Final Project Report: Smart Agriculture Monitoring and Control System using IoT Sensors, Gateway and TCP Sockets

Kerem KURU

Student ID: 220303016

Data Communications and Computer Networks Project

Date: January 22, 2026

Abstract

This report presents a smart agriculture monitoring and control system implemented with multiple TCP-based IoT sensor nodes, an application-level gateway, a multi-threaded cloud server, and an actuator node. Soil moisture, weather, GPS, and thermal sensor nodes periodically transmit measurements to the gateway. The gateway acts as an intermediary layer between edge devices and the cloud, applying traffic shaping using a token bucket algorithm to prevent excessive or bursty sensor traffic. Filtered data is forwarded to the cloud server, where a decision mechanism evaluates soil moisture and rain probability and generates irrigation commands. Commands are delivered to the actuator node via the gateway, and acknowledgements are returned to complete the control loop. The project demonstrates socket programming, client–server communication, gateway-based traffic control (QoS concept), and real-time decision making using simulated IoT data.

Arel Üniversitesi, Müh. Mim. Fak. Bilgisayar Mühendisliği.
Türkoba Mah., Erguvan Sok., No:26, K 34537, Tepekent – Büyükçekmece, İstanbul.

1 Introduction

Modern agriculture increasingly relies on automated monitoring systems to reduce water waste and protect crops against environmental stress. Manual measurements taken once or twice per day are insufficient when soil conditions and weather can change rapidly throughout the day.

This project implements a complete IoT-based monitoring and control system using TCP socket programming in C/C++ on Windows. Multiple software-based sensor nodes simulate soil moisture, weather conditions, GPS location and thermal surface temperature. Instead of connecting sensors directly to a cloud server, all sensor data is first collected by an application-level gateway.

The gateway provides an intermediate control layer between edge devices and the cloud. It performs traffic shaping using a token bucket algorithm to regulate excessive or bursty sensor transmissions and protect the cloud server from overload. Filtered data is forwarded to the cloud server, where a decision mechanism evaluates environmental conditions and generates irrigation commands. Commands are delivered to an actuator node via the gateway, and acknowledgements are returned to complete the control loop.

2 System Architecture and Implementation

The final system consists of independent executable programs that communicate over TCP sockets:

System Components

- **Sensor Nodes (TCP clients)** – *sensor_moisture*, *sensor_weather*, *sensor_gps*, and *sensor_thermal*. Each sensor periodically sends a measurement message to the gateway using a unified format.
- **Gateway (intermediary layer)** – Listens for sensors on port 8000, listens for the actuator on port 8100, connects to the cloud server on port 9000, applies token bucket traffic shaping per sensor, buffers accepted messages in a queue, and forwards them to the cloud server.
- **Cloud Server (multi-threaded TCP server)** – Receives forwarded sensor messages, parses the message fields, keeps the most recent moisture and rain probability values, and produces control decisions. Commands are sent back via the gateway.
- **Actuator Node (TCP client)** – Connects to the gateway and applies received commands (simulated), then sends an ACK message back.

Unified Message Format

All components use a common message structure:

```
MSGID=...|SENDER=...|TYPE=...|TS=...|PAYLOAD=...
```

This structure enables simple parsing with a shared helper function (`GetField`) across gateway, cloud and actuator code.

Ports and Communication Paths

- Sensors → Gateway: TCP port 8000
- Gateway → Cloud server: TCP port 9000
- Actuator ↔ Gateway: TCP port 8100

Implementation Steps (Final)

1. Implement sensor nodes as TCP clients that periodically generate simulated measurements.
2. Implement the gateway as a TCP server for sensors and actuator, and as a TCP client for the cloud.
3. Add token bucket rate limiting in the gateway for each sensor (`SENDER` based).
4. Implement the cloud server as a multi-threaded TCP server that evaluates data and generates commands.
5. Implement the actuator node that receives commands and returns ACK messages.
6. Test the system under different sampling intervals and verify PASS/DROP behaviour in the gateway logs.

3 Theory

TCP Client–Server Model

Communication is handled through TCP sockets, which provide reliable, ordered, byte-stream delivery. In the implemented system, sensor nodes and the actuator operate as TCP clients, while the gateway and cloud operate as servers for different links. Multi-threading is used to handle multiple simultaneous connections.

Gateway Concept in IoT Networks

In realistic IoT deployments, edge devices (sensors) typically do not send data directly to the cloud. A gateway is placed between sensors and the cloud to aggregate traffic, control data rates, enforce policies and isolate the edge network. In this project, the gateway is a software component that receives sensor data locally and forwards it to the cloud server through a single uplink connection.

