SMART GUARD: Centralized IoT Environmental Monitoring System (Using Mininet and MQTT)

Chepuri Venkata Naga Thrisha (23110079), Katta Revathi (23110159), Pappala Sai Keerthana (23110229), Thoutam Dhruthika (23110340).
IIT Gandhinagar.

***Abstract*— In this project, we propose and develop "Smart Guard," a scalable IoT-based solution for environmental monitoring across classrooms and laboratories. The system collects real-time data on temperature, humidity, light intensity, and air quality (CO_2) from 100 simulated sensor nodes deployed using Mininet network emulation. Data is transmitted to a centralized server using the MQTT protocol. The system employs the publish/subscribe (Pub/Sub) communication model, utilizing the MQTT protocol, which enables efficient and scalable message exchange between multiple clients and a central broker. A backend data collector service subscribes to sensor topics, persists time-series data in an InfluxDB database, and an anomaly detection module analyzes incoming data to identify conditions such as high CO_2, low light, or overheating. A comprehensive dashboard built with Grafana provides real-time monitoring, historical trend analysis, and automated anomaly detection with threshold-based alerts. This report outlines the system's architecture, implementation, testing methodology, results, and future scope.**

Index Terms— IoT, Environmental Monitoring, MQTT, Mininet, InfluxDB, Grafana, Network Simulation, Time-Series Database

I. Introduction

Modern classrooms and laboratories require continuous monitoring of environmental conditions to maintain comfort, safety, and energy efficiency. However, using separate sensors or manual checks makes it challenging to track conditions in real-time or manage multiple rooms simultaneously.

To address this, the project employs a centralized monitoring architecture that combines network simulation, MQTT messaging, time-series databases, and dynamic visualization tools. Using Mininet [1], a software-defined network emulator, this system simulates 100 sensor clients deployed across various virtual rooms. These sensors publish environmental data to a Mosquitto MQTT broker, which decouples data producers from consumers via topic-based publish/subscribe communication. A backend data collector ingests and stores sensor readings in InfluxDB, while an anomaly detection module triggers alerts based on predefined thresholds. The collected data are visualized on a Grafana dashboard, featuring heatmaps, graphs, and alerts, which provide comprehensive insights into both real-time and historical environmental conditions.

This system demonstrates how large-scale environmental monitoring can be achieved efficiently and can be applied in schools, universities, and industrial facilities.

What is MQTT?

MQTT [2] (Message Queuing Telemetry Transport) is a lightweight communication protocol designed for use with devices, often found in Internet of Things (IoT) applications. Unlike traditional client-server communication models that require direct connections between endpoints, MQTT decouples message publishers (senders) and subscribers (receivers) through an intermediary known as a broker. A typical MQTT client initiates a TCP/IP connection to the broker with a `CONNECT` command. Publishers then send messages, while subscribers listen to topics and react to incoming data. MQTT's minimal overhead, reliability, and topic-based filtering make it an ideal choice for real-time sensor data exchange, as demonstrated in our current environmental monitoring project.

II. Problem Statement

In classrooms and labs, the environment has a significant impact on comfort, safety, and productivity. Checking conditions manually takes time and does not provide live updates, which means problems like high CO_2 may go unnoticed. It is also challenging to manage multiple rooms simultaneously because the data is scattered and not stored in a centralized location.
This project aims to develop a centralized monitoring system that can collect data from multiple sensors simultaneously. To test this without using real hardware, we simulate the entire network in Mininet and design it to operate efficiently, even with hundreds of sensor devices. Additionally, this project features a visualization dashboard that consolidates live and historical sensor data from all rooms and labs, allowing users to quickly identify environmental issues, monitor trends, and efficiently manage multiple spaces from a single interface.

