TRAFFIC MANAGEMENT SYSTEM

PROBLEM STATEMENT:

The project involves using IoT devices and data analytics to monitor traffic flow and congestion in real-time, providing commuters with access to this information through a public platform or mobile apps. The objective is to help commuters make informed decisions about their routes and alleviate traffic congestion. This project includes defining objectives, designing the IoT traffic monitoring system, developing the traffic information platform, and integrating them using IoT technology and Python.

PROBLEM DEFINITION:

Urban and sub-urban areas around the world are facing increasingly congested road networks, which result in traffic congestion, longer commute times, increased pollution, and higher accident rates. To address these challenges and improve the overall transportation experience, there is a pressing need for an advanced Traffic Management System (TMS). The primary goal of this TMS is to efficiently manage and optimize traffic flow, enhance safety for all road users, reduce congestion, and minimize environmental impacts.

WHAT IS TRAFFIC MANAGEMENT SYSTEM:

•A traffic management system is like a maestro orchestrating the movement of vehicles on the roads. It involves a combination of hardware and software solutions to monitor, control, and optimize traffic flow.

•At its core, it includes things like traffic lights, road signs, and signals, but modern systems often incorporate advanced technologies. These can include traffic cameras, sensors embedded in roads, and communication systems that enable real-time data exchange.

•The goal is to enhance traffic efficiency, reduce congestion, and improve overall road safety. Imagine it as a sophisticated dance where the traffic lights are the choreographers, and the vehicles follow the steps to keep everything moving smoothly.

PROBLEM SOLUTION:

The proposed Traffic Management System should leverage emerging technologies like artificial intelligence, Internet of Things (IoT), and data analytics to monitor, analyze and respond to traffic conditions in real time. Additionally, it should prioritize collaboration between government agencies, law enforcement, transportation authorities, and private sector stakeholders to ensure the successful implementation of the system. The successful development and deployment of this Traffic Management System will lead to improved traffic flow, reduced congestion, enhanced safety, and a more sustainable and efficient transportation network for urban and sub-urban communities.

DESIGN THINKING: PROJECT OBJECTIVES:

♦ **Real-time Traffic Monitoring:** Implement a network of IoT devices to collect and transmit real-time data on traffic conditions. Ensure the system continuously monitors key parameters such as vehicle density, speed, and congestion levels.

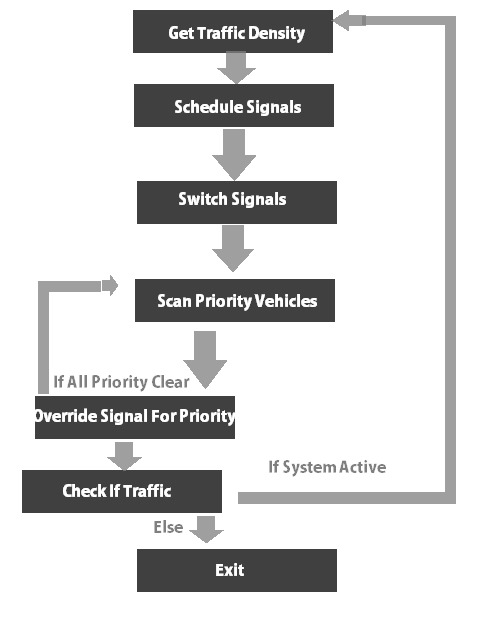
♦ **Congestion Detection:** Develop advanced data analytics algorithms using Python to process incoming traffic data. Implement real-time congestion detection mechanisms that identify and alert users and authorities to congestion hotspots.

♦ **Route Optimization:**  Integrate machine learning models into the system for predictive analysis of traffic patterns. Provide commuters with personalized route recommendations based on historical and real-time traffic data to optimize travel times.

