Smart Traffic Management System

CS1005 – Internet of Things Laboratory

&

CMP515 - Internet of Things

Faculty Name: Prof. Gaurav Kumar Gaharwar

Mr. Yash V Jhaveri

SCHOOL OF ENGINEERING & TECHNOLOGY

BACHELOR OF TECHNOLOGY

5th SEMESTER

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Documented By:

Pathan Mohammed Sarhankhan (23000008)

Amin Preet (23000771)

Chauhan Harsh (23000929)

Introduction

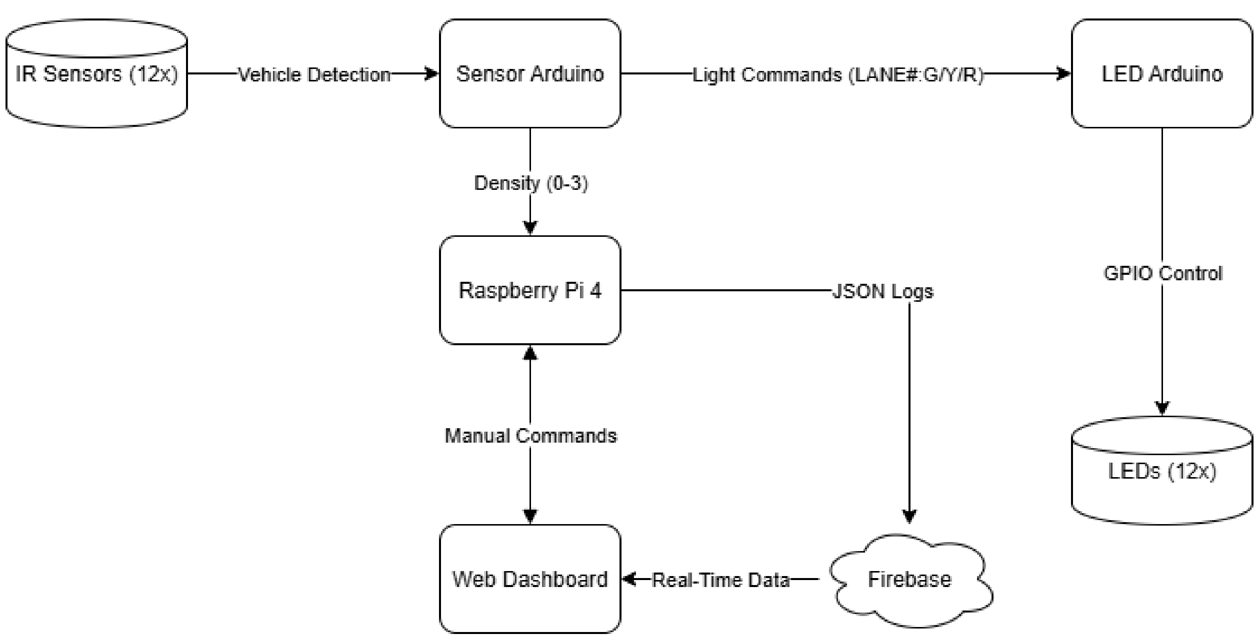
This Phase 3 report details the data communication and storage mechanisms for the IoT-based Smart Traffic Management System designed for a four-way intersection in Indian cities. The system uses two Arduino Unos (Sensor Arduino for 12 IR sensors and LED Arduino for 12 LEDs), a Raspberry Pi 4 for cloud connectivity, Firebase for data storage, and a React.js web dashboard for real-time monitoring and manual control. The report analyzes data flow, communication models, protocols, storage, and processing to ensure efficient traffic management, reduced congestion, and reliable operation.

Data Communication & Storage Analysis

1. Data Flow

The data flow in the system is structured to ensure real-time vehicle density detection, traffic light control, data logging, and remote monitoring/control. The process is as follows:

