Lab 4 Write-up

Lucais Sanderson

1 Introduction

In this lab, we form an end-to-end IoT system: from a sensor network periodically marking its data to an interactive web interface. The goal was to materialize this setup and make it as robust as possible, improving upon Lab 3.

2 Overall Design

We start with a high-level look at the system. It's comprised of the sensor nodes (Raspberry Pis) that form an IoT network, a database server for storing timestamped data, and a web server that consults the database and serves the data to a browser. See figure 1 for a diagram of the system. There is also a picture of the setup in figure 2.

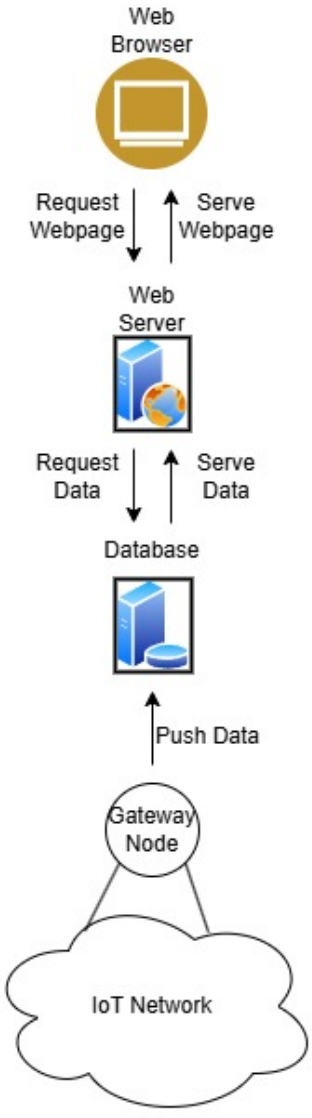

Figure 1: Diagram of the IoT system.

2.1 Node

A "node" is a Raspberry Pi that has 3 sensors connected to it. Its main function is to poll data from those sensors and move the data upstream in the network. A single node is designated as the gateway to the database. So all of the nodes' data is aggregated at the gateway node, which then pushes the bulk of the data to the database.

2.2 Database

The database is a repository running on a server to save the timestamped data for each node. Data is inserted by the gateway node and is queried by the web server.

2.3 Web Server

The web server has a couple of tasks before it can send a webpage to the client/web browser. It needs to query the database for data within a

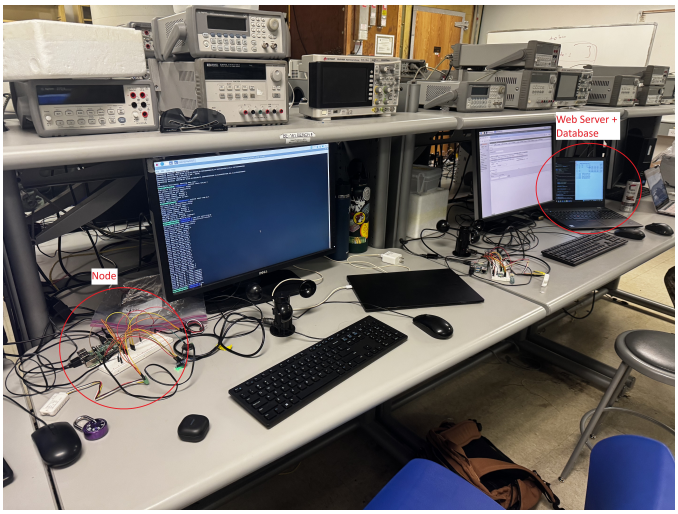

Figure 2: Picture of the system setup with labels.

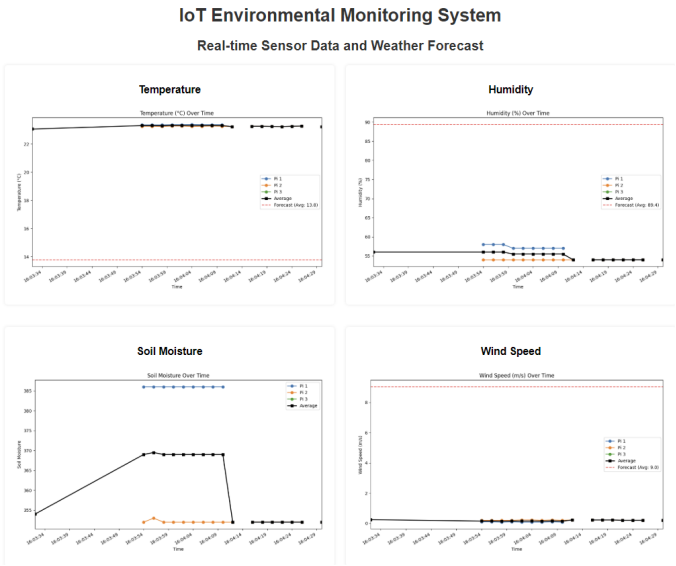

Figure 3: Snapshot of our web page with recent data read from the database.

time window to display in a plot. Additionally, it needs to get and superimpose daily forecast data, from a source on the Internet, onto the data plot. Once these things are done, it can serve the webpage to the client.