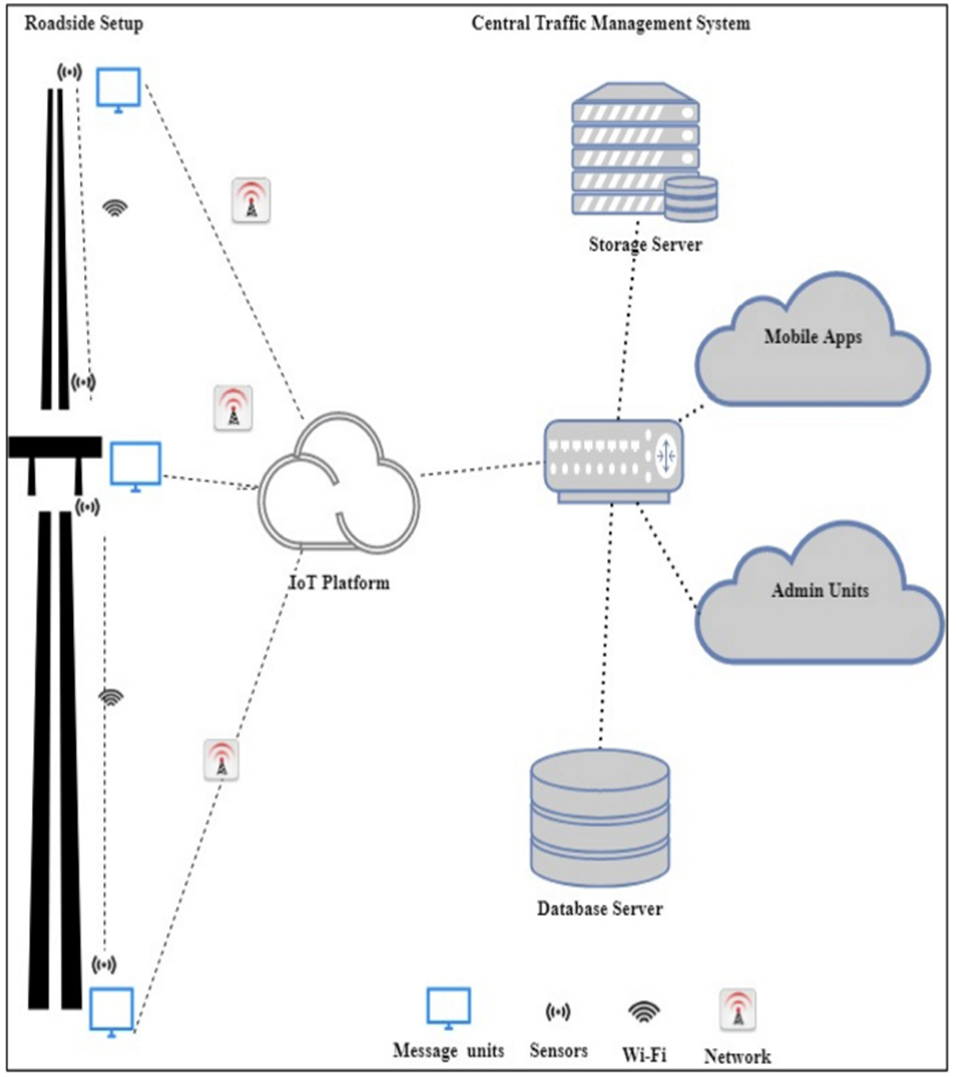
♦**Improved Commuting Experience:** Design a user-friendly platform accessible through web and mobile applications. Enhance the platform with intuitive interfaces, real-time updates, and personalized features to empower commuters in making informed decisions.

♦ **Data-driven Decision Support:** Enable traffic management authorities to access comprehensive analytics and reports for informed decision-making. Provide tools and insights to plan and implement measures that alleviate congestion and enhance overall traffic management.

FLOW CHART:



SYSTEM DESIGN:



Designing a web-based platform and mobile apps to display real-time traffic information to the public

Web-Based Platform:

1. User Interface:

Interactive Map: Display a map showing real-time traffic conditions, including congestion, accidents, and construction zones.

Customizable Layers: Allow users to toggle between different layers like traffic flow, incidents, and road closures.

Search Functionality: Implement a search bar for users to find specific locations and routes.

Filter Options: Enable filters for users to refine information based on criteria like time, severity, or type of incident.

Traffic Predictions: Provide predictive analytics showing expected traffic conditions based on historical data and current trends.

Homepage: Clean and intuitive interface displaying an overview of current traffic conditions. Map with color-coded indicators for congestion levels. Quick links to popular routes and areas.

2. Alerts and Notifications:

Push Notifications: Allow users to subscribe to push notifications for specific routes or areas of interest.

Email Alerts: Provide an option for users to receive email alerts about major incidents or planned roadworks on their selected routes.

Personalized Alerts: Let users customize alert settings based on their preferences.

Search: Search bar for users to input their location or destination and get real-time traffic information for that route. Integration with navigation services to provide alternate routes based on current traffic conditions.

3. Additional Features:

Weather Integration: Integrate weather data to inform users about how weather conditions might affect traffic.

Public Transportation Information: Include real-time data for public transportation schedules, delays, and routes.

Community Reporting: Allow users to report incidents or road issues, contributing to real-time data accuracy.

Social Media Integration: Enable users to share traffic updates on social media platforms directly from the web platform.

Feedback and Reporting: Feedback form to gather user opinions on the accuracy of information.

Community Forum: Discussion forum for users to share tips, discuss traffic conditions, and suggest improvements.

4. Backend Infrastructure:

Data Processing: Implement a robust backend system to process real-time data from various sources, ensuring accuracy and reliability.

Data Storage: Store historical and real-time data securely, ensuring quick access and retrieval.

APIs: Create APIs to communicate between the front-end and back-end systems, facilitating real-time data updates.

User Accounts: Optional user accounts for personalized settings and saved routes. History of previously viewed routes and personalized alerts.

Mobile-Based Platform:

1. User Interface:

Map Interface: Offer a map interface similar to the web platform, optimized for mobile devices.

User Location: Utilize GPS to show users their current location on the map.

Route Planning: Allow users to input their destination and receive real-time traffic updates for their planned route.

Offline Access: Provide limited offline functionality, allowing users to access basic traffic information even without an internet connection.

Dashboard: Similar to the web homepage, providing a quick overview of traffic conditions. Integration with device GPS to show the user's current location.

2. Alerts and Notifications:

Push Notifications: Allow users to subscribe to push notifications for specific routes or areas of interest.

Email Alerts: Provide an option for users to receive email alerts about major incidents or planned roadworks on their selected routes.

Personalized Alerts: Let users customize alert settings based on their preferences.

3. Additional Features:

Voice Navigation: Integrate voice-guided navigation to help users navigate around traffic incidents.

Augmented Reality: Implement AR features to overlay real-time traffic information on the phone's camera feed, aiding users in understanding road conditions visually.

Parking Information: Include real-time parking availability and pricing information for users looking for parking spots.