Data Flow Diagram: Smart Traffic Management System

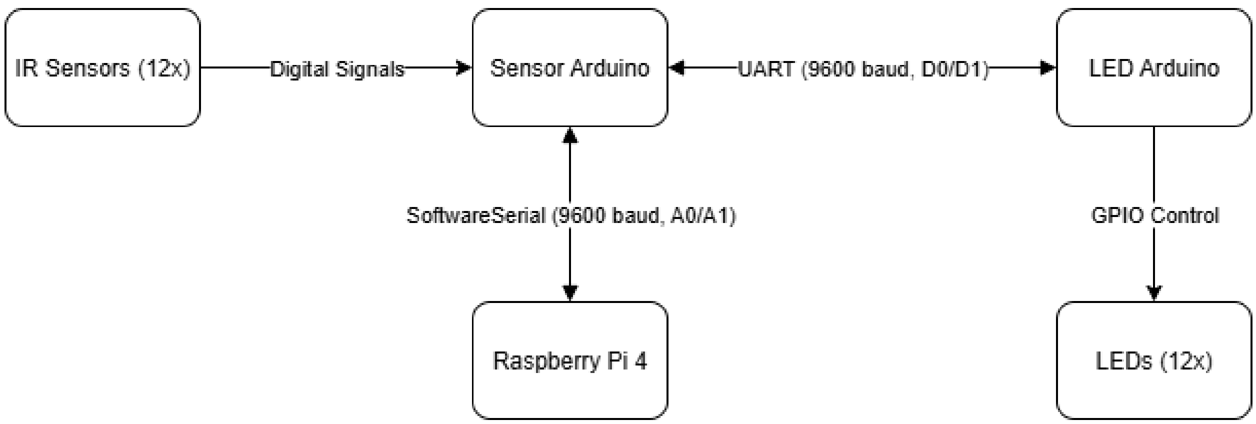


- Sensor Layer (Input):
  12 IR sensors (3 per lane for 4 lanes: North, East, South, West) detect vehicle presence, outputting digital signals.
  Each sensor connects to the Sensor Arduino, providing data for density calculation (0–3 vehicles density per lane based on triggered sensors).
- Data Collection Layer (Sensor Arduino):
  The Sensor Arduino reads sensor states every cycle (~100ms), computes density per lane, and determines traffic light states based on the algorithm (e.g., green time = 15s base + 10s per density level).
  Density and state data are serialized as a string (e.g., "LANE0:2;LANE1:0;LANE2:3;LANE3:1;") and sent to:
  LED Arduino via hardware serial for traffic light control.
  Raspberry Pi via SoftwareSerial for logging.
- Processing Layer (Sensor Arduino, LED Arduino, Raspberry Pi):
  Sensor Arduino processes density to decide green durations and sends commands (e.g., "LANE1:G") to LED Arduino.
  LED Arduino directly sets LED states based on commands.
  Raspberry Pi parses serial data, logs to Firebase, and hosts the React.js dashboard for visualization and manual overrides.
- Storage Layer (Firebase):
  Raspberry Pi formats data as JSON (e.g., { "lane": "Lane1", "density": 2, "timestamp": "2025-09-29T11:49:00Z", "light_state": "Green" }) and sends it to Firebase Realtime Database via HTTP POST.
- Output Layer (LEDs, Dashboard):
  LEDs (4 Red, 4 Yellow, 4 Green) reflect traffic states for each direction.
  The dashboard displays real-time density and light states, allowing manual overrides (e.g., force green for Lane 2).
- Feedback Loop:
  Dashboard commands (e.g., "MANUAL:2") are sent to Raspberry Pi, relayed to Sensor Arduino via SoftwareSerial, enabling mode switches or emergency overrides.

2. Communication Models

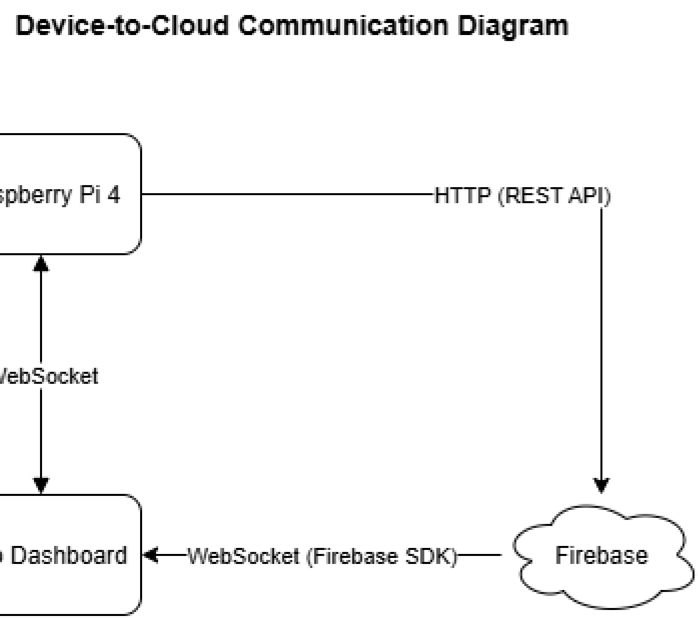
The system employs two communication models: Device-to-Device and Device-to-Cloud.

