

Summer Internship Project Report

Ansuman Panigrahi

SIP09082002

Under the guidance of

Prof. Santos Kumar Das

July 7, 2024

**Abstract**

The dynamic traffic light control system using an ESP32 microcontroller aims to enhance urban traffic management by adapting traffic light signals based on real-time vehicle density at intersections. Traditional traffic light systems, which operate on fixed timing schedules, often lead to inefficiencies such as increased waiting times, higher fuel consumption, and elevated emissions, especially during fluctuating traffic conditions. This project addresses these inefficiencies by implementing a dynamic control mechanism that optimizes traffic flow through real-time data analysis.

Leveraging the powerful features of the ESP32, including its dual-core processor and WiFi connectivity, the system monitors vehicle density using sensor inputs and adjusts the duration of traffic signals accordingly. The ESP32 also serves as a web server, providing a user-friendly web interface for remote monitoring and control. Users can switch between time-based and density-based control modes, view the current status of each traffic light, and adjust system parameters dynamically.

The system operates in two main modes: time-based control, which follows predefined signal durations, and density-based control, which adapts signal timings based on real-time vehicle counts at each intersection. This flexibility allows for better traffic flow management during peak and off-peak hours, reducing congestion and improving overall traffic efficiency.

In addition to hardware components such as LEDs for traffic lights and resistors for current limiting, the system's software components include the Arduino IDE for programming, and HTML, CSS, and JavaScript for the web interface. The project demonstrates the potential of IoT and smart technologies in solving real-world problems, paving the way for more intelligent and responsive urban infrastructure.

The dynamic traffic light control system can be a crucial component in modern smart cities, offering scalable and adaptable traffic management solutions. Its applications extend to emergency response scenarios, where traffic lights can be controlled to clear paths for emergency vehicles, and research and development in traffic management systems. By reducing waiting times and fuel consumption, the system not only improves traffic flow but also contributes to environmental sustainability.

## **Introduction**

The objective of this project is to design and implement a dynamic traffic light control system using an ESP32 microcontroller. The system aims to optimize traffic flow by adjusting traffic light timings based on the vehicle density at four intersections. This approach not only improves traffic management but also reduces waiting time and fuel consumption. By utilizing the capabilities of the ESP32, particularly its WiFi features, we can create a robust and flexible system that can be monitored and controlled remotely through a web interface.

Traditional traffic light systems operate on fixed timings regardless of traffic conditions, leading to inefficiencies, particularly during off-peak hours. These systems can cause unnecessary delays and increased fuel consumption due to the static nature of their operation. A dynamic traffic light control system, however, adapts to real-time traffic conditions, thereby improving overall traffic flow. By leveraging modern microcontroller technology like the ESP32, which is equipped with WiFi capabilities, the system can be easily monitored and controlled remotely, providing a significant advantage over traditional systems.