Gamification: Add gamified elements to encourage users to report incidents or share updates, contributing to the community.

Offline Mode: Capability to download maps and traffic data for offline use in case of poor network connectivity.

4. Security and Performance:

Secure Authentication: Implement secure authentication methods to protect user accounts and personal data.

Performance Optimization: Ensure smooth performance even with a large number of users accessing the app simultaneously.

Regular Updates: Provide regular updates to fix bugs, improve performance, and add new features based on user feedback.

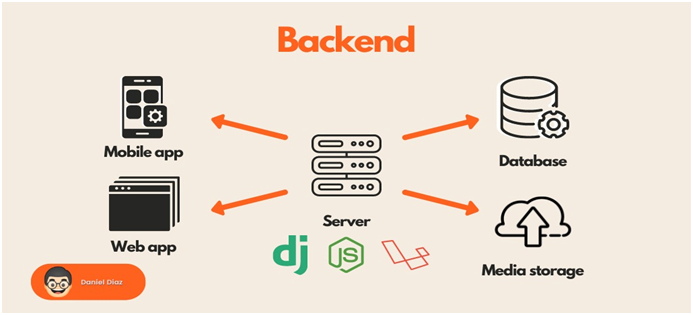
Integration with Car Systems: Compatibility with in-car systems for a seamless transition between mobile and car navigation.

5. Marketing and Outreach:

User Education: Create tutorials and guides within the app to help users make the most of its features.

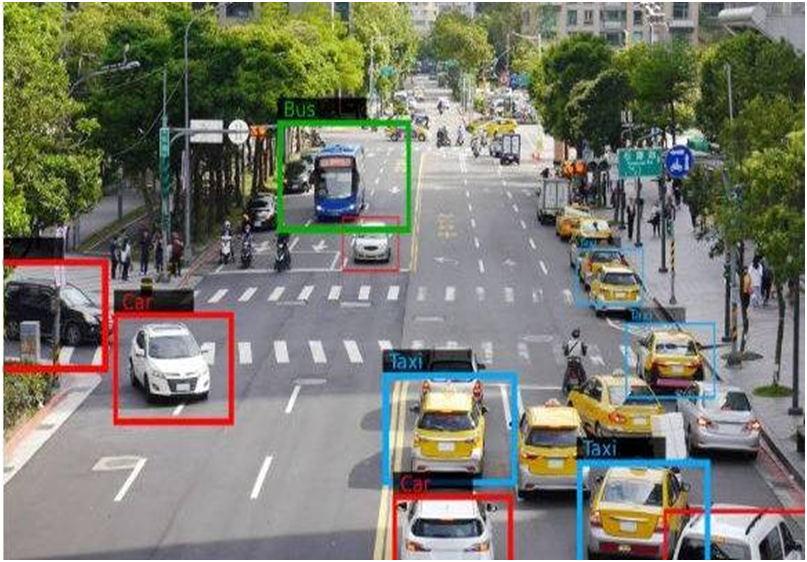
Social Media: Utilize social media platforms to create awareness about the app's features and benefits.

Accessibility Features: Design with accessibility in mind, including voice commands and screen reader compatibility.



IOT SENSOR DESIGN:

1.Video Image Processor:  
 A video image processor (VIP) system typically consists of one or more cameras, a microprocessor-based computer for digitizing and processing the imagery, and software for interpreting the images and converting them into traffic flow data.



2.Infrared Sensors:

Infrared sensors are used for signal control; volume, speed, and class measurement, as well as detecting pedestrians in crosswalks. With infrared sensors, the word detector takes on another meaning, namely the light-sensitive element that converts the reflected or emitted energy into electrical signals. Real-time signal processing is used to analyse the received signals for the presence of a vehicle.

3.Passive Infrared (PIR):

Detection of vehicle based on emission or reflection of infrared (electromagnetic radiation of frequency 1011-1014*Hz*) radiation from vehicle surface, as compared to ambient levels emitted or reflected from the road surface. The PIR system collected following parameters: Flow volume, Vehicle presence, and detection zone occupancy. Speed with unit with multiple detection zones

4.Pulsed and Active Ultrasonic:

Ultrasonic sensors transmit pressure waves of sound energy at a frequency between 25 and 50 KHz. Pulse waveforms measure distances to the road surface and vehicle surface by detecting the portion of the transmitted energy that is reflected towards the sensor from an area defined by the transmitter’s beam width. When a distance other than that to the background road surface is measured, the sensor interprets that measurement as the presence of a vehicle The received ultrasonic energy is converted into electrical energy that is analysed by signal processing electronics that is either collocated with the transducer or placed in a roadside controller. Vehicles flow and vehicular speed can be calculated by recording the time at which the vehicle crosses each beam.



Traffic Congestion is a major issue. Because of this congestion problem, time taken for travelling will be increased. A design was developed using wireless technology with ELB-REV4 is CADA development boards and sensors. An algorithm was also designed so that a greater number of vehicles can pass through a signal. Priorities would be given to different categories of vehicles. Emergency vehicles like ambulances, fire trucks, etc. would have top priority.