Device-to-Device Communication Diagram



- Device-to-Device
    - Components Involved: Sensor Arduino, LED Arduino, Raspberry Pi 4.
    - Description:
    - Sensor Arduino to LED Arduino: The Sensor Arduino sends traffic light commands (e.g., "LANE1:G", "ALL:R") to the LED Arduino via hardware serial (UART, 9600 baud) using pins D0 (RX) and D1 (TX). This ensures real-time control of LEDs based on sensor data.
    - Sensor Arduino to Raspberry Pi: Density and state data are sent via SoftwareSerial (A0 TX, A1 RX, 9600 baud) to the Raspberry Pi's GPIO 14 (TXD) and 15 (RXD). A level shifter (5V to 3.3V) ensures safe communication.
    - Raspberry Pi to Sensor Arduino: Manual override commands (e.g., "MANUAL:2", "SMART") are sent back via the same SoftwareSerial channel.
    - Network: Local, wired serial connections (UART), ensuring low latency (~1ms per message) and reliability within the system.

- Purpose: Enables real-time coordination between sensor processing, light control, and higher-level management without external network dependency.



- Device-to-Cloud
    - Components Involved: Raspberry Pi 4, Firebase Realtime Database, React.js Web Dashboard.
    - Description:
    - Raspberry Pi to Firebase: The Raspberry Pi sends JSON-formatted traffic data (density, light states, timestamps) to Firebase Realtime Database using HTTP POST requests over Wi-Fi. The Firebase SDK simplifies authentication and data pushing.
    - Firebase to Dashboard: The React.js dashboard, hosted on the Raspberry Pi, retrieves real-time data from Firebase using the Firebase SDK (WebSocket for live updates). Users view lane densities and light states.
    - Dashboard to Raspberry Pi: Manual override commands are sent via WebSocket to the Raspberry Pi, which relays them to the Sensor Arduino.

- Network: Wi-Fi (Raspberry Pi's built-in module) connects to the internet, enabling cloud storage and remote access. Cellular could be added for remote deployments.
- Purpose: Facilitates data logging for analysis, remote monitoring, and control, critical for traffic optimization and emergency response.

3. Communication Protocols

The system uses a subset of the listed protocols, selected for efficiency, compatibility, and IoT constraints.

Chosen Protocols

- UART (Serial):
  - Role: Device-to-Device communication (Sensor Arduino to LED Arduino, Sensor Arduino to Raspberry Pi).
  - Details: 9600 baud rate, asynchronous serial over D0/D1 (Arduino-to-Arduino) and SoftwareSerial A0/A1 (Arduino-to-RPi). Messages are plain text (e.g., "LANE1:G", "LANE0:2;LANE1:0;").
  - Reason: Low latency, reliable for local wired connections, no external network needed. SoftwareSerial avoids pin conflicts.
  - Constraints: Requires level shifter for Arduino (5V) to Raspberry Pi (3.3V). Limited to short distances.
- HTTP:
  - Role: Device-to-Cloud communication (Raspberry Pi to Firebase).
  - Details: REST API POST requests send JSON data to Firebase Realtime Database.
  - Reason: Supported by Firebase SDK, reliable over Wi-Fi, and simple to implement with Python's `requests` library.
  - Constraints: Higher bandwidth than MQTT, but Raspberry Pi's resources handle it well.
- WebSocket:
  - Role: Dashboard-to-Raspberry Pi communication for real-time updates and control.

  - Details: React.js dashboard uses WebSocket (via Firebase SDK on Raspberry Pi) to receive live data and send manual commands (e.g., "MANUAL:2").
  - Reason: Enables real-time, bidirectional communication for dynamic monitoring and overrides.
  - Constraints: Requires stable Wi-Fi; fallback to HTTP polling if WebSocket fails.
- Connectivity Method: Wi-Fi (Raspberry Pi's built-in module) for cloud and dashboard communication. Serial (wired) for device-to-device. Cellular or Bluetooth could be added for remote deployments or alternative control interfaces.

4. Data Storage

The system uses Firebase Realtime Database for cloud-based storage, chosen for its simplicity, real-time capabilities, and integration with the React.js dashboard.

- Storage Type: Cloud-based, NoSQL, real-time database.
- Database: Firebase Realtime Database.
- Structure: Hierarchical JSON tree. Data stored under `/traffic_logs` with entries like: json
- Access: Raspberry Pi writes via HTTP POST; dashboard reads via Firebase SDK (WebSocket for real-time).
- Data Format for Transfer:
  - Sensor Arduino to LED Arduino: Plain text via UART, e.g., "LANE1:G" (Lane 1 Green), "ALL:R" (all Red).
  - Sensor Arduino to Raspberry Pi: Plain text via SoftwareSerial, e.g., "LANE0:2;LANE1:0;LANE2:3;LANE3:1;".
  - Raspberry Pi to Firebase: JSON via HTTP POST, e.g., { "timestamp": "2025-09-29T11:49:00Z", "densities": { "LANE0": 2, ... } }.
  - Dashboard: JSON via WebSocket, same structure as Firebase.