III. Objectives

The primary objectives of this project are:

- Develop a centralized environmental monitoring system that collects real-time environmental parameters from classrooms and laboratories, including temperature, humidity, light intensity, and carbon dioxide (CO_2) levels.
- **Scalability Testing:** Simulate a scalable network of around 100 sensor nodes using Mininet, representing virtual IoT clients distributed across multiple rooms.
- **Protocol Implementation**: Implement data transmission using the MQTT protocol, ensuring asynchronous publish/subscribe communication between sensors and server modules.
- **Real-Time Data Collection**: Establish automated pipelines for continuous collection of temperature, humidity, CO_2, and light intensity measurements.
- **Time-Series Storage**: Store collected sensor data efficiently in a time-series database (InfluxDB) to enable historical data analysis.
- **Anomaly Detection**: Implement threshold-based alerting for environmental hazards (overheating, poor air quality, inadequate lighting, humidity extremes).
- **Visualization**:
    - Use Grafana for heatmaps, time-series graphs, and alert panels.
    - Build a user-friendly web dashboard (using Dash) that consolidates live sensor data, historical trends, and alert logs to facilitate intuitive monitoring and management.

IV. SYSTEM ARCHITECTURE

The system employs a centralized client–server architecture, where multiple simulated IoT sensor clients transmit real-time environmental data to a central server for storage, processing, and visualization.

At the lowest layer, the environment is simulated using Mininet, where each Mininet host acts as a **simulated IoT sensor node**. These sensor nodes periodically generate readings of temperature, humidity, light intensity, and air quality, and publish them to the MQTT Broker using predefined MQTT topics (e.g., sensors/temperature/CR101/h1). The **Mosquitto MQTT Broker** acts as a message routing service, forwarding incoming data to any subscribed processing services.

The asynchronous, efficient, and reliable QoS support, topic-based filtering, and scalable model of MQTT better match the requirements of your extensive simulated sensor network over other protocols.

On the processing layer, two backend Python services subscribe to these MQTT topics.

- The **Data Collector** receives all sensor readings and writes them into **InfluxDB**, a time-series database optimized for high-frequency IoT streams.
- The **Anomaly Detector** continuously checks for abnormal values (e.g., high temperature, increased CO_2 levels) and logs alerts into a dedicated database measurement.

At the top layer, data visualization is handled by Grafana dashboards directly querying InfluxDB. Grafana displays time-series graphs, heatmaps, and alert panels, making it easy to monitor room conditions and spot anomalies. Additionally, a custom web dashboard built with Dash and Flask provides a unified interface to view live sensor metrics, past trends, and alert logs. The dashboard incorporates filtering and interactive data exploration, complementing Grafana's functionality and enhancing usability for facility administrators..

This architecture is scalable, supporting over 100 sensors, and clearly separates simulation, data routing, analytics, storage, and visualization layers.

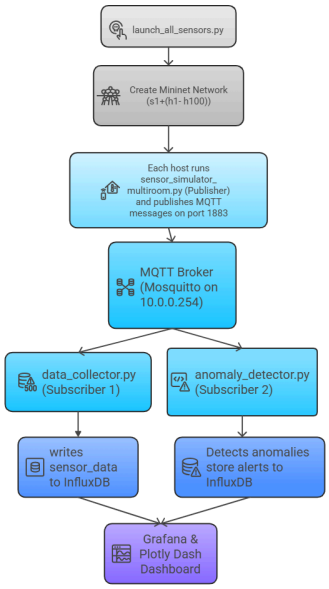


Fig1: Architecture Overview

V. METHODOLOGY

The SmartGuard system was developed and evaluated through Network Setup Script, network simulation, sensor data generation, anomaly generation, backend processing and storage with InfluxDB, and visualization. Each stage is described below.

4.0 Network Interfaces and IPs in VM

The installed Mininet VM is configured with two network interfaces:

- **Adapter 1 (NAT)**: Interface eth1 receives a DHCP IP address, such as 10.0.2.15, and provides internet access.
  The eth1 interface must be manually brought up after each reboot using:
  ```
  sudo ip link set eth1 up
  sudo dhclient eth1
  ```
  to enable external internet connectivity from the VM.

