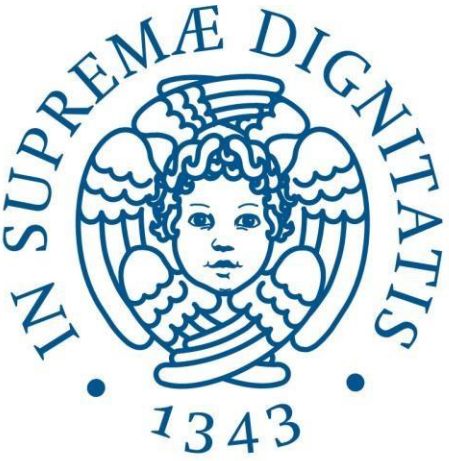


UNIVERSITÀ DI PISA

Artificial Intelligence and Data Engineering Internet of Things

Project Documentation

Smart Diabetic Patient Monitoring System

AUTHOR
Tsegay Teklay Gebrelibanos

Academic Year 2023/2024

Table of Contents

List of Figures

1. Abstract

The Smart Diabetic Patient Monitoring System (SDPMS) is specifically designed to support individuals who are living with diabetes. This system is equipped with a network of sensors capable of providing real-time patient status updates and actuators for automated medication, and emergency assistance. In addition to that, individuals with diabetes often experience cardiovascular disease, the system also includes a feature to monitor cardiovascular health.

The system utilizes various types of sensors for continuous data reading of glucose levels, heart rate, and blood pressure. This data is transferred to a collector application running on a computer and stored in a MySQL database for analysis and vital signal variation identification. The system's actuators play a critical role in maintaining patient well-being. Based on data stored in MySQL and pre-settled thresholds, the system can automatically control insulin and glucagon pumps to regulate glucose levels. An emergency alert system, triggered by either critical health metrics or a dedicated push button, enables patients to quickly summon help when needed.

Additionally, the system also applies MQTT and CoAP application protocols for better energy efficiency.

2. Introduction

2.1.Smart Diabetic Patient Monitoring System

Diabetes is a common chronic disease that affects millions of individuals worldwide, characterized by the body's inability to regulate blood glucose levels effectively. Managing diabetes requires continuous monitoring and timely intervention to prevent complications, including cardiovascular diseases, which are prevalent among diabetic patients. Traditional methods of diabetes treatment often involve frequent manual blood glucose testing and medication administration It is laborious and prone to mistakes made by humans. Therefore, there is a critical need for advanced, automated systems that can provide reliable and continuous monitoring, as well as emergency treatment measures.

The Smart Diabetic Patient Monitoring System (SDPMS) addresses this need by utilizing sensor and actuator IoT technologies to create smart and automated solutions for diabetes treatment. This system is designed to support individuals living with diabetes by providing real-time health status updates and automated medication delivery.

Main functionalities of smart diabetic patient monitoring system:

- **Real-time Monitoring**: Provides continuous data updates of glucose levels, heart rate, and blood pressure, allowing for proactive management of diabetes and cardiovascular health.
- **Automated Responses**: Based on the reading data and pre-set thresholds, the system can automatically regulate glucose levels by controlling insulin or glucagon through pumps.
- **Early Detection of Emergencies**: The analysis engine can identify trends or sudden changes in vital signs that might indicate potential emergencies, triggering alerts for quicker intervention.
- **Improved Patient Outcomes**: Offering real-time data and automated responses can potentially improve diabetic control, reduce complications, and enhance overall health

3. Methodology and Materials

3.1.System Architecture

The architecture of this application is a Wireless Sensor Network (WSN) with two application protocols: One protocol is CoAP, used for direct communication between the IoT field devices (sensors to sense glucose levels and actuators to automate drugs and alert systems). The other is MQTT, which enables the sensor to publish data to the MQTT Broker, and then the data retrieved by the Collector application from the MQTT broker and vice versa. Data received by the collector application is stored in a MySQL database. This database serves as a central repository, enabling the application to activate actuators based on predefined thresholds and to provide meaningful insights into the patient's status.