## **Protocols Used**

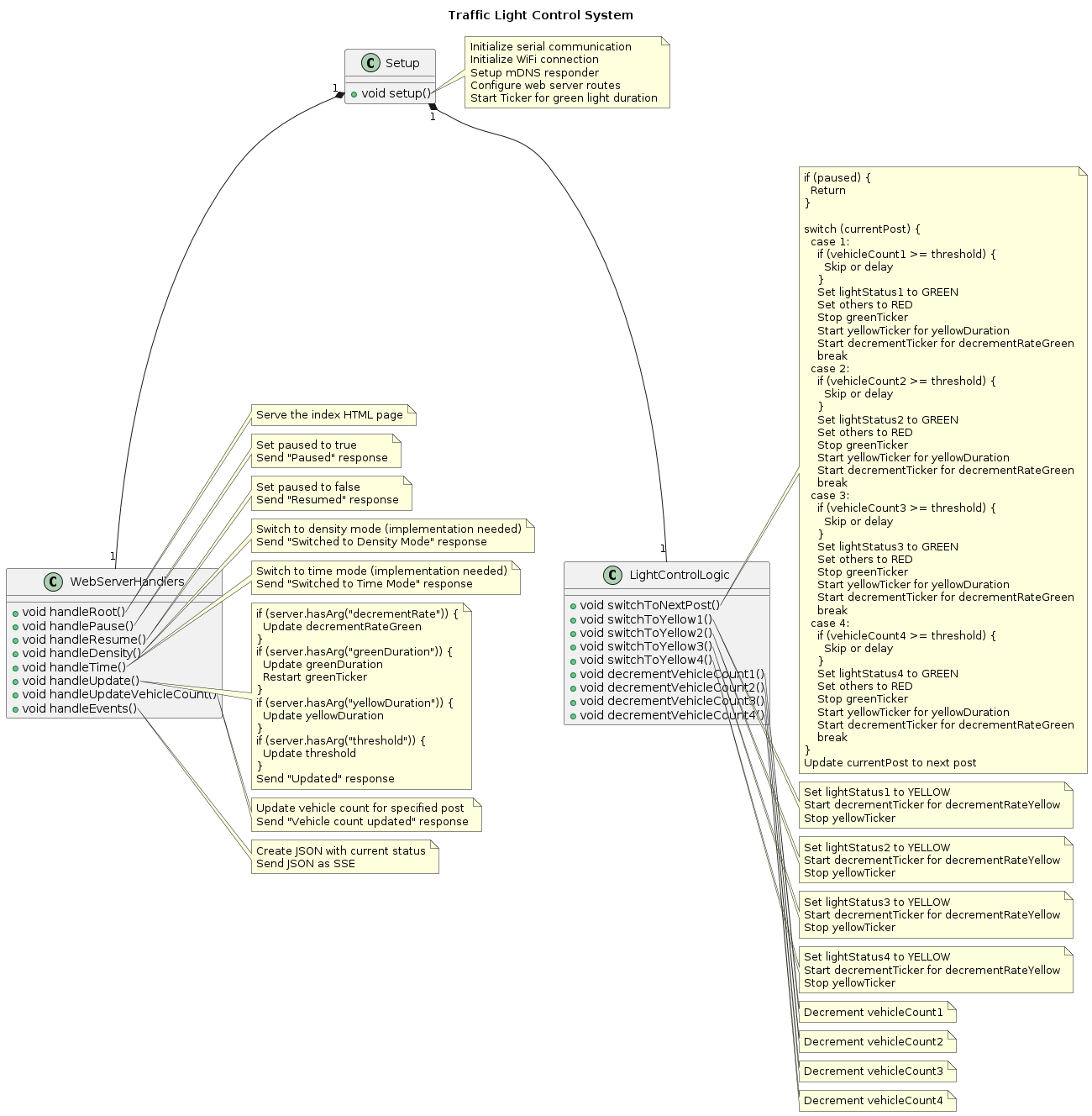
1. **WiFi (IEEE 802.11):** The ESP32 microcontroller connects to a WiFi network, enabling remote monitoring and control. WiFi is a crucial technology for this project as it provides the backbone for communication between the ESP32 and the web interface. By connecting to a local network, the system can be accessed from any device within the network, facilitating easy interaction and control.
2. **HTTP:** Hypertext Transfer Protocol (HTTP) is used for communication between the ESP32 and the web interface. The web server hosted on the ESP32 handles HTTP requests and serves HTML, CSS, and JavaScript files. HTTP is a widely used protocol for data communication on the web, making it an ideal choice for creating a web-based control interface for the traffic light system.
3. **mDNS (Multicast DNS):** Multicast DNS (mDNS) allows the ESP32 to be accessed via a domain name (e.g., esp32.local) instead of an IP address. This simplifies access to the device, especially in a local network where IP addresses can change dynamically. mDNS makes the system more user-friendly by eliminating the need to remember or lookup IP addresses.
4. **JSON (JavaScript Object Notation):** JSON is used for data exchange between the server and the client, particularly for real-time updates. It is a lightweight data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate. JSON is used to send data such as vehicle counts and traffic light statuses to the web interface, enabling real-time monitoring and control.

## **Logics and Algorithms**

The traffic light control system employs two main modes: time-based control and density-based control.

1. **Time-Based Control:**
   * In this mode, the traffic lights switch based on predefined durations (e.g., green for 5 seconds, yellow for 2 seconds).
   * This mode is simpler but less efficient as it does not account for real-time traffic conditions, potentially causing delays during peak hours or unnecessary waiting times during off-peak hours.
2. **Density-Based Control:**
   * The system adjusts the green light duration based on the vehicle count at each intersection.
   * Higher vehicle counts result in longer green light durations to clear the traffic more efficiently.
   * This mode utilizes real-time data to optimize traffic flow, reducing congestion and improving overall traffic efficiency.

