the Arduino IDE for initialization and repeated execution phases.

2. Arduino Mega Algorithm:
   - The Mega board is tasked with controlling and managing the light-emitting diodes (LEDs) used as traffic signals and driving the servo motors that control barricades.

- It plays a central role in the traffic signal control system.
- The algorithm on the Mega board determines when to change traffic signals, raising or lowering barricades as needed.

2. Minimizing Traffic Congestion:
   - The primary objective of the methodology is to minimize traffic congestion by efficiently managing traffic signals and barricades.

4. Prioritizing Emergency Vehicles:
   - The system is designed to give priority to emergency service vehicles by allowing them to pass through the intersection with minimum delay.

5. Lane-Specific Detection:
   - Emergency vehicles are detected and scanned for RFID cards if they approach from the left lane of any road.

The system combines the functionalities of the Uno and Mega boards to create an intelligent traffic management solution. It uses RFID technology to identify emergency vehicles and controls traffic signals and barricades in real-time to optimize traffic flow and prioritize emergency responses.

DEVELOPMENT PART:

To build a complete traffic management system using RFID, Arduino Uno, and Arduino Mega is a complex task involving different components and hardware, so we are going with a simplified Python script for Arduino Uno and Mega collect RFID data and map traffic congestion Statistics. To fully implement this functionality, we need the necessary hardware components and libraries for the RFID connection.

Here is a basic Python script for Arduino Uno and Arduino Mega. This script reads the RFID data from the RFID sensor and plots the traffic jam data. To collect real RFID data, we need an RFID sensor and a library to communicate with.

The Arduino Uno python script is:

```
import serial
# Define the Arduino Uno's serial port.
uno_port = "COM3"  # Update this with your Uno's serial port.
try:
    uno_serial = serial.Serial(uno_port, 9600)
except serial.SerialException:
    print("Failed to open the Uno's serial port.")
    exit()
while True:
    uno_data = uno_serial.readline().strip().decode('utf-8')
    print("RFID data from Arduino Uno:", uno_data)
    # Simulate traffic congestion data (0-100, where 100 means heavy congestion).
    traffic_congestion = 50  # Change this value as needed.
    print("Traffic Congestion Level:", traffic_congestion)
    # Upload the RFID data and traffic congestion to a website or database.
    # You will need to implement the upload functionality.
# Close the serial connection when done.
uno_serial.close()
```

The Arduino Mega python script is:

```
import serial

# Define the Arduino Mega's serial port.
```

```
mega_port = "COM4"  # Update this with your Mega's serial port.
try:
    mega_serial = serial.Serial(mega_port, 9600)
except serial.SerialException:
    print("Failed to open the Mega's serial port.")
    exit()
while True:
    mega_data = mega_serial.readline().strip().decode('utf-8')
    print("RFID data from Arduino Mega:", mega_data)
    # Simulate traffic congestion data (0-100, where 100 means heavy congestion).
    traffic_congestion = 60  # Change this value as needed.
    print("Traffic Congestion Level:", traffic_congestion)
    # Upload the RFID data and traffic congestion to a website or database.
    # You will need to implement the upload functionality.
# Close the serial connection when done.
```

This code reads RFID data and maps the traffic congestion pattern. For this function to work, we need to add the following items.