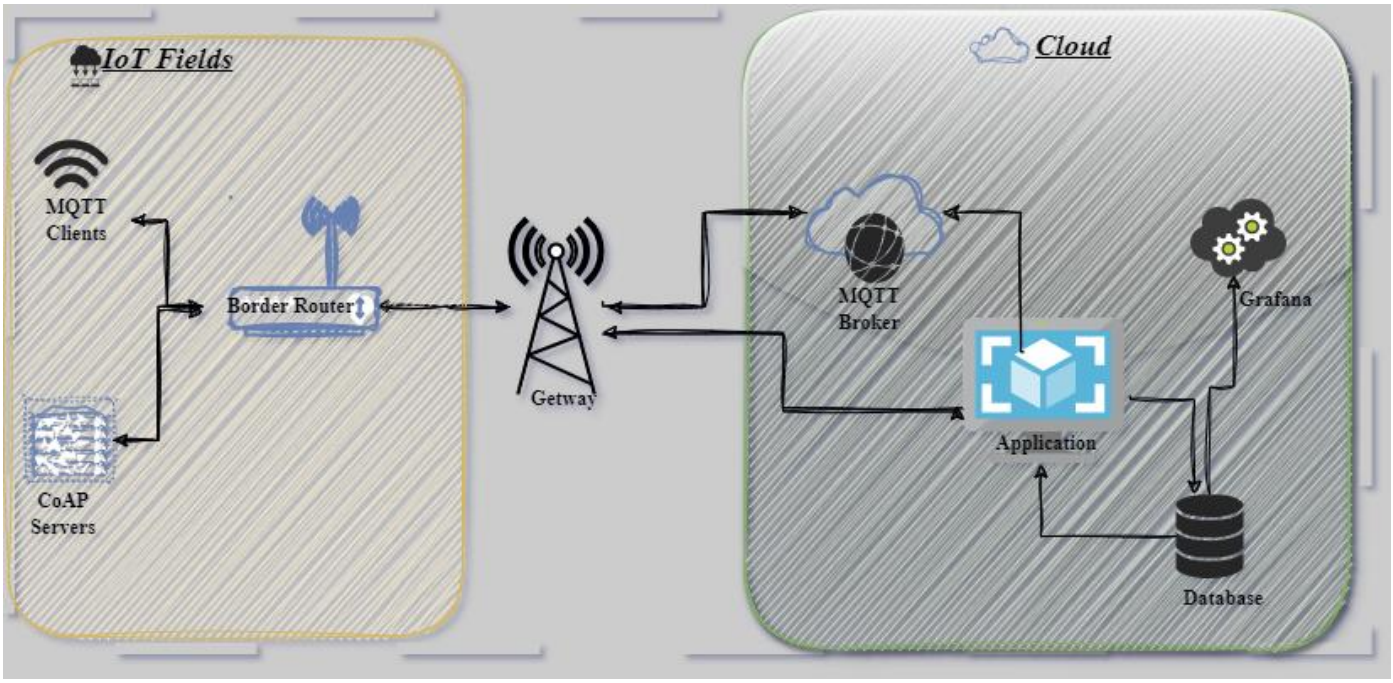


Figure 1: System scheme and structure

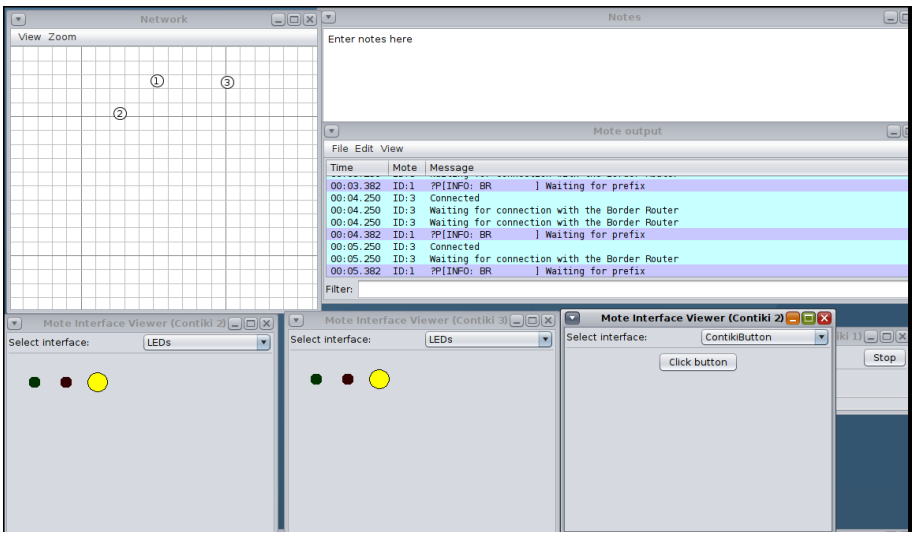


Figure 2: Contiki simulation

3.2. Materials

To implement a demo of the Smart Diabetic Patient Monitoring System I used three Nordic Nrf52840-Dongle USB dongle compatible devices.

Device 1: Border Router to connect the IoT field devices to the gateway. The gateway also forwards data to the external networks to reach the application in the cloud. In the same way, the collector application can reach IoT field devices through the gateway. In this case gateway is the laptop connected to the external internet.

Device 2: This device is dedicated to monitoring and controlling the Patient's Glucose level by continuously reading the glucose levels and automatically activating medications. To achieve this the following sensors and actuators are needed:

- **Sensor:**
  - *Continuous Glucose Monitor (CGM)*: This sensor painlessly tracks glucose levels throughout the day to provide real-time data.
- **Actuators:**
  - *Insulin Pump*: This actuator is automatically controlled by the collector application in the cloud, to deliver precise amounts of insulin based on pre-programmed settings and real-time glucose readings. In my demo, the application activates the insulin pump when the average of 10 consecutive readings of glucose level is greater than 120.

```
Average_glucase ← average (10 consecutive readings)
If (50 < average_glucose >= 120)
      Call ActivateInsulin(put_Insulin_path)
End If
```

  - *Glucagon Pump*: This actuator is also under the control of the collector application. When blood sugar drops dangerously (hypoglycemia) the glucagon pump will activate by the application. In this demo, the collector activates glucagon if the average of 10 consecutive readings is less than 70

```
Average_glucase ← average (10 consecutive readings)
If (average_glucose <= 70)
      Call ActivateGlucagon(put_glucagon_path)
End If
```

  - *Alert System*: the sensor feeds glucose level readings to the application. And the application immediately stores the data in the database. Meanwhile, the application retrieves the 10 last readings from the database to monitor the glucose level. If the average of last readings is greater than 180 or less than 50 the application triggers an emergency alert and notifies medical staff. *Average_glucase ← average (10 consecutive readings*