**Flow Chart**

Fig. Representation of the logic of the code developed in a flowchart

## **Explanation of Code**

### Initialization

* **Initialize ESP32:** The system begins by initializing the ESP32 microcontroller, setting up serial communication for debugging, and configuring the GPIO pins for the traffic lights.
* **Connect to WiFi:** The ESP32 attempts to connect to the specified WiFi network. This step is crucial for enabling remote control and monitoring via the web interface.
* **WiFi Connection Check:** The system continuously checks if the WiFi connection is successful. Upon successful connection, the system starts the mDNS service, initializes the web server, and sets up tickers for managing the light states and timers. If the connection fails, the system retries the WiFi connection.
* **Set Initial Light States:** Initially, all traffic lights are set to RED to ensure a safe starting state.

### Main Loop

* **Main Loop:** The system runs continuously in a loop, checking and updating the state of the traffic lights and handling web server requests.
* **Paused Check:** If the system is paused, it waits without making any changes to the traffic lights. This feature allows manual intervention or system maintenance without disrupting the traffic flow.
* **Mode Check:** The system determines if it is operating in Time-Based Mode or Density-Based Mode.

Modes of Traffic-

* + **Time-Based Mode:** The system switches to the next traffic light post and sets the green and yellow durations based on predefined times.
  + **Density-Based Mode:** The system calculates the green duration based on vehicle counts, switches to the next post, and sets the durations accordingly.
* **Decrement Vehicle Counts:** Vehicle counts are decremented periodically to simulate the passage of vehicles through the intersection.
* **Update Timers:** The system updates the timers for the traffic lights, ensuring accurate timing for the light changes.

### Web Server Handlers

* **Handle Root Request:** Serves the main HTML page when the root endpoint is accessed. This page contains the web interface for monitoring and controlling the traffic lights.
* **Handle Pause Request:** Pauses the system when a pause request is received. This allows users to stop the system temporarily for manual control or maintenance.
* **Handle Resume Request:** Resumes the system when a resume request is received, returning it to normal operation.
* **Handle Mode Switch:** Switches the control mode between Time-Based and Density-Based when respective requests are received. This feature allows users to choose the most suitable mode for current traffic conditions.
* **Handle Update Requests:** Updates the system configurations (e.g., timings, thresholds) when update requests are received. This feature allows dynamic adjustment of the system parameters.
* **Handle Vehicle Count Update:** Updates the vehicle counts for the intersections when update requests are received, ensuring accurate real-time data for density-based control.
* **Handle Events Request:** Sends real-time data to the client when an events request is received. This enables continuous monitoring and updates on the web interface.

## **Components and Libraries**

Hardware Used

1. ESP32 Microcontroller:
   * Specifications: Dual-core processor, WiFi and Bluetooth connectivity, 520 KB SRAM, 4 MB Flash memory.
   * Role: Central controller for the traffic light system, handling all logic, communication, and control tasks.
2. Traffic Lights:
   * Specifications: Standard 3-color (red, yellow, green) LED traffic lights.
   * Role: Visual indicators for controlling vehicle movement at intersections.
3. Resistors:
   * Specifications: Various values as needed.
   * Role: Current limiting components to protect LEDs and other circuit elements.
4. Breadboard and Jumper Wires:
   * Specifications: Standard breadboard and jumper wires.
   * Role: Prototyping platform and connections between components.

Software Used

1. Arduino IDE:
   * Specifications: Integrated development environment for writing, compiling, and uploading code to the ESP32.
   * Role: Development and programming environment for the ESP32 microcontroller.
2. HTML/CSS/JavaScript:
   * Specifications: Standard web technologies for creating the web interface.
   * Role: Provides the user interface for monitoring and controlling the traffic light system.
3. C++:
   * Specifications: Programming language used for writing the main logic and control code.
   * Role: Provides the core functionality for traffic light control and communication.

**Libraries Used**

1. **WiFi.h:** This library handles WiFi connectivity, allowing the ESP32 to connect to a wireless network. It provides functions for connecting to WiFi, checking connection status, and managing network credentials.
2. **WebServer.h:** This library manages the web server functionalities. It provides a simple way to set up a web server on the ESP32, handling HTTP requests and serving HTML, CSS, and JavaScript files.
3. **ESPmDNS.h:** This library enables mDNS functionality for easier network access. It allows the ESP32 to be accessed via a domain name instead of an IP address, simplifying the process of connecting to the device.
4. **ArduinoJson.h:** This library facilitates JSON parsing and generation for data exchange. It provides functions for creating JSON objects and arrays, parsing JSON strings, and serializing data for transmission.
5. **Ticker.h:** This library provides timed callbacks to manage traffic light states and vehicle count decrements. It allows the system to execute functions at regular intervals without blocking the main loop.

