IoT Alarm System with Bed Presence Detection

Lorenzo Galfano
lorenzo.galfano@studio.unibo.it

I. Introduction

Traditional alarm clocks can often be disruptive to our sleep patterns. This IoT alarm system aims to provide a more seamless and non-intrusive waking experience by incorporating bed presence detection. At its core is an edge device equipped with an ESP32 microcontroller and a pressure sensor, which together monitor bed presence and store essential data in a database via a proxy server.

To create a soothing wake-up environment, the system employs an mp3 module and speaker, allowing users to wake up to customized alarm sounds. Through intuitive terminal commands, users can easily adjust settings such as wake-up time and sleep duration, offering a more personalized and flexible routine tailored to individual preferences.

The system also helps users gain insights into their sleep patterns with a data-analysis module, which tracks sleep duration and other useful metrics, including data acquisition efficiency. Finally, Grafana provides an easy-to-use platform for visualizing these sleep statistics, enabling users to monitor and improve their sleep habits over time.

II. Project's Architecture

The architecture is comprised of different key components:

- ESP32: The edge device responsible for data collection.
- Pressure Sensor: Measures current pressure values to determine whether the user is in bed or not.
- MP3 TF-ID Module and Speaker: These components allow the system to play the alarm when triggered.
- Data Proxy Server: A multi-functional server that:
  - Communicates with the ESP32 via HTTP and MQTT protocols, enabling remote control over the sampling rate and the ability to start or stop the alarm.
  - Personalizes the user experience by allowing the setup of important parameters via terminal commands, such as wake-up time, desired sleep duration, and the days when the alarm should activate.
  - Sends the data collected by the ESP32 to an InfluxDB instance.
  - Checks whether it is time to trigger the alarm.
- InfluxDB: A time-series database that stores data collected by the ESP32 along with other relevant information.
- Data-Analysis Module: Computes meaningful metrics utilizing the data collected in the database, such as the total sleep time of the user in a day.
- Grafana: A visualization dashboard used to display the data stored in the database.
- Data-evaluation Module: Computes metrics using the data stored in influxDB to evaluate the overall system performance.

A. Data acquisition

The data acquisition process is primarily handled by the ESP32, which is equipped with a pressure sensor. At regular intervals, determined by the preset *sampling_rate*, the device reads the sensor's value and sends it to the proxy server via an HTTP request. Following this, a second request is sent containing the current sampling rate and the time taken for the last request.

Upon receiving the data, the proxy server processes the information by first determining whether the user is in bed, comparing the pressure value to a predefined threshold. It also checks if the user should be in bed, based on the current time and date. All collected data is then stored in an InfluxDB instance for future analysis and evaluation.

B. Threshold configuration

Since user detection relied on a threshold value, further analysis was conducted to determine an appropriate value for detecting bed usage. The threshold was established empirically by placing the pressure sensor under the bedsheets and allowing the ESP32 to collect data.

In the first phase, data was gathered for approximately five minutes in an idle setting, where the bed remained empty. The recorded values mostly hovered around 0, with occasional higher readings reaching up to 60. In the second phase, data was collected while the user was on the bed for a few minutes. This phase revealed important issues that will be discussed in detail in section IV. The values during this phase ranged from 80 to over 2000, depending on two key factors:

- The sensor's placement: Positions closer to the lower part of the body generated higher values.
- The user's sleeping position: Lying on the side produced significantly higher values than lying on the back.

Given the wide variation in readings, a threshold of 120 was chosen as a balanced value to account for potential false detections, such as objects like books being placed on the bed.

The initial sampling rate was set to 5 seconds but was later adjusted to a range of 15 to 20 seconds. This range was found to be sufficient for detecting meaningful changes, such as the user getting up, moving, or shifting positions in a way that might momentarily affect the sensor's reading.

C. Alarm implementation

The final and most crucial part of making the alarm system functional is the alarm triggering mechanism. The proxy server communicates with the edge device via MQTT and HTTP protocols, with the ESP32 subscribed to three key topics: *'esp32\commands\sampling_rate'*, *'esp32\commands\trigger_alarm'* and *'esp32\commands\stop_alarm'*.

The proxy server runs a daemon thread that routinely checks several conditions. It verifies if the current time matches the scheduled wake-up time, checks if the day is within the days the user set up, and confirms whether the user is still in bed based on the latest sensor data. If all these conditions are met, an HTTP request is sent to the MQTT broker. Upon receiving this request, the ESP32 triggers the alarm.