- **Adapter 2 (Host-Only)**: Interface eth0 with an IP address like 192.168.56.103 used for SSH and Grafana access.

Inside Mininet, hosts are assigned IP addresses in the 10.0.0.0/8 range, and the virtual switch is statically assigned the IP address 10.0.0.254.

4.1 Network Setup Script (`setup_network.sh`)

To enable network connectivity for Mininet hosts after each system reboot, a Bash script is executed to configure IP forwarding and NAT translation. The command `sudo sysctl -w net.ipv4.ip_forward=1` enables packet forwarding across network interfaces, while the iptables masquerading rule `sudo iptables -t nat -A POSTROUTING -s 10.0.0.0/8 \! -d 10.0.0.0/8 -j MASQUERADE` provides NAT for the Mininet subnet. This configuration ensures that hosts within the 10.0.0.0/8 network can access external networks while routing traffic through the host machine's IP address.

4.2 Network Simulation using Mininet (`launch_all_sensors.py`)

The `launch_all_sensors.py` script initializes a Mininet virtual network with a single switch (s1) connected to 100 hosts (h1 to h100), each representing a sensor node placed in different rooms (e.g., CR101, LAB5, OFFICE9). Each host is configured to route outbound data through the MQTT broker at `IP 10.0.0.254`, ensuring published sensor readings reach the broker. The script then launches `sensor_simulator_multiroom.py` asynchronously on every host with its corresponding room name and host ID, enabling all 100 sensors to continuously generate and transmit data to the MQTT broker on `port 1883`.

4.3 Sensor Data Simulation (`sensors/sensor_simulator_multiroom.py`) and MQTT Communication

Each Mininet host ran the script `` `sensor_simulator_multiroom.py ``, which simulates environmental parameters such as temperature, humidity, light intensity, and CO_2 concentration every 5 seconds. These values were generated at fixed intervals with controlled randomness to resemble real sensor fluctuations.

It connects to the Mosquitto broker at 10.0.0.254 on port 1883 using the paho-mqtt library with no authentication or TLS (plain MQTT). Each sensor published data to MQTT topics structured as:

sensors/<metric>/<room>/<sensor_id>

4.4 Data Collection (`server/data_collector.py`) and Storage in InfluxDB

A Python-based backend (`data_collector.py`) subscribed to sensors/#, allowing it to receive messages from every sensor across all rooms. Uses paho-mqtt to connect to the broker at `IP 10.0.0.254` on `port 1883`. Upon receiving data, the system parsed the JSON payload, extracted sensor identifiers, timestamps, and measurement values, and stored the readings into InfluxDB[4], a time-series database optimized for high-speed ingestion and historical querying. Each stored entry contained tags such as room, sensor_id, and metric type to support filtering and analytical queries.

4.5 Anomaly Detection (`server/anomaly_detector.py`)

To identify unsafe or abnormal environmental conditions, an anomaly detection service (`anomaly_detector.py`) subscribed to the same MQTT sensor data stream. It compared incoming values against predefined thresholds (e.g., high CO_2 > 1000 ppm, low light < 300 lux). When a threshold was crossed, an alert was both:

- Printed in real-time to the console
- Logged into a separate alerts measurement in InfluxDB

This enabled both instant notification and historical alert tracking.

4.6 Visualization with Grafana Dashboard

A Grafana dashboard[3] was configured to query InfluxDB and display system insights. The dashboard provided:

- Real-time time-series plots of sensor readings
- Heatmaps to compare conditions across multiple rooms
- Alert panels showing anomalies detected by the system

Grafana was accessed from the host machine using the VM's Host-Only `IP 192.168.56.103:3000`.