## **Network Configuration**

## WiFi Configuration:

const char\* ssid = "OnePlus";

const char\* password = "719622@Aa";

* SSID and Password: The ESP32 connects to a specified WiFi network using the provided SSID and password. This configuration enables remote access to the web interface from any device on the same network.
* IP Address: The ESP32 obtains an IP address from the DHCP server on the network. This IP address is used to access the web interface and control the traffic light system.

1. mDNS Configuration:

* Domain Name: The ESP32 is configured with a domain name (e.g., esp32.local) using mDNS. This allows users to access the web interface without needing to know the IP address, providing a more user-friendly experience.

**Traffic Light States and Variables**

### Traffic Light States

1. **RED:** Indicates that vehicles must stop at the intersection.
2. **YELLOW:** Indicates that vehicles should prepare to stop, signaling an imminent change to red.
3. **GREEN:** Indicates that vehicles can proceed through the intersection.

### Variables

1. **Current Post:** The intersection currently being controlled by the system.
2. **Vehicle Counts:** The number of vehicles waiting at each intersection.
3. **Timers:** The durations for which each traffic light stays in a particular state.
4. **Light Status:** The current state of each traffic light (red, yellow, green).
5. **Paused:** A flag indicating whether the system is paused.
6. **Control Mode:** The current mode of operation (time-based or density-based).

* LightStatus: An enumeration defining the possible states of each traffic light (RED, YELLOW, GREEN, ADJACENT).

enum LightStatus { RED, YELLOW, GREEN, ADJACENT };

LightStatus lightStatus1 = RED, lightStatus2 = RED, lightStatus3 = RED, lightStatus4 = RED;

unsigned long greenDuration = 5000;

unsigned long yellowDuration = 2000;

unsigned long decrementRateGreen = 400;

unsigned long decrementRateYellow = 700;

unsigned long threshold = 5;

* greenDuration, yellowDuration: Define the duration of green and yellow lights in milliseconds.
* decrementRateGreen, decrementRateYellow: Define the rates at which the vehicle count is decremented during green and yellow light phases.
* threshold: Specifies the vehicle count threshold for switching between density-based and time-based control.

**Setup and Initialization**

* Initialize Serial Communication: Sets up serial communication for debugging purposes.

void setup() {

Serial.begin(115200);

WiFi.begin(ssid, password);

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

delay(1000);

}

if (!MDNS.begin("esp32")) return;

server.on("/", handleRoot);

server.on("/pause", handlePause);

server.on("/resume", handleResume);

server.on("/density", handleDensity);

server.on("/time", handleTime);

server.on("/update", handleUpdate);

server.on("/updateVehicleCount", handleUpdateVehicleCount);

server.on("/events", handleEvents);

server.begin();

greenTicker.attach\_ms(greenDuration, switchToNextPost);

}

* Initialize Serial Communication: Sets up serial communication for debugging purposes.
* WiFi Connection: Connects the ESP32 to the specified WiFi network, enabling remote access and control.
* MDNS Initialization: Allows accessing the ESP32 using a domain name (e.g., esp32.local), simplifying the connection process.
* Web Server Initialization: Sets up various endpoints for handling HTTP requests, enabling interaction with the web interface.
* Ticker Initialization: Sets up a ticker to switch the traffic lights at specified intervals, ensuring timely transitions.

**Web Server Handlers**

* Root Handler - Serves the main HTML page: This page contains the web interface for monitoring and controlling the traffic lights.

void handleRoot() {

server.send\_P(200, "text/html", index\_html);

}

* Pause and Resume Handlers - Pauses the system: Temporarily stops the traffic light control for manual intervention or maintenance.
* Resumes the system: Returns the system to normal operation.

void handlePause() {

paused = true;

server.send(200, "text/plain", "Paused");

}

void handleResume() {

paused = false;

server.send(200, "text/plain", "Resumed");

}

**Mode Switch Handlers -** Switches between control modes: Allows the user to select between time-based and density-based control modes.

void handleDensity() {

server.send(200, "text/plain", "Switched to Density Mode");

}

void handleTime() {

server.send(200, "text/plain", "Switched to Time Mode");

}

**Update Handlers -** Updates configurations: Allows dynamic adjustment of system parameters such as timings and vehicle counts.

void handleUpdate() {

if (server.hasArg("decrementRate")) {

decrementRateGreen = server.arg("decrementRate").toInt();

}

if (server.hasArg("greenDuration")) {

greenDuration = server.arg("greenDuration").toInt();

greenTicker.attach\_ms(greenDuration, switchToNextPost);