The alarm plays for 60 seconds, but the user can stop it early by sending a GET request via terminal with the correct denomination, which will be further discussed in III. This activates the *'stop_alarm'* command.

D. Data analysis and evaluation

Finally the last part of the project pipeline is the data analysis and evaluation, both modules communicates with InfluxDB to retrieve the data collected and evaluate the system overall performance.

III. Project's Implementation

A. ESP32

The esp32 firmware was coded using the arduino IDE, which eased the task of installing the needed libraries and writing the code. Several libraries were used for this project and they were:

- *SoftwareSerial.h*: This library allows the Arduino to use additional serial ports (software-defined), which is useful for communication with devices that require serial communication but the hardware serial port is already in use.
- *DFRobotDFPlayerMini.h*: This library is designed to control the DFPlayer Mini MP3 module. It provides simple commands to play, pause, stop, and control the volume of audio files stored on an SD card connected to the DFPlayer Mini.
- *WiFi.h*: A core library for ESP32, used to connect the ESP32 to a Wi-Fi network. It enables the device to act as a client or server in a network, making it suitable for IoT applications.
- *PubSubClient.h*: A lightweight MQTT client library for Arduino. It allows the ESP32 to publish messages to an MQTT broker or subscribe to topics to receive messages from other devices in the network.
- *HTTPClient.h*: A library that allows the ESP32 to make HTTP requests, such as GET or POST, to web servers. This is useful for web-based communication or retrieving data from APIs.

B. MQTT and Docker

The MQTT broker used in this project was **Mosquitto** [1], an open-source message broker that implements the MQTT protocol, which is widely utilized in IoT systems for lightweight and efficient message exchange between devices. Initially, Mosquitto was hosted locally on my machine, but the ESP32 encountered difficulties in receiving messages from the broker. To address these issues and improve reliability, the broker was later containerized using **Docker** [2], a platform that allows applications to be packaged into lightweight, portable containers that can run consistently across different environments.

By containerizing Mosquitto, the broker was isolated from potential conflicts on the host machine, ensuring a more stable and controlled environment for communication between the edge device and the server.

C. Proxy Server

The proxy server was developed using **Flask** and serves as the core component of the application. It allows users to manage their sleep schedule and ensures the alarm is triggered when necessary, while also handling the transmission of data to the time series database. Users interact with the server through terminal commands, which include:

- *set_time*: Sets the wake-up time and automatically adjusts the user's in-bed time based on the current sleeping hours.
- *set_sleeping_hours*: Sets the number of sleeping hours and recalibrates the user's in-bed time according to the wake-up time.
- *set_threshold*: Configures the pressure sensor threshold, with values above the threshold indicating the user is in bed.
- *set_days*: Selects the days on which the alarm should activate.
- *sampling_rate*: Determines the interval (in seconds) between each sample of the pressure sensor taken by the ESP32.
- *stop_alarm*: Stops the alarm if it is currently playing.
- *show_variables*: Displays all the key variables set by the user.

Additionally, the ESP32 communicates with the server through several methods:

- *alarm_stopped*: Informs the server that the alarm has sounded for 60 seconds and has stopped.
- *data*: Sends the current pressure sensor value to the server.
- *time*: Transmits the duration of the HTTP request for sending pressure data. When triggered, the server sends this data to the InfluxDB, along with other metrics such as 'user_in_bed' and 'supposed_to_be_in_bed' for future analysis.

D. InfluxDB

InfluxDB was hosted on the local machine and served as the database throughout the entire project. It was used to

store time-series data, particularly the pressure sensor readings and time taken for the http request. Other values such as 'user_in_bed', 'supposed_to_be_in_bed' and 'sampling_rate' were also stored to evaluate the performance of the system which will be further discussed in IV .

E. Grafana

Grafana was also hosted on the local machine and served to provide a clear visual interface for monitoring these metrics, allowing for real-time insights into sleeping schedules and pressure values.

F. Data Analysis Module

The data analysis module was responsible for calculating the user's sleep duration based on the pressure sensor data. The user can specify the time period for analysis by providing four parameters when launching the program: *start_date*, *start_time*, *end_date* and *end_time*. This flexibility allowed for detailed examination of sleep patterns on any given day.