4.7 REST API Backend (`web_dashboard.py`)

The `web_dashboard.py` file sets up a Flask-based [7] REST API [8] that allows other applications to query, filter, and retrieve sensor and alert data stored in InfluxDB. Key endpoints include registering new sensors, listing blocks and rooms, getting the latest and historical measurements, and fetching threshold and alert information. It connects to InfluxDB (by default at port 8086), exposes HTTP endpoints at port 5000, and uses typical network concepts such as port binding and TCP request handling. Web clients and dashboards consume outputs and responses from this service via standard HTTP requests.

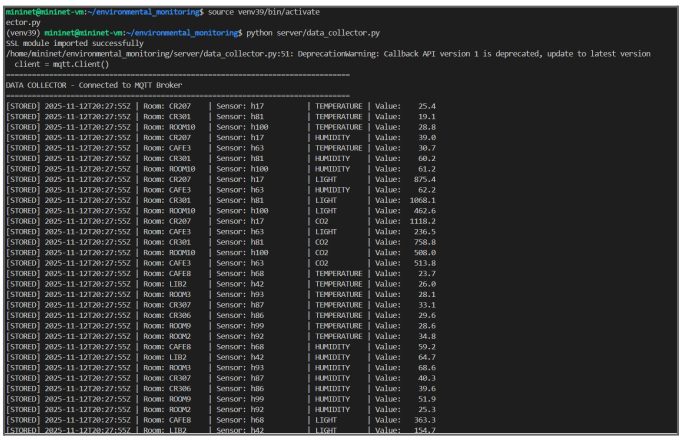
4.8 Custom Web Dashboard Frontend (`app.py`)

The `app.py` file implements an interactive dashboard using Dash (Plotly)[6] served on port 8050. It communicates with the Flask REST API (running on port 5000) to fetch sensor readings, historical trends, room lists, and alert logs in real-time. The dashboard allows users to browse and filter sensor data, view time-series plots and heatmaps for selected metrics and rooms, and check anomaly alerts. Output is rendered as dynamic web charts, cards, and tables, accessible through any web browser. All backend data is retrieved over HTTP and visualized in real-time for effective monitoring and analysis.

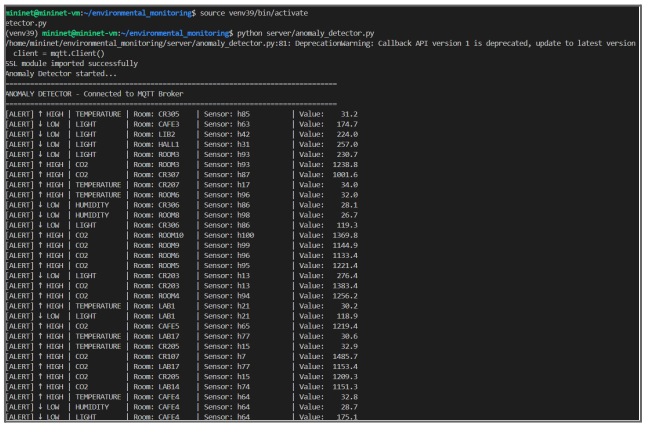
VI. OBJECTIVES ACHIEVED

After running the above scripts in parallel in separate terminals, we obtained the following results:

The terminal output below demonstrates the successful real-time collection and logging of environmental sensor data, including temperature, humidity, light, and CO_2 levels, from multiple rooms and sensors via MQTT. This confirms that the data collector script is actively subscribing to sensor topics and storing incoming values in InfluxDB for dashboard analysis.

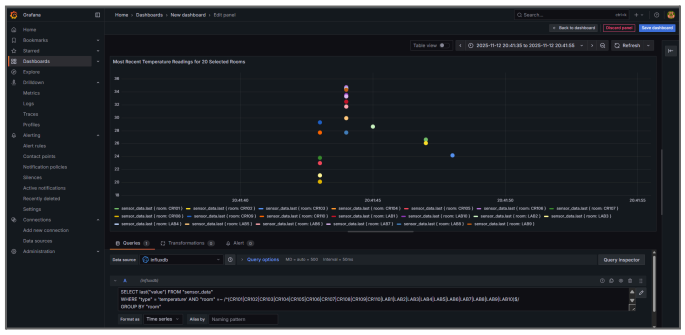


The terminal output below verifies the anomaly detection module is actively monitoring sensor streams and issuing real-time alerts for abnormal values (high or low) across temperature, humidity, light, and CO_2 levels in different rooms.