}

if (server.hasArg("yellowDuration")) {

yellowDuration = server.arg("yellowDuration").toInt();

}

if (server.hasArg("threshold")) {

threshold = server.arg("threshold").toInt();

}

server.send(200, "text/plain", "Updated");

}

void handleUpdateVehicleCount() {

int postId = server.arg("postId").toInt();

int count = server.arg("count").toInt();

switch (postId) {

case 1: vehicleCount1 = count; break;

case 2: vehicleCount2 = count; break;

case 3: vehicleCount3 = count; break;

case 4: vehicleCount4 = count; break;

}

server.send(200, "text/plain", "Vehicle count updated");

}

**Real-Time Updates -** Sends real-time updates: Provides continuous monitoring and updates on the web interface using Server-Sent Events (SSE).

void handleEvents() {

String json = createJson();

server.send(200, "text/event-stream", json.c\_str());

}

**Traffic Light Control Logic - Switches traffic lights:** Changes the traffic lights at each post based on the current mode and vehicle counts.

### Switch to Next Post

void switchToNextPost() {

if (paused) return;

switch (currentPost) {

case 1:

lightStatus1 = GREEN; lightStatus2 = RED; lightStatus3 = RED; lightStatus4 = RED;

greenTicker.detach();

yellowTicker.attach\_ms(yellowDuration, switchToYellow1);

decrementTicker.attach\_ms(decrementRateGreen, decrementVehicleCount1);

break;

// Similar logic for other posts...

}

currentPost = (currentPost % 4) + 1;

}

### Switch to Yellow

void switchToYellow1() {

lightStatus1 = YELLOW;

decrementTicker.attach\_ms(decrementRateYellow, decrementVehicleCount1);

yellowTicker.detach();

greenTicker.attach\_ms(greenDuration, switchToNextPost);

}

// Similar logic for other posts...

* **Changes light to yellow:** Prepares for the next post by switching the current green light to yellow.

### Vehicle Count Decrement

void decrementVehicleCount1() {

if (vehicleCount1 > 0) vehicleCount1--;

}

// Similar logic for other posts...

* **Decrements vehicle counts:** Simulates the passage of vehicles through the intersection.

## Create JSON for Real-Time Updates

String createJson() {

DynamicJsonDocument doc(1024);

doc["vehiclecount1"] = vehicleCount1;

doc["vehiclecount2"] = vehicleCount2;

doc["vehiclecount3"] = vehicleCount3;

doc["vehiclecount4"] = vehicleCount4;

doc["timer1"] = timer1;

doc["timer2"] = timer2;

doc["timer3"] = timer3;

doc["timer4"] = timer4;

doc["lightStatus1"] = lightStatus1 == RED ? "red" : lightStatus1 == YELLOW ? "yellow" : lightStatus1 == GREEN ? "green" : "adjacent";

doc["lightStatus2"] = lightStatus2 == RED ? "red" : lightStatus2 == YELLOW ? "yellow" : lightStatus2 == GREEN ? "green" : "adjacent";

doc["lightStatus3"] = lightStatus3 == RED ? "red" : lightStatus3 == YELLOW ? "yellow" : lightStatus3 == GREEN ? "green" : "adjacent";

doc["lightStatus4"] = lightStatus4 == RED ? "red" : lightStatus4 == YELLOW ? "yellow" : lightStatus4 == GREEN ? "green" : "adjacent";

String json;

serializeJson(doc, json);

return json;

}

* **Creates JSON data:** Generates JSON data for real-time updates, including vehicle counts and traffic light statuses.

## **Implementation Procedure**

1. **Set Up Hardware:**
   * Connect the ESP32 to your computer via USB.
   * Set up the Arduino IDE with ESP32 board support.
2. **Write and Upload Code:**
   * Write the code for traffic light control, web server, and real-time updates.
   * Upload the code to the ESP32 using the Arduino IDE.
3. **Connect to WiFi:**
   * Ensure the ESP32 connects to the specified WiFi network.
   * Access the web interface through the IP address or esp32.local.
4. **Interact with Web Interface:**
   * Use the web interface to monitor and control the traffic lights.
   * Adjust the configurations as needed via the control menu.
5. **Monitor and Adjust:**
   * Monitor the real-time updates.
   * Make adjustments to the traffic light timings and vehicle counts as required.