- Storage Capacity: Firebase Realtime Database supports up to 200,000 concurrent connections and 1GB free storage, sufficient for traffic logs (estimated ~100KB/day for continuous operation).
- Constraints:
  - Requires internet connectivity (Wi-Fi).
  - Data retention policy needed for long-term storage (e.g., archive old logs).
  - Security rules configured in Firebase to restrict access to authorized users (e.g., dashboard, Raspberry Pi).
- Alternative Storage: Local storage on Raspberry Pi (e.g., SQLite) could be used as a backup if internet fails, syncing to Firebase when reconnected. Not implemented to keep system simple.

5. Data Processing and Analysis

Data processing occurs at multiple levels to enable real-time traffic management and provide insights for optimization.

- Sensor Arduino (Real-Time Processing):
  - Process: Reads 12 IR sensor states every ~100ms, counts LOW signals per lane to compute density (0–3). Runs traffic algorithm:
  - Smart Mode: Cycles through lanes (North, East, South, West), setting green time = 15s + (density × 10s), 5s yellow.
  - Manual Mode: Forces green for specified lane on dashboard command.
  - Output: Serial commands to LED Arduino (e.g., "LANE1:G") and density data to Raspberry Pi.
  - Purpose: Real-time decision-making for traffic light control based on vehicle density.
  - Constraints: Limited by Arduino’s 2KB SRAM and 16MHz clock. Density calculation is lightweight (simple counter).
- LED Arduino (Output Control):

    - Process: Parses serial commands from Sensor Arduino and sets LED states using `digitalWrite()`. No processing logic; acts as a slave.
    - Purpose: Simplifies pin management, ensuring reliable LED control.
    - Constraints: Dependent on Sensor Arduino for commands; no local decision-making.
- Raspberry Pi (Data Aggregation and Cloud Interface):
    - Process: Parses serial data from Sensor Arduino (e.g., "LANE0:2;..."), formats as JSON, and sends to Firebase via HTTP POST. Hosts Flask/React.js dashboard for real-time visualization.
    - Analysis: Basic aggregation (e.g., average density per lane) could be computed locally before logging. Currently, raw data is logged for dashboard analysis.
    - Purpose: Bridges local IoT system to cloud, enabling remote monitoring and control.
    - Constraints: Requires stable Wi-Fi; processing limited by Python script efficiency.
- Firebase and Dashboard (Analysis and Insights):
    - Process: Firebase stores all logs. The React.js dashboard queries Firebase for real-time (WebSocket) or historical data (HTTP GET), displaying: Current density per lane (0–3), Current light states (Red/Yellow/Green per lane), Historical trends (e.g., density over time via charts).
    - Analysis: Dashboard can compute insights, e.g., average wait time per lane, peak congestion hours. Currently basic; future enhancements could include predictive analytics.
    - Purpose: Provides traffic authorities with real-time status and manual control (e.g., emergency vehicle prioritization).
    - Constraints: Dependent on Firebase latency (~100ms for WebSocket updates); requires user authentication.
    - Real-Time Decisions:
    - Sensor Arduino: Adjusts green time dynamically (e.g., 45s for density 3, 5s for empty).
    - Dashboard: Allows manual override for emergencies (e.g., green for ambulance in Lane 2).

- Future Analysis: Historical Firebase data can be analyzed (e.g., via Python scripts or Google Cloud tools) for traffic patterns, optimizing fixed timings or predicting congestion.

Conclusion

The Smart Traffic Management System efficiently handles data communication and storage using UART for device-to-device (Sensor Arduino to LED Arduino, Arduino to Raspberry Pi), HTTP for device-to-cloud (Raspberry Pi to Firebase), and WebSocket for dashboard interaction. Data flows seamlessly from IR sensors to LEDs and cloud storage, enabling real-time control and monitoring. Firebase Realtime Database stores JSON-formatted traffic logs, supporting scalable analysis. Processing is distributed across Sensor Arduino (density and timing), LED Arduino (output), and Raspberry Pi (cloud/dashboard), ensuring low-latency decisions. Future enhancements include MQTT for multi-intersection systems and machine learning for predictive traffic management.