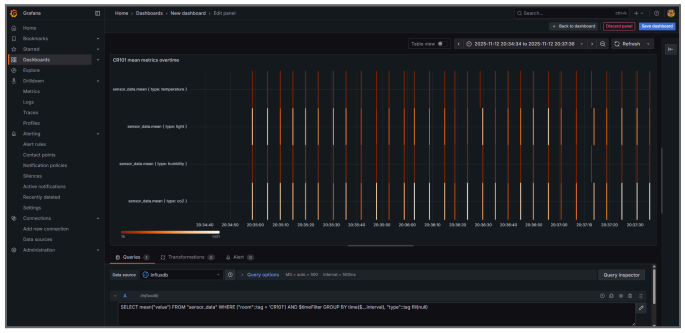


The Grafana panel below displays the most recent temperature readings from 20 selected rooms, allowing for a quick

comparison of environmental conditions across multiple locations using data retrieved from InfluxDB.



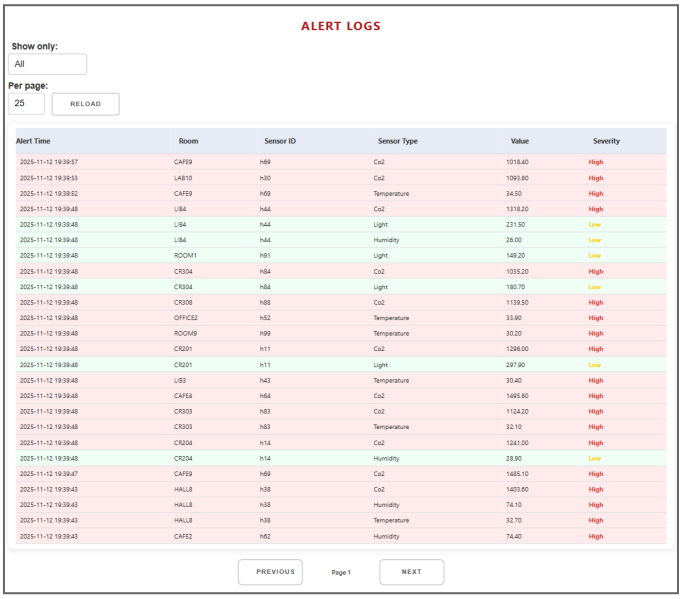
The Grafana dashboard below visualizes real-time mean sensor metrics (temperature, humidity, light, and CO_2) for room CR101 over time, utilizing data queried from InfluxDB to monitor environmental conditions dynamically.



The plotly dashboard below displays real-time and historical environmental data (temperature, humidity, light, and CO_2) for a selected room, allowing users to monitor and analyze room conditions over a chosen time range.

The plotly dashboard below visualizes past trends of individual environmental metrics—temperature, humidity, light intensity, and CO_2 levels—over time, helping to monitor and analyze variations in room conditions.

The plotly dashboard below displays alert logs generated by sensor readings, showing the time, room, sensor type, value, and severity (high or low) to help identify abnormal environmental conditions in real-time.



VII. LEARNINGS

1. Developed practical experience in building a scalable, modular IoT monitoring system using network simulation, message queuing, backend databases, and visualization tools.

2. Deepened understanding of MQTT as an asynchronous publish/subscribe protocol and its advantages for large-scale sensor deployments.

3. Learned how Mininet can be leveraged to create and test virtual networks with many hosts, replicating real-world IoT scenarios without requiring physical hardware.

4. Gained hands-on skills integrating Python (Flask, Dash, Paho-MQTT), InfluxDB, and Grafana to create full-stack data pipelines and dashboards.