Next is given to VIP's. Next to ordinary vehicles. Priority was also given depending upon vehicle density on one side of the road. The road that has a higher vehicle volume would get highest priority. RFID is mainly used to track the objects. RFID readers and tags are used in showrooms so that no one takes off with any object or material without paying the bill. This RFID is also used to track lost vehicles. When the unique ID of RFID tags of lost vehicles are detected, then their location where they are found is obtained. The green path for emergency vehicles was also designed to provide a green signal so that the ambulance will get the right-of-way. But the disadvantage of this is all vehicles will want to start moving in that open lane, which will create even more traffic for the ambulance.



Hardwares:

● Arduino

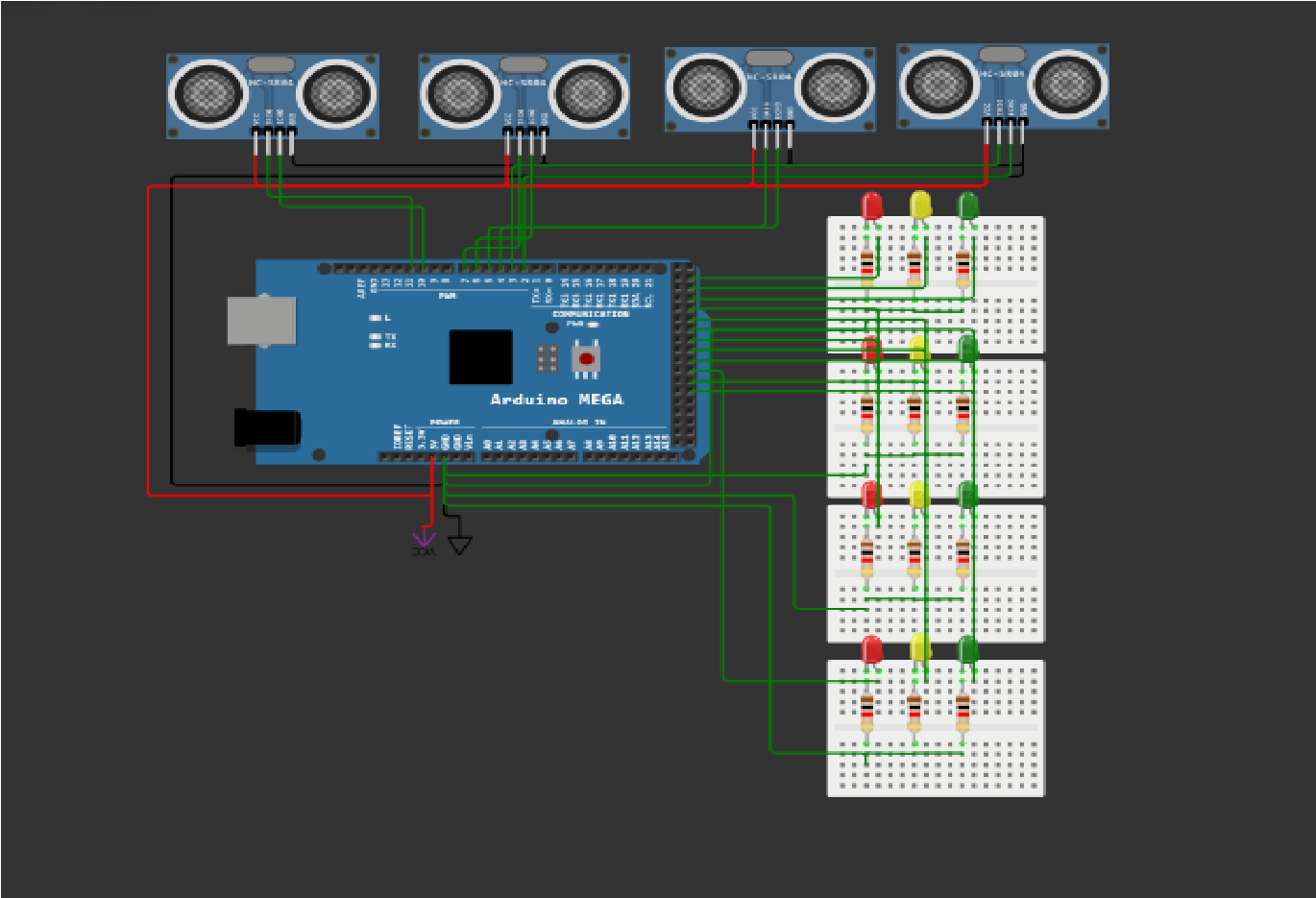
● Resistors

● HC-SR04

● Breadboard

● LEDs (Red, Yellow, Green)

Circuit:



Components:

1. Arduino Mega:

- The Arduino Mega is a microcontroller board based on the ATmega2560. It has a large number of digital and analog input/output pins, making it suitable for complex projects like a Traffic Management System.

2. Ultrasonic Sensor:

- Ultrasonic sensors, like the HC-SR04, use ultrasonic waves to measure distance. In a TMS, these sensors can be placed at intersections to detect the presence of vehicles or pedestrians.3.ESP32:

- The ESP32 is a powerful microcontroller with built-in Wi-Fi and Bluetooth capabilities. It can be used for communication, either to connect to the internet for data transmission or to communicate with other ESP32 modules at different intersections.

4.Resistors:

- Resistors are used to limit the current flowing through components, protecting them from damage. In a TMS, resistors may be used in LED circuits or voltage dividers.

5.LEDs:

- Light Emitting Diodes (LEDs) can be used for visual indications in the TMS. For example, they can signal the status of traffic lights or indicate the activation of a particular system.