G. Data Evaluation Module

The evaluation module was utilized to assess various performance metrics of the system. Primarily, it measured the accuracy of detecting whether the user was in bed or not, alongside mean latency of http requests. Additionally, it computed precision, recall and precision-recall curves, providing a comprehensive evaluation of the system's reliability and performance.

IV. Results

A. Computing user's sleep duration

The data sent to InfluxDB included the **sampling_rate**, which was crucial for calculating the total sleep time of the user. The computation process began by identifying the data points that fell within the user's designated sleep period, defined for simplicity as 23:00 to 07:00 each night. Once the relevant time frame was established, only the data points corresponding to this interval were considered for evaluation.

The evaluation involved checking the **user_in_bed** variable, which the proxy server set to one when the pressure sensor's reading exceeded the threshold.

Subsequently, the sampling_rate was assessed to determine its contribution to the total sleep time. For instance, if the sampling_rate was set to 2000 ms, it would imply that 20 seconds were added to the overall sleep duration for each applicable data point. The sum of the sampling_rates gave the total sleep time of the user during the specified time window.

B. Accuracy, Precision and Recall

The primary metric, accuracy, serves as a crucial indicator of the system's overall performance, illustrating the proportion of correct detections relative to the total observations. In addition to accuracy, precision and recall were calculated to offer a more detailed understanding of the system's behavior, particularly regarding false positives and false negatives. These metrics are especially relevant in this context, as misclassifications can significantly affect user experience.

The evaluation metrics reveal a general trend: while the average accuracy barely exceeds random chance on the last day, the system struggles considerably with both precision and recall throughout the entire evaluation period. This indicates that, despite achieving a moderate level of overall correctness, the system has not effectively identified or classified the relevant instances, leading to potential issues for users.

These results are primarily influenced by two key factors:

- **Oversimplification of the System**: The system is designed to consider only the periods between 23:00:00 and 07:00:00 as times when the user is in bed. However, there were numerous instances during the day when I was in bed outside of these designated hours, leading to an increased false positive rate.
- **Sensor Placement Limitations**: The effectiveness of the system is significantly hampered by the placement of the pressure sensor. The mat only covered a small portion of the bed sheet and did not extend the full length of the bed. As a result, a person which moves a lot during night could easily lead to fall off the pressure mat, resulting in a failure to detect the presence. Additionally, sleep position often affected data acquisition; even when being idle in bed, the pressure sensor sometimes failed to register sufficient pressure due to the positioning.

Also, the alarm did not activate on any of the evaluated days, as I consistently moved away from the sleeping sensor before the designated wake-up time of 07:00:00, leading to another potential issue of using this method.

TABLE I
METRICS FOR SLEEP DETECTION OVER THREE DAYS

Day	Accuracy (%)	Precision (%)	Recall (%)
Day 1	62.9	33.5	16.9
Day 2	64.1	35.4	22.0
Day 3	55.7	37.9	45.5
Whole evaluation	58.2	37.2	36.2

C. Mean latency

The mean latency time of HTTP requests was computed by taking all the data points stored in influxDB, the result was 135.5 ms, demonstrating the system's capability for prompt data transmission. This rapid response time is essential for maintaining the real-time nature of the monitoring process.

D. Precision recall curves

Additionally, the precision-recall curve was analyzed to evaluate the stability and reliability of the detection threshold. This curve is essential for understanding the trade-offs between precision and recall, as it illustrates how varying the threshold impacts the rates of true positives and false positives. Given the inherent instability associated with threshold selection, the curve serves as a crucial tool for fine-tuning the threshold value to optimize system performance.

To construct the curve, thresholds ranging from 20 to 4000 were tested in increments of 20. The analysis of the curve indicates that lower thresholds tend to yield higher precision.

Notably, the fifth point on the curve emerges as a promising compromise, where a slight reduction in precision could result in an approximately 20% increase in recall.

Ultimately, a threshold value of 120 appears to be well-suited for this system. While a higher threshold might improve the accuracy of detecting when the user is in bed, it could also lead to the omission of valid data due to the limitations associated with the sensor setup. This nuanced understanding highlights the importance of carefully selecting a threshold that balances the detection of actual user presence in bed while accounting for the variability in sensor performance.