Token Bucket Traffic Shaping (QoS)

IoT sensors can generate bursty or excessive traffic. To prevent overload, the gateway applies a token bucket algorithm per sensor. Each sensor has a bucket with capacity (burst allowance) and a refill rate (average message rate). When a sensor message arrives, the gateway checks whether a token is available:

- If a token exists, the message is accepted (`PASS`) and forwarded to the cloud.
- If no token exists, the message is dropped (`DROP`).

This demonstrates a practical Quality of Service (QoS) mechanism by regulating traffic at the gateway.

Decision Logic in the Cloud Server

Moisture data alone may produce suboptimal irrigation decisions. Therefore, the cloud server combines soil moisture and rain probability:

- If `moisture < 30` and `rainProb < 30`, the server sends `IRRIGATE`.
- Otherwise, the server sends `DELAY`.

To avoid repeated commands, the cloud server suppresses duplicate consecutive commands.

4 Results

The final implementation successfully demonstrates end-to-end IoT communication using TCP sockets:

- Multiple sensor nodes transmit periodic measurements to the gateway.
- The gateway enforces token bucket traffic shaping per sensor and logs `PASS`/`DROP` events.
- Accepted sensor data is forwarded to the cloud server, where a decision is generated based on moisture and rain probability.
- The actuator node receives commands via the gateway, simulates applying the action, and returns ACK messages.

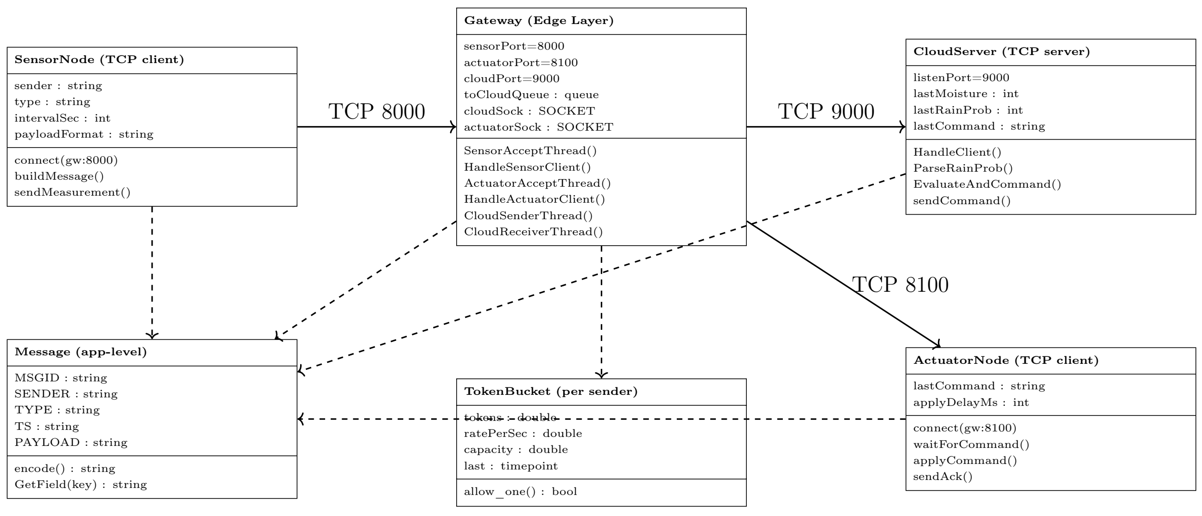

Figure 1: UML diagram of the final system: sensors send to the gateway (token bucket), cloud decides commands, actuator applies and returns ACK

5 Remarks

This project demonstrates a realistic IoT communication architecture in which sensor nodes do not connect directly to the cloud, but communicate through an application-level gateway. The gateway provides an effective control point for traffic regulation and system coordination.

The use of a token bucket mechanism at the gateway limits excessive sensor traffic and prevents burst overload, illustrating a basic Quality of Service (QoS) approach suitable for IoT environments. The modular message structure allows new sensor types to be integrated without modifying the core system.

Overall, the system successfully combines socket programming, multi-threaded servers, gateway-based traffic control, and simple decision logic into a coherent end-to-end IoT monitoring and control solution.

References

1. Beej, B. "Beej's Guide to Network Programming".
2. Stevens, W. R. *UNIX Network Programming*, Vol.1: The Sockets Networking API. Prentice Hall.
3. Tanenbaum, A. S., Wetherall, D. J. *Computer Networks*, 5th Edition.
4. Tutorialspoint. "C — Socket Programming".
5. IBM Developer. "Basics of Message Protocol Design".
6. "Data Communications and Computer Networks Lecture Notes".