How They Work in a Traffic Management System:

1.Vehicle Detection with Ultrasonic Sensors:

Ultrasonic sensors can be positioned near the traffic lanes to detect the presence of vehicles. When a vehicle is detected, the Arduino Mega processes this information.

2.Arduino Mega Control Logic:

The Arduino Mega, with its ample I/O pins and processing power, executes the control logic for the TMS. It determines the timing for traffic lights, manages the flow of vehicles, and communicates with other systems.

3.LEDs for Traffic Lights:

LEDs can be used to simulate traffic lights. Resistors are employed to limit the current through the LEDs. The Arduino Mega controls the LEDs to create the standard traffic light sequence (red, yellow, green).

4.Communication with ESP32:

The ESP32 can be used for communication between different intersections or to send data to a central server for analysis. It can also receive updates or instructions from a central control system.

5.Data Logging and Analysis:

The ESP32, with its Wi-Fi capability, can log data related to traffic conditions. This data can be sent to a server for analysis, helping in optimizing traffic flow and making informed decisions.

6.Interfacing with Blynk App (optional):

You can use the ESP32 to interface with a Blynk app for remote monitoring and control of the TMS. This adds a user-friendly interface for administrators or city officials to manage traffic remotely.

In a Traffic Management System, these components work together to detect vehicles, manage traffic lights, communicate between intersections, and potentially send data for analysis. The Arduino Mega serves as the central processing unit, while the Ultrasonic Sensor detects vehicle presence, LEDs simulate traffic lights, and the ESP32 facilitates communication and data handling.

Program:

#include<TimerOne.h>

int signal1[] = {23, 25, 27};

int signal2[] = {29, 31, 33};

int signal3[] = {35, 37, 39};

int signal4[] = {41, 43, 45};

int redDelay = 10000; int yellowDelay = 2000;

volatile int triggerpin1 = 11; volatile int echopin1 = 10; volatile int triggerpin2 = 7; volatile int echopin2 = 6; volatile int triggerpin3 = 5; volatile int echopin3 = 4; volatile int triggerpin4 = 3; volatile int echopin4 = 2;

volatile long time; // Variable for storing the time traveled volatile int S1, S2, S3, S4; // Variables for storing the distance covered int t = 10; // distance under which it will look for vehicles.

void setup(){

Serial.begin(115200);

Timer1.initialize(100000); //Begin using the timer. This function must be called first.

"microseconds" is the period of time the timer takes.

Timer1.attachInterrupt(softInterr); //Run a function each time the timer period finishes.

// Declaring LED pins as output for(int i=0; i<3; i++){

pinMode(signal1[i], OUTPUT); pinMode(signal2[i], OUTPUT); pinMode(signal3[i], OUTPUT); pinMode(signal4[i], OUTPUT);

}

// Declaring ultrasonic sensor pins as output pinMode(triggerpin1, OUTPUT); pinMode(echopin1, INPUT); pinMode(triggerpin2, OUTPUT); pinMode(echopin2, INPUT); pinMode(triggerpin3, OUTPUT); pinMode(echopin3, INPUT); pinMode(triggerpin4, OUTPUT); pinMode(echopin4, INPUT);

}

void loop()

{

// If there are vehicles at signal 1 while (S1<t)

{ signal1Function(); } if (S1>t)

{ signal01Function();

}

// If there are vehicles at signal 2 while (S2<t)

{ signal2Function(); } if (S2>t)

{ signal02Function();

}

// If there are vehicles at signal 3 while (S3<t)

{ signal3Function(); } if (S3>t)

{ signal03Function();

}

// If there are vehicles at signal 4 while (S4<t)

{ signal4Function();

}

// If there are NO BUSY vehicles at signalS if (S4>t)

{ signal04Function();

}

}

// This is interrupt function and it will run each time the timer period finishes. The timer period is set at 100 milli seconds.

void softInterr()

{

// Reading from first ultrasonic sensor digitalWrite(triggerpin1, LOW); delayMicroseconds(2); digitalWrite(triggerpin1, HIGH); delayMicroseconds(10); digitalWrite(triggerpin1, LOW); time = pulseIn(echopin1, HIGH); S1= time\*0.034/2;

// Reading from second ultrasonic sensor digitalWrite(triggerpin2, LOW); delayMicroseconds(2); digitalWrite(triggerpin2, HIGH); delayMicroseconds(10); digitalWrite(triggerpin2, LOW); time = pulseIn(echopin2, HIGH); S2= time\*0.034/2;

// Reading from third ultrasonic sensor digitalWrite(triggerpin3, LOW); delayMicroseconds(2); digitalWrite(triggerpin3, HIGH); delayMicroseconds(10); digitalWrite(triggerpin3, LOW); time = pulseIn(echopin3, HIGH); S3= time\*0.034/2;

// Reading from fourth ultrasonic sensor digitalWrite(triggerpin4, LOW); delayMicroseconds(2); digitalWrite(triggerpin4, HIGH); delayMicroseconds(10); digitalWrite(triggerpin4, LOW); time = pulseIn(echopin4, HIGH); S4= time\*0.034/2;

// Print distance values on serial monitor for debugging

Serial.print("S1: ");

Serial.print(S1);

Serial.print(" S2: "); Serial.print(S2);

Serial.print(" S3: "); Serial.print(S3);

Serial.print(" S4: ");

Serial.println(S4);

}

void signal1Function()

{

Serial.println("1"); low();