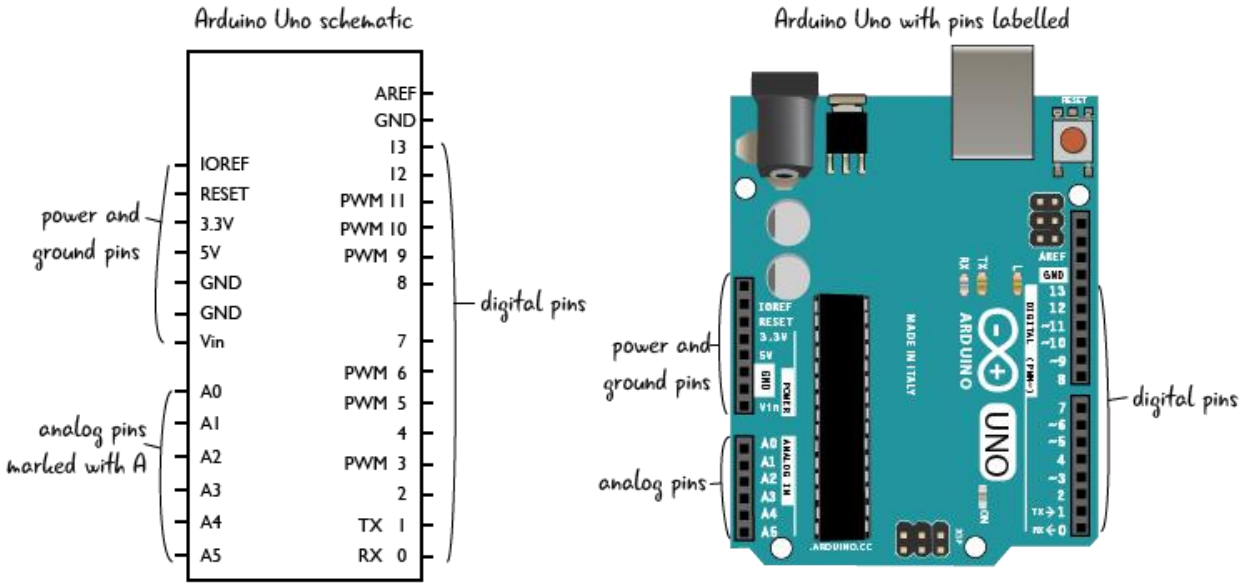
1) RFID readers and RFID tags.
2) Appropriate libraries and drivers for communication with RFID readers.
3) A web page or database that is loaded and displayed on stored data.
4) Proper Arduino code for communication with RFID readers.

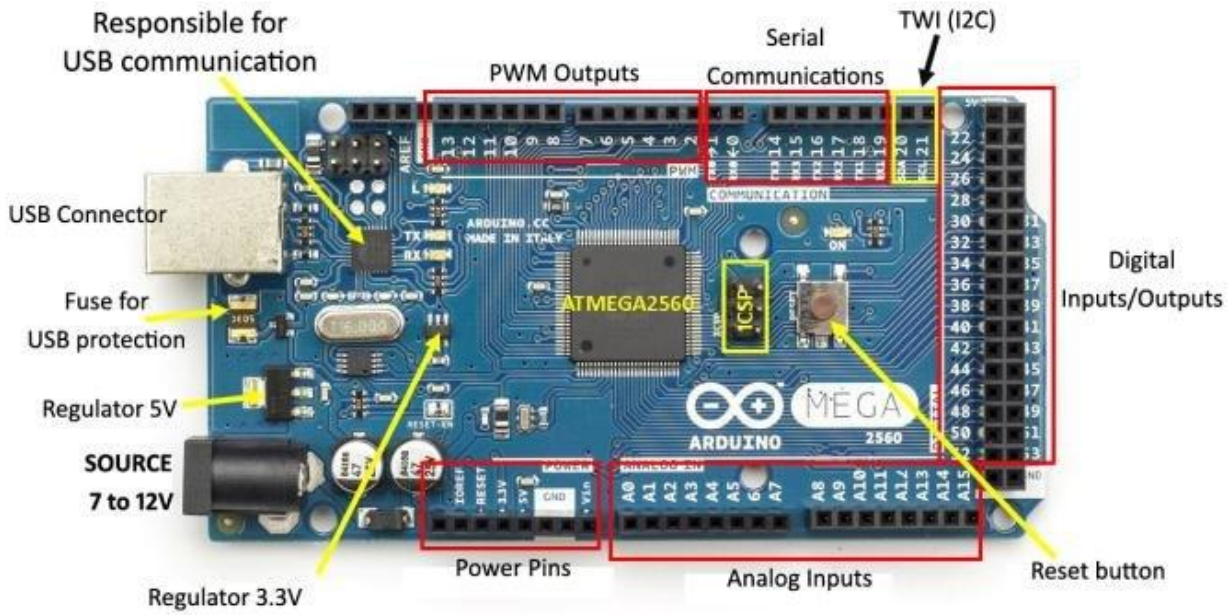
To send data to a specific website, we can use Python and the 'requests' library to make HTTP POST requests to a web server. In this example, we will see how to modify an Arduino Uno Python script to send data to a website. We need to have a server-side script to process the data and store it in a database on the web.