5. Explored methods to implement threshold-based anomaly detection and real-time alerting for environmental data streams.

6. Enhanced knowledge of practical computer network concepts such as IP routing, NAT, port assignment, and subnetting in a VM environment.

VIII. FUTURE PROSPECTS

1. Deployment with Real Hardware Sensors
The next step is to replace simulated sensors with actual IoT devices, such as ESP32, DHT22, BH1750, CO_2/MQ-135 sensors, and configure them to publish data over Wi-Fi using MQTT. This would validate the system in real classroom or lab environments.

2. Machine Learning Based Prediction
Data stored in InfluxDB can be used to train ML models for:

- Predicting temperature rise
- Detecting abnormal sensor patterns
- Early anomaly warnings before thresholds are crossed

This allows proactive monitoring rather than reactive alerts.

3. Enhanced Security and Authentication
Upgrade the system to support secure MQTT (TLS encryption, improved QoS) and add user authentication to both data transmission and dashboard access, ensuring privacy and authorized control in real-world deployments.

4. Integration of User Authentication
Extending the REST API and dashboards with user authentication (e.g., OAuth 2.0) will enable secure and personalized access for different user roles, facilitating deployment in public or critical settings.

X. ACKNOWLEDGEMENTS

We sincerely thank our course instructor, **Mr. Sameer G. Kulkarni**, for his valuable guidance and encouragement throughout the CS331: Computer Networks course. We would also like to extend our heartfelt gratitude to **Mr. Ayushman Singh**, our teaching assistant, for his constant support and technical assistance throughout the development of this project. His frequent meetings, prompt responses, and patient explanations greatly helped us resolve doubts and understand complex concepts effectively.

IX. CONCLUSION

This project successfully demonstrates a scalable, centralized environmental monitoring system using simulated IoT sensors within a Mininet network environment. By using the MQTT protocol for efficient data transmission, analysis, and anomaly detection across multiple rooms. The use of simulation allowed performance testing with over 100 clients, proving that the architecture can handle large-scale deployments. This inclusion of anomaly alerts enhances safety by identifying conditions that require immediate attention and action.

Overall, the SmartGuard system provides a robust foundation for practical deployment in educational and industrial settings, where continuous environmental awareness is crucial for maintaining comfort, ensuring safety, and optimizing energy efficiency.

XI. REFERENCES

[1]Lantz, B., Heller, B., and McKeown, N. "A network in a laptop: Rapid prototyping for Software-Defined Networks." *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, Monterey, CA, USA, October 2010. *(The foundational paper for Mininet, which you used for network simulation.)*

[2]Banks, A., and Gupta, R. "MQTT Version 3.1.1." OASIS Standard. October 2014. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html *(The official specification for the MQTT protocol you implemented.)*

[3]S. H. L., T. K. L., and C. F. L. "Design and Implementation of an IoT-Based Environmental Monitoring System with MQTT, InfluxDB, and Grafana," *2021 International Conference on Computer and Information Sciences (ICCOINS)*, 2021, pp. 1-6. *(A paper with an almost identical technology stack, ideal for citing as related work.)*

[4]InfluxData. "InfluxDB Technical Documentation." [Online]. Available: https://docs.influxdata.com/influxdb/ *(The official documentation for the time-series database you used.)*

[5]Grafana Labs. "Grafana Documentation." [Online]. Available: https://grafana.com/docs/grafana/latest/ *(The official documentation for the Grafana visualization platform.)*

[6]"Dash Documentation & User Guide | Plotly," *dash.plotly.com*. https://dash.plotly.com

[7]Flask, "Welcome to Flask — Flask Documentation (1.1.x)," *flask.palletsprojects.com*. https://flask.palletsprojects.com/

[8]L. Gupta, "What is REST – Learn to create timeless REST APIs," *Restfulapi.net*, Jun. 13, 2019. https://restfulapi.net/