3 Web Server/Application Implementation

3.1 Web Server

We configured the web server to run on one of our laptops on an open port, 5000. I did have to disable some firewall settings to access it externally for the web server (and the database server).

3.2 Web Application

We set up the web application so that the last minute of data found in the database was pulled and plotted. The nodes' data points are plotted as points, as well as the average. For the daily forecast, we superimposed that metric as a red line so that there was a clear difference between our data points. A snapshot of the web application can be seen in figure 3.

4 Database Implementation

4.1 Design

We used XAMPP to manage our database (MySQL), PHP handling, and database web interface (phpMyAdmin). This made setting up the database very straightforward and manageable in the context of this lab.

Three tables were needed because we have three nodes generating data. These were named `sensor_readings1`, `sensor_readings2`, and `sensor_readings3` which can be seen in figure 4. This made differentiating between the different nodes' data for insertion and reading far easier than using one table for all the data.

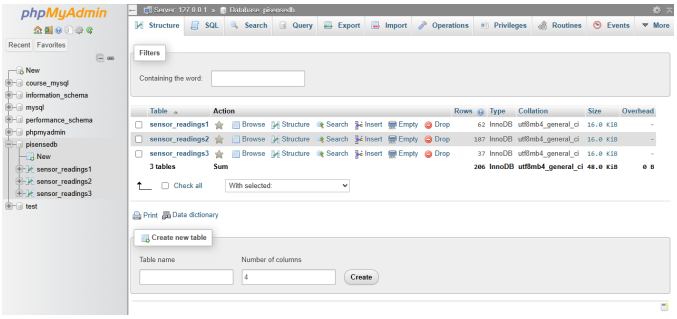

Figure 4: Image on phpMyAdmin showing the three tables in the database.

Each table should be identical in structure. Creating these were done with phpMyAdmin, which executes MySQL functions via a web UI. We used columns: `id` (integer), `timestamp` (timestamp type), `temperature` (float), `humidity` (integer), `soil_moisture` (integer), and `wind_speed` (float). The `id` is

auto-incremented when a new entry is added to the table, just to keep track of each entry. Then, `timestamp` is taken at the time the data is taken at each node, so we know how fresh the data is, and it aids in the time-series plotting by the web server. See figure 5 for an example of one of these tables' contents.

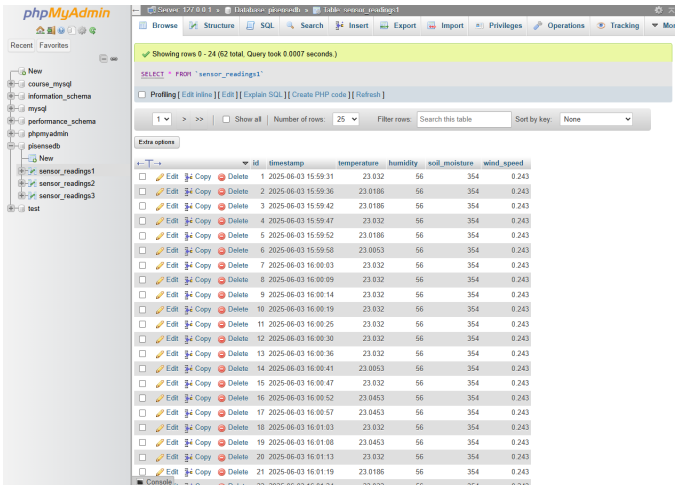

Figure 5: Image on phpMyAdmin showing the contents of a single table.

4.2 Consulting Functions

Having created the tables, we now need to somehow add entries to the database remotely from the gateway node and read from the web server. As suggested by the lab document, we used the MySQLConnector Python library. Using the `.execute()` method from this library, we can exercise any MySQL query on the database, which is chosen in the configuration. For the gateway node, the query is

```
"INSERT INTO `sensor_readings{i+1}`\
    (`id`,`timestamp`,`temperature`,\
    `humidity`, `soil_moisture`,\
    `wind_speed`) "\
    "VALUES(NULL, %s, %s, %s, %s, %s)"
```

The `{i+1}` is a string format for the table corresponding to the node whose data is being sent. The `%s` arguments are filled with the actual sensor data in tuple form.

Reading from the database by the web server is done similarly. Below is the query used by the web server.

```
SELECT timestamp, temperature, humidity,
soil_moisture, wind_speed
FROM {table_name}
WHERE timestamp > %s
ORDER BY timestamp ASC
```

Instead of inserting, we select the data we're interested in, in the table we want, and sort by timestamp.

5 Adapting the Polling vs. Token-Ring Configurations

In this section, we'll compare and contrast the two configurations we implemented for the IoT network. We also review the changes in robustness for each design.