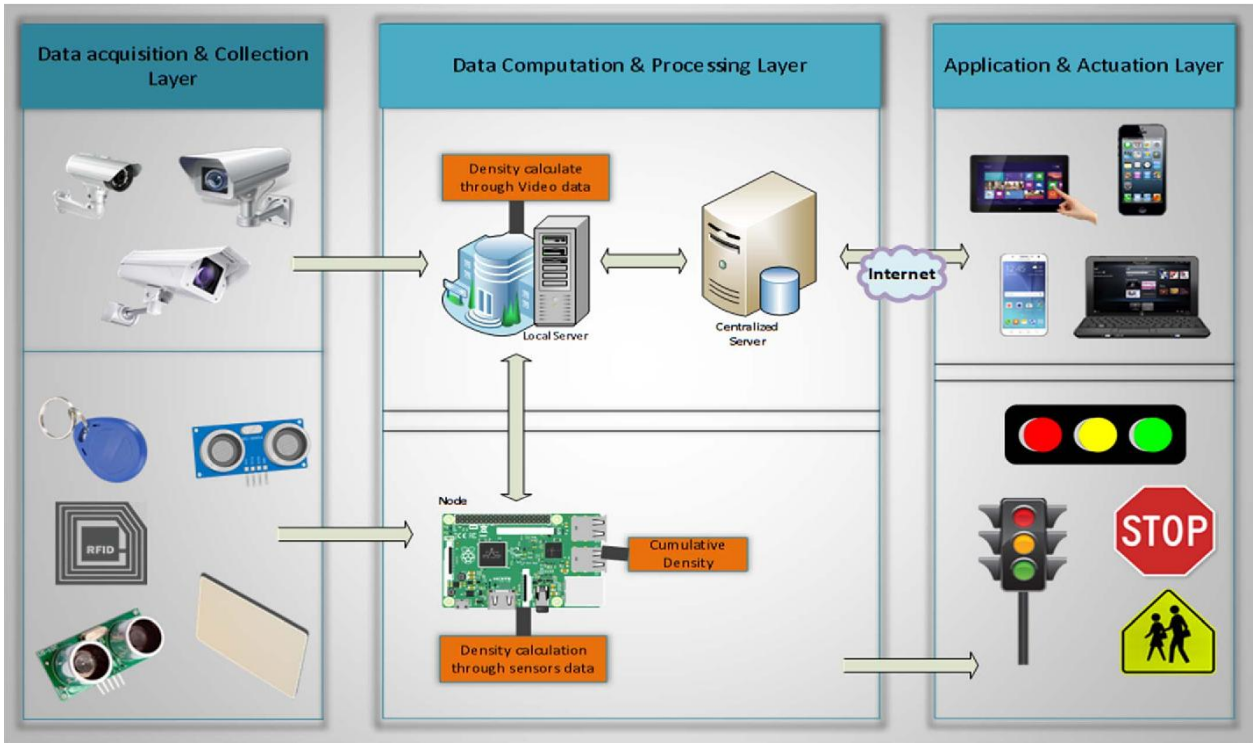
Here's how we can modify Arduino Uno python script to upload RFID and traffic congestion data to a website:

```
import serial
import requests
# Define the Arduino Uno's serial port and website URL.
uno_port = "COM3"  # Update this with your Uno's serial port.
website_url = "https://example.com/upload_traffic_data.php"  # Update with your website's URL.
try:
    uno_serial = serial.Serial(uno_port, 9600)


except serial.SerialException:
    print("Failed to open the Uno's serial port.")
    exit()
while True:
    uno_data = uno_serial.readline().strip().decode('utf-8')
    print("RFID data from Arduino Uno:", uno_data)
    # Simulate traffic congestion data (0-100, where 100 means heavy congestion).
    traffic_congestion = 50  # Change this value as needed.
    print("Traffic Congestion Level:", traffic_congestion)
    # Prepare the data to send to the website.
    data_to_send = {
        "rfid_data": uno_data,
        "traffic_congestion": str(traffic_congestion)
    }

    try:
        response = requests.post(website_url, data=data_to_send)
        if response.status_code == 200:
            print("Data uploaded successfully.")
        else:
            print("Failed to upload data. Status code:", response.status_code)
    except requests.exceptions.RequestException as e:
        print("Error:", e)
# Close the serial connection when done.
uno_serial.close()
```

In this code, we use the ‘requests.post()’ method to send RFID data and traffic congestion information to the website. The website URL must point to a server-side script (e.g., PHP) that can process the incoming data and store it in a database. We should replace "https://example.com/upload_traffic_data.php" with the actual URL of our website's data-checking script. In our website, we need to create a script that retrieves the data and inserts it into a database.







CONCLUSION:

To sum up, IoT in traffic management emerges not just as a technological advancement but as a promising solution to the issues of traffic congestion, safety concerns, and urban efficiency. By integrating sensors, automation, and data analysis, IoT traffic management offers the potential to transform our cities into more navigable, safer, and environmentally friendly spaces.