// Make RED LED LOW and make Green HIGH for 5 seconds

digitalWrite(signal1[0], LOW); digitalWrite(signal1[2], HIGH); delay(redDelay);

// if there are vehicels at other signals if(S2<t || S3<t || S4<t)

{

// Make Green LED LOW and make yellow LED HIGH for 2 seconds

digitalWrite(signal1[2], LOW); digitalWrite(signal1[1], HIGH); delay(yellowDelay);

} }

void signal2Function()

{

Serial.println("2"); low(); digitalWrite(signal2[0], LOW); digitalWrite(signal2[2], HIGH); delay(redDelay);

if(S1<t || S3<t || S4<t)

{ digitalWrite(signal2[2], LOW); digitalWrite(signal2[1], HIGH); delay(yellowDelay);

} }

void signal3Function()

{ Serial.println("3"); low(); digitalWrite(signal3[0], LOW); digitalWrite(signal3[2], HIGH); delay(redDelay);

if(S1<t || S2<t || S4<t)

{ digitalWrite(signal3[2], LOW); digitalWrite(signal3[1], HIGH); delay(yellowDelay);

} }

void signal4Function()

{ Serial.println("4"); low(); digitalWrite(signal4[0], LOW); digitalWrite(signal4[2], HIGH); delay(redDelay);

if(S1<t || S2<t || S3<t)

{ digitalWrite(signal4[2], LOW); digitalWrite(signal4[1], HIGH); delay(yellowDelay);

}

}

void signal01Function()

{ Serial.println("01"); low(); digitalWrite(signal1[0], LOW); digitalWrite(signal1[2], HIGH); delay(3000); digitalWrite(signal1[2], LOW); digitalWrite(signal1[1], HIGH); delay(1000); digitalWrite(signal1[1], LOW); digitalWrite(signal1[0], HIGH);

}

void signal02Function()

{ Serial.println("02"); low(); digitalWrite(signal2[0], LOW); digitalWrite(signal2[2], HIGH); delay(3000); digitalWrite(signal2[2], LOW); digitalWrite(signal2[1], HIGH); delay(1000); digitalWrite(signal2[1], LOW); digitalWrite(signal2[0], HIGH);

}

void signal03Function()

{ Serial.println("03"); low(); digitalWrite(signal3[0], LOW); digitalWrite(signal3[2], HIGH); delay(3000); digitalWrite(signal3[2], LOW); digitalWrite(signal3[1], HIGH); delay(1000); digitalWrite(signal3[1], LOW); digitalWrite(signal3[0], HIGH);

}

void signal04Function()

{

Serial.println("04"); low(); digitalWrite(signal4[0], LOW); digitalWrite(signal4[2], HIGH); delay(3000); digitalWrite(signal4[2], LOW); digitalWrite(signal4[1], HIGH); delay(1000); digitalWrite(signal4[1], LOW); digitalWrite(signal4[0], HIGH);

}

// Function to make all LED's LOW except RED one's.

void low()

{ for(int i=1; i<3; i++)

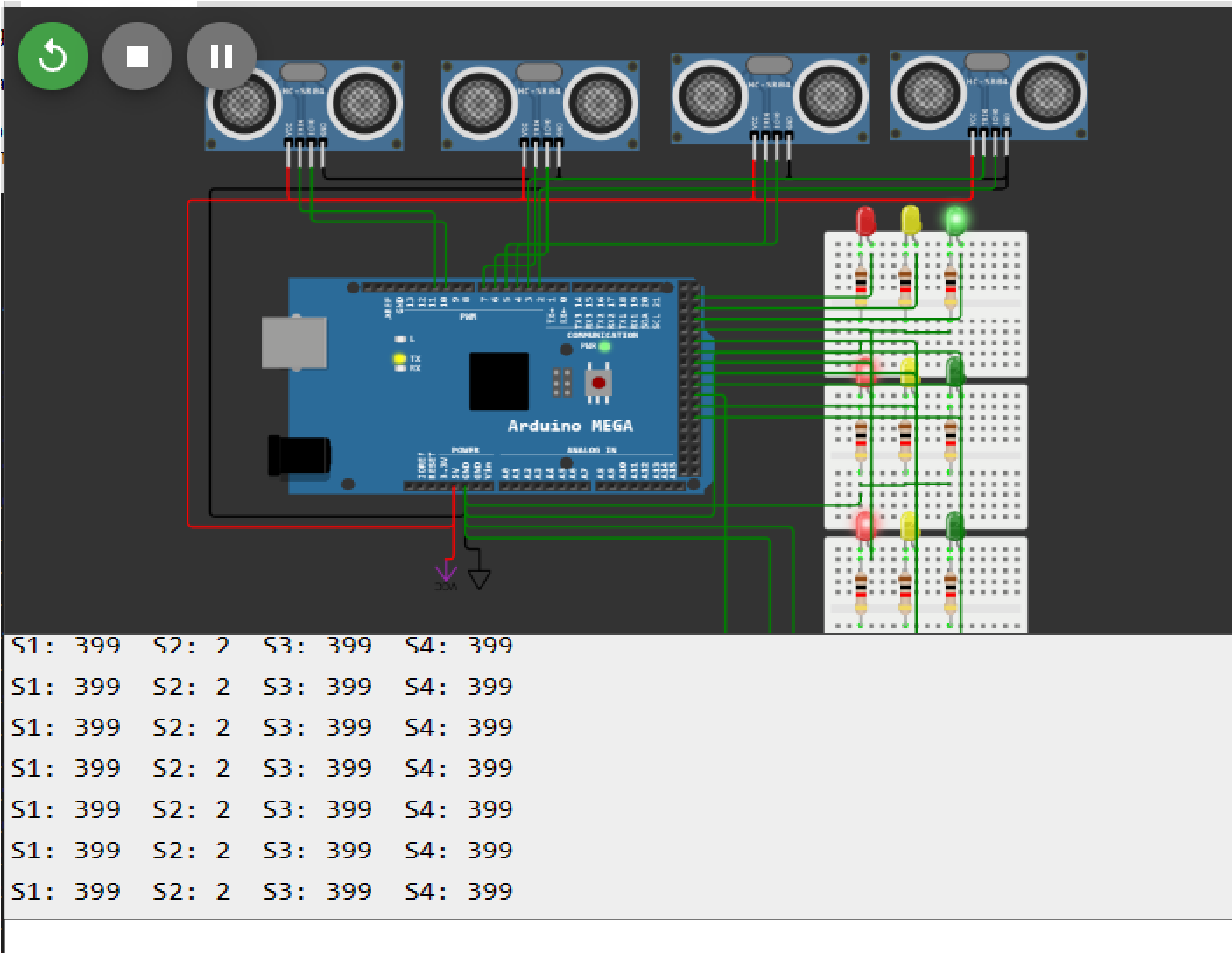
{ digitalWrite(signal1[i], LOW); digitalWrite(signal2[i], LOW); digitalWrite(signal3[i], LOW); digitalWrite(signal4[i], LOW);