## **Additional Hardware Options**

1. **Raspberry Pi:**
   * **Specifications:** More powerful processing capabilities, built-in Ethernet and WiFi, support for a variety of sensors and peripherals.
   * **Advantages:** Can handle more complex algorithms and larger data sets, suitable for integrating with other IoT devices and services.
2. **Arduino UNO with WiFi Shield:**
   * **Specifications:** Simpler microcontroller with WiFi capabilities through an additional shield.
   * **Advantages:** Easier to set up for beginners, cost-effective for smaller projects.
3. **BeagleBone Black:**
   * **Specifications:** Similar to Raspberry Pi, with more focus on industrial applications.
   * **Advantages:** Robust and reliable, with extensive I/O options.

## **Applications**

1. **Urban Traffic Management:**
   * Optimizes traffic flow in cities, reducing congestion and travel times.
2. **Smart Cities:**
   * Integrates with other smart city solutions for comprehensive urban management.
3. **Emergency Response:**
   * Provides priority passage for emergency vehicles by dynamically adjusting traffic lights.

## **Advantages**

1. **Reduced Congestion:**
   * By adapting to real-time traffic conditions, the system minimizes waiting times and improves traffic flow.
2. **Fuel Efficiency:**
   * Reduced idling at intersections leads to lower fuel consumption and emissions.
3. **Scalability:**
   * The system can be easily scaled to cover more intersections or integrate with other traffic management solutions.

## **Modifications**

### First Modification: Vehicle Count Rate Adjustment

1. **Add a Slider/Button:**
   * Implement a slider or button in the web interface to adjust the vehicle count decrement rate.
2. **Decrement Vehicle Count:**
   * Decrease vehicle count by 1 every 400 ms during the green light and every 700 ms during the yellow light.

### Second Modification: Round Timer Element

1. **Create Round CSS Element:**
   * Design a round CSS element with transparency and hover effects to display a countdown timer for each light at every post.

### Third Modification: No Post Repetition

1. **Ensure No Post Repetition:**
   * Modify the control logic to ensure that no traffic post is repeated twice in the signal sequence.
   * If a post crosses the threshold value, either skip it or delay the green signal until the vehicle count decreases below the threshold.

### Fourth Modification: Control Menu and Performance Optimization

1. **Create Control Menu:**
   * Design a menu in the web interface to control variables such as vehicle count rate, green/yellow light durations, and threshold value.
2. **Optimize Performance:**
   * Improve website performance to reduce lag and ensure it works in real-time without delays, even beyond localhost.

### Example Code Modifications

<!-- Add sliders and buttons in the web interface -->

<div>

<label for="decrementRate">Vehicle Count Decrement Rate:</label>

<input type="range" id="decrementRate" min="100" max="1000" value="400" step="100" oninput="updateDecrementRate(this.value)">

<span id="decrementRateValue">400 ms</span>

</div>

<!-- CSS for round timer element -->

<style>

.round-timer {

border-radius: 50%;

background: rgba(0, 0, 0, 0.5);

color: white;

text-align: center;

line-height: 50px;

width: 50px;

height: 50px;

transition: background 0.3s;

}

.round-timer:hover {

background: rgba(0, 0, 0, 0.7);

}

</style>

<!-- Example round timer element in HTML -->

<div class="round-timer">10</div>

<!-- Update JavaScript to handle new features -->

<script>

function updateDecrementRate(value) {

document.getElementById('decrementRateValue').innerText = value + ' ms';

fetch(`/update?decrementRate=${value}`);

}

</script>

Example Code for No Post Repetition

void switchToNextPost() {

if (paused) return;

do {

currentPost = (currentPost % 4) + 1;

} while (vehicleCount[currentPost - 1] >= threshold);

switch (currentPost) {

case 1:

lightStatus1 = GREEN; lightStatus2 = RED; lightStatus3 = RED; lightStatus4 = RED;

greenTicker.detach();

yellowTicker.attach\_ms(yellowDuration, switchToYellow1);

decrementTicker.attach\_ms(decrementRateGreen, decrementVehicleCount1);

break;

// Similar logic for other posts...

}

}

This comprehensive report provides a detailed overview of the traffic light control system using ESP32, covering the theoretical background, practical implementation, and suggested modifications for further enhancement.

## **Conclusion**

The ESP32-based traffic light control system effectively manages traffic flow by dynamically adjusting light timings based on vehicle density. The user-friendly web interface provides an intuitive means of controlling and monitoring the system, making it a practical solution for modern traffic management. Potential enhancements include integrating real-time traffic sensors and developing a mobile app interface. The project demonstrates the power and flexibility of the ESP32 microcontroller in creating smart and connected systems, paving the way for future innovations in traffic management technology.