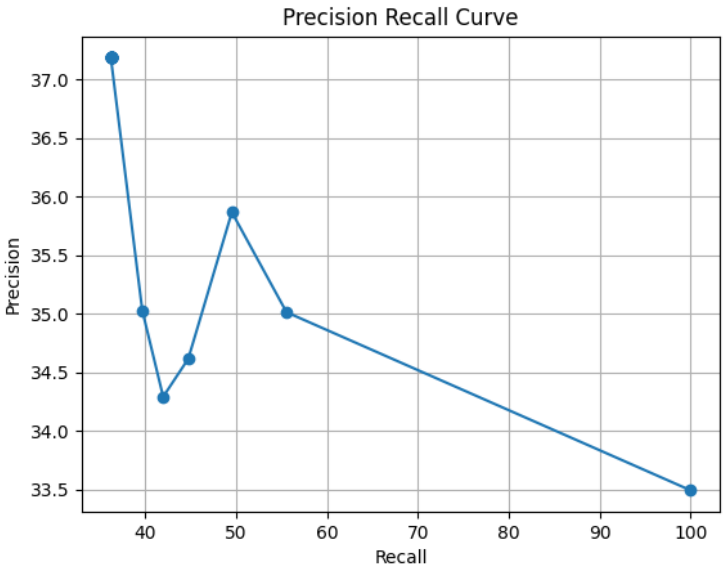

Fig. 1. Precision recall curve of the whole evaluation process.

E. Grafana visualization

Grafana was employed to visualize the values stored in the database. Two distinct panels were created for this purpose: one panel displays the sampling rate, while the other presents the data without the sampling rate. This separation was added because the sampling rate made it challenging to interpret the other values effectively.

The average sleep time across the 3-day evaluation period was computed, revealing an estimated average of 6 hours. While this aligns closely with the expected amount of sleep during night time, it is slightly underestimated due to the inclusion of non-sleep periods during the rest of the day. This inconsistency highlights the need for more refined filtering of data to isolate genuine sleep periods.

To conclude, the system presents several areas for improvement, spanning both hardware configuration and software optimization. The primary issue lies in the placement of the pressure sensor, which led to inaccurate readings, causing the alarm to never trigger as intended. Adjustments in sensor placement or using a larger detection area could address this. Furthermore, the checking of the alarm could be refined by

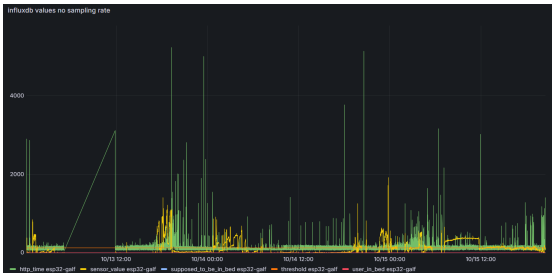

Fig. 2. InfluxDB values without the sampling rate.

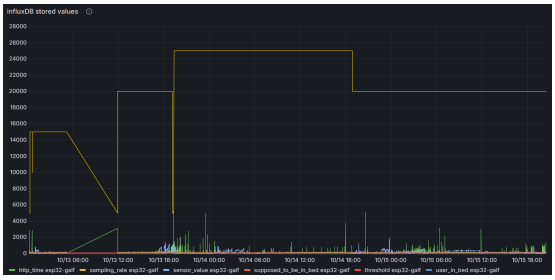

Fig. 3. InfluxDB values with the sampling rate.

introducing variability, such as implementing a lower sampling rate and expanding the alarm check window to include a minute before and after the designated wake time.

With these changes, the system has the potential to provide a more reliable and non-intrusive user experience, achieving its goal of effective sleep monitoring and alarm triggering.

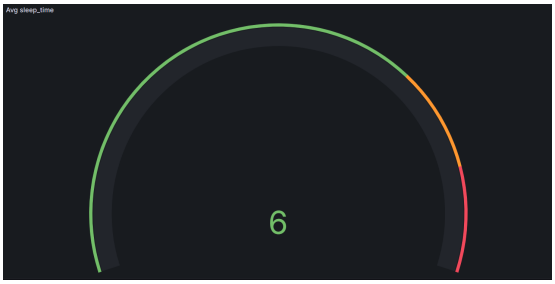

Fig. 4. Average sleep time.

REFERENCES

[1] Light, (2017), Mosquitto: server and client implementation of the MQTT protocol, Journal of Open Source Software, 2(13), 265, doi:10.21105/joss.00265.

[2] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. Linux Journal, 2014(239), 2.

[3] Grinberg, M. (2018). Flask web development: developing web applications with python. " Ox27;Reilly Media, Inc."

[4] K. Elissa, "Title of paper if known," unpublished.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.