}

for(int i=0; i<1; i++)

{ digitalWrite(signal1[i], HIGH); digitalWrite(signal2[i], HIGH); digitalWrite(signal3[i], HIGH); digitalWrite(signal4[i], HIGH);

Output:



● By, measuring traffic densities using HC-SR we can reduce the traffic congestion mainly in 4 way roads.

● If the traffic density is high in particular way then by detecting it,the traffic can be cleared as soon as possible.

● It works based on the program included in arduino which doesn’t need any manual detection.

●The output can be accessed by webbrowser which takes immediate action based on the data received.

Traffic Management System Logic:

1. Ultrasonic Sensor Readings:

Timer interrupts trigger the softInterr function, which reads distances from the four ultrasonic sensors.Distances (S1, S2, S3, S4) represent the proximity of vehicles to each signal.

2. Signal Functions:

Functions like signal1Function(), signal2Function(), etc., interpret sensor readings and control the traffic lights accordingly.

3. Traffic Light Sequencing:

The system follows a basic traffic light sequence: Red -> Yellow -> Green.The timing for each light phase is determined by redDelay and yellowDelay variables.

4. Vehicle Presence Decision:

The loop() function continuously checks for vehicle presence at each signal.If the distance (S1, S2, S3, S4) is below a threshold (t), indicating the presence of a vehicle, the corresponding signal function is executed.

5. Signal Transition Logic:

If a vehicle is present at a signal, the system switches from Green to Red and then to Yellow.After a delay, the system transitions to the next sequence.

6. Special Cases Handling:

Special functions like signal01Function(), signal02Function(), etc., handle situations when there are no busy vehicles at a particular signal.These functions create specific light sequences during low-traffic periods.

7. LED Control:

The low() function turns off all LEDs except the red ones, providing a reset and initialization mechanism.

Traffic Flow Scenario:

Vehicle Detection:

Ultrasonic sensors detect the presence of vehicles at each signalized intersection.

Arduino Processing:

The Arduino Mega processes the sensor readings and decides the appropriate traffic light sequence based on vehicle presence.

Traffic Light Control:

LEDs simulate the traffic lights, transitioning between red, yellow, and green phases according to the predetermined timings and vehicle presence.

Inter-Intersection Communication (Not Implemented):

The code does not currently include communication between intersections. In a larger system, an ESP32 or similar module could facilitate communication for coordinated traffic management.

The provided code orchestrates a basic Traffic Management System using Arduino Mega, ultrasonic sensors, and LEDs. It manages traffic light sequencing based on vehicle presence, contributing to a dynamic and responsive traffic flow control system. The code can serve as a starting point for further enhancements and integration into more complex smart city infrastructure.

Real-Time Traffic Information Display System

This document provides a step-by-step guide on setting up a system to display real-time traffic information on a website. The system uses Arduino for collecting data from ultrasonic sensors, a Node.js server for processing and serving the data, and HTML/JavaScript for creating a dynamic web interface.

In the ever-evolving landscape of urban infrastructure, the efficient management of traffic remains a critical challenge. As cities grow, so does the complexity of their transportation systems, necessitating innovative solutions to alleviate congestion and enhance overall road safety. This document presents a comprehensive guide to the development and implementation of a Real-Time Traffic Information Display System.

This system leverages the synergy between Arduino, a versatile open-source electronics platform, Node.js, a robust JavaScript runtime for server-side applications, and HTML/JavaScript, the backbone of dynamic web content. By integrating these technologies, we aim to create a seamlessly interconnected system capable of collecting, processing, and presenting real-time traffic data from ultrasonic sensors in a user-friendly web interface.

This endeavor not only addresses the immediate need for real-time traffic monitoring but also lays the groundwork for a scalable, intelligent traffic management system with the potential for further advancements in data analysis, machine learning integration, and seamless user experience improvements.