5.1 Token-Ring Robustness Improvements

Our token-ring design this time around is far better than in Lab 3. Before, if a failed link was detected, then we had the whole system essentially shut down. Not very useful. Now, the system will continue to operate unless every node has "died". As a brief on the covered cases, see the bullet points below:

- Case 1: A node that is not a gateway dies, and the token does not die with that node.
  - Eventually, a node that needs to send the token to the downed node in the ring will experience a sender timeout. At this point, this node alters the topology held in the token and attempts to send it to the other node, assuming it is alive. (The case where it is not alive is handled as well)
- Case 2: The gateway node dies, but does not die with the token.
  - The same sender timeout is experienced by one of the living nodes. The gateway node ID is noted in the token. So the node trying to send to the downed gateway node can test this based on its own IP/ID table. When this case is

detected, not only is the topology altered in the token, but the other living node is designated as the new gateway in the token, and the token is sent to that node.

- Case 3: Any node dies before sending the token.
  - In this case, the only timeout we'd experience is a listen timeout. When a listen timeout is detected by any node, each node moves into an independent mode. This means that each becomes a gateway to the database so data can continue flowing.

Case 3 was born from the case where only one node is left standing. It's a valid fail-safe in our scenario because we used the common access point in the lab, and we know each node can contact the database server.

Case 3 can be seen as extreme if the gateway node dies with the token, because technically, the other nodes could reestablish the ring without giving up on the topology. That's certainly another improvement we could make and would likely be more advantageous, especially if we used this system in an environment where the line-of-sight (LOS) to the access point (AP) is only available to certain nodes.

5.2 Polling Robustness Improvements

We couldn't see many new changes to make to the polling robustness. As it was in Lab 3, our robustness plan was satisfactory to us, given the topology. Below are the cases and handling we implemented:

- Case 1: Either or both secondary nodes go offline.
  - The primary experiences a sender timeout and continues as normal, and tries the other node. If no response is found at that node either, then the primary will only send its data to the database.
- Case 2: The primary goes offline.
  - The secondaries experience a listen timeout and shut down. This effectively shuts down the system.

While these safety measures are satisfactory to us for a hierarchical topology, we could take a path similar to token-ring where the secondaries connect directly to the database after timing out. My thoughts on this are that it works in our controlled environment, but as I mentioned earlier, it may be a little unrealistic. Using this topology would imply that (hypothetically) the primary node is the only one that has a connection to the AP, or the best connection.

5.3 Token-Ring vs. Polling

Let's discuss the fairness with which each topology handles the shared medium. This discussion will include general qualities of the topologies as well as our actual implementation.

5.3.1 Fairness

The polling method is inherently less fair to the nodes in terms of access to the shared medium. The primary node is the only one with a direct link to the AP and thus the database. The secondaries depend on the primary to stay alive and be reliable.

In contrast, I think that the token-ring topology is fair in every way, especially in our implementation. Every node can establish a connection to every other node if needed. Any one node can also be a gateway to the database.

5.3.2 Two Scenarios Explored

Here, we compare the behavior of each topology in two scenarios.

First, let's assume that not all nodes have data to send. The polling topology might struggle here. Take that a secondary is ready to send data, but the primary is the controller. i.e., the secondary can only move data upstream after the primary has requested/initiated the connection. In token-ring, if the token only moves after the token-holding node is done adding/modifying it, then nodes that are ready to send data need to wait.

In the second case, we assume all nodes have data to send. Polling does okay here because the primary is ready to grab data, so it polls the secondaries, who are also ready, and the data moves upstream quickly. In token-ring, things can move a little faster than before. No nodes have to wait for the previous one to be ready because all have data to send.

6 Conclusion

This lab was a very satisfying experience to put together all the knowledge from this class and previous. I have been working on IoT projects similar to this and making an end-to-end system like this helps me understand the problems I need to consider for those as well.

7 Group Contribution Statement

My role this this project was to work on the token-ring robustness mechanisms and PHP communication from the node side. I also set up the XAMPP utilities on my laptop, which we used for the demonstration.

Acknowledgments

Thank you to the contributors of the lab document: Andrea David, Harikrishna Kuttivelil, and Firouz Vafadari.

References

[1] "XAMPP tutorial: installation and first steps." Accessed: June 5, 2025. [Online]. Available: https://www.ionos.com/digitalguide/server/tools/xampp-tutorial-create-your-own-local-test-server/

[2] N. Frostbutter, "phpMyAdmin Tutorial and MySQL Tutorial." Accessed: June 5, 2025. [Online]. Available: https://www.youtube.com/playlist?list=PLk-EmIiBIYGF8WCitdIVq7dvvacqY0rdl

[3] "Getting Started with MySQL Connector/Python." Accessed: June 5, 2025. [Online]. Available: https://www.mysqltutorial.org/python-mysql/getting-started-mysql-python-connector/

[4] "Flask." Accessed: June 5, 2025. [Online]. Available: https://flask.palletsprojects.com/en/stable/

[5] "Open-metro: Free Weather API." Accessed: June 5, 2025. [Online]. Available: https://open-meteo.com/