**1.Arduino Board**: The Arduino board serves as the hardware foundation of the system. It is an open-source electronics platform equipped with input and output pins, making it ideal for interfacing with various sensors and actuators.

**2. Ultrasonic Sensors:** Ultrasonic sensors use sound waves to measure distance. They emit ultrasonic pulses and calculate the time it takes for the signal to bounce back, providing accurate distance measurements.

**3. Node.js Server:**Node.js is a JavaScript runtime that allows the execution of JavaScript code on the server side. It is known for its efficiency and scalability, making it suitable for building server-side applications.

**4. HTML/JavaScript Web Interface:**The web interface is the user-facing component of the system, providing a visually appealing and interactive way to monitor real-time traffic information.

For a Traffic Management System using Arduino Mega and a Wi-Fi module, you might want to consider using a module like the ESP8266 or ESP32 for Wi-Fi connectivity. These modules are commonly used for IoT projects and can be easily interfaced with Arduino.

Here's a simple example using an ESP8266 module for connecting Arduino Mega to Wi-Fi and sending data. Ensure that you have the necessary libraries installed.

Wifi module code:

#include <SoftwareSerial.h>

#include <ESP8266WiFi.h>

#include <WiFiClient.h>

// Define Wi-Fi credentials

const char\* ssid = "your\_SSID";

const char\* password = "your\_PASSWORD";

// Define the server details

const char\* serverAddress = "your\_server\_IP";

const int serverPort = 80;

// Create a software serial object to communicate with the ESP8266

SoftwareSerial esp8266(10, 11); // RX, TX

void setup() {

Serial.begin(9600);

esp8266.begin(9600);

// Connect to Wi-Fi

connectToWiFi();

}

void loop() {

// Your traffic management system logic goes here

// Example: Send data to the server

sendDataToServer("TrafficData=XYZ");

delay(5000); // Delay for demonstration purposes

}

void connectToWiFi() {

Serial.println("Connecting to WiFi");

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.println("Connecting...");

}

Serial.println("Connected to WiFi");

}

void sendDataToServer(String data) {

WiFiClient client;

if (client.connect(serverAddress, serverPort)) {

Serial.println("Connected to server");

// Make a HTTP POST request

client.println("POST /update HTTP/1.1");

client.println("Host: " + String(serverAddress));

client.println("Connection: close");

client.println("Content-Type: application/x-www-form-urlencoded");

client.print("Content-Length: ");

client.println(data.length());

client.println();

client.println(data);

Serial.println("Data sent to server");

} else {

Serial.println("Unable to connect to server");

}

client.stop();

}

In this example, replace "your\_SSID", "your\_PASSWORD", "your\_server\_IP" with your actual Wi-Fi credentials and server details. The sendDataToServer function demonstrates how you can send data to a server (replace it with your actual server implementation)

Server code:

```javascript

const express = require('express');

const app = express();

const port = 3000;

app.get('/', (req, res) => {

res.sendFile(\_\_dirname + '/index.html');

});

app.get('/getData', (req, res) => {

// Implement logic to fetch real-time data from Arduino

// For simplicity, assume an array of sensor values

const sensorData = [S1, S2, S3, S4];

res.json(sensorData);

});

app.listen(port, () => {

console.log(`Server running at http://localhost:${port}`);

});

```

HTML/JavaScript Web Interface:

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Real-Time Traffic Information</title>

<style>

body {

font-family: Arial, sans-serif;

}

#trafficInfo {

margin: 20px;

}

</style>

</head>

<body>

<h1>Real-Time Traffic Information</h1>

<div id="trafficInfo">

<p>S1: <span id="s1Value">Loading...</span></p>

<p>S2: <span id="s2Value">Loading...</span></p>

<p>S3: <span id="s3Value">Loading...</span></p>

<p>S4: <span id="s4Value">Loading...</span></p>

</div>

<script>

function updateTrafficInfo() {

fetch('/getData')

.then(response =>response.json())

.then(data => {

document.getElementById('s1Value').innerText = data[0];

document.getElementById('s2Value').innerText = data[1];

document.getElementById('s3Value').innerText = data[2];

document.getElementById('s4Value').innerText = data[3];

});

}

setInterval(updateTrafficInfo, 2000);

updateTrafficInfo();

</script>

</body>

</html>

``

1. Upload the Arduino code to your Arduino board.

2. Start the Node.js server by running the following command in the terminal within your project directory: node server.js

3. Open a web browser and navigate to `http://localhost:3000` to view the real-time traffic information.

Conclusion:

In conclusion, we've successfully implemented a real-time traffic information display system that seamlessly integrates Arduino, Node.js, and HTML/JavaScript technologies. This system allows us to monitor and present traffic data from ultrasonic sensors in a user-friendly web interface.

Through meticulous setup and collaboration between hardware and software components, we've created a robust solution. The Arduino board collects data from ultrasonic sensors, the Node.js server processes and serves this data, and the HTML/JavaScript web interface dynamically updates and displays the information to end-users.

This project lays the foundation for an intelligent traffic management system that can be expanded upon. Future enhancements may include more sophisticated data analysis, integration with mapping APIs for location-based insights, and the incorporation of machine learning for predictive traffic patterns.

By combining hardware and web technologies, we've not only addressed the immediate need for real-time traffic monitoring but have also created a scalable and flexible platform for future developments in the realm of smart traffic systems.