```
If(not(50<=average_glucose <= 180))
    Call ActivateAlert(put_Alert_path
End If
```

Device 3: This last device is also dedicated to monitoring the patient's cardiovascular diseases using the following sensors and actuators.

- **Sensor:**
  - *Heart rate*: This sensor continuously measures the patient's heart rate, providing essential data for assessing cardiac rhythm and overall cardiovascular stability
  - *Blood pressure:* This sensor monitors blood pressure, another vital sign for cardiovascular health.
  - *Push button:* this enables the patient to trigger an emergency alert when he needs help.
- **Actuator:**
  - *Emergency Alert*: Triggers an emergency notification to pre-defined contacts or healthcare providers when critical health metrics are detected, or a dedicated button is pressed. This alert system is activated by the collector application based on the health metrics. The metrics is:

    if (not(60<average_heart_rate<100) or
    not(90<average_bp 120) or
    button == 1)

    => The averages are calculated from the last 10 readings of the database.

4. Implementation

4.1. Control Application

To control this system, I developed a collector application in Python programming which utilizes some Python libraries.

`paho paho.mqtt.client:` **paho client** library to communicate over MQTT, which enables the application to publish and subscribe to MQTT Topics. For example, the collector application subscribed to the topic called "Heart/Data" to get heart rate readings from the MQTT Broker. And also publish data on a to Topic " Emergency alert" to trigger the emergency alert.

```
// MQTT broker settings, assuming I used a laptop instead of getaway
broker_address = "127.0.0.1"
broker_port = 1883
app_mqtt_client = mqtt.Client()
```

`coapthon.client.helperclient: The` **COAPTHON client** library also allows the collector application to interact with the CoAP server to access resources. The collector reads data from the resources using the GET method and sends the command to the server resources using the PUT method. In this system, the collector gets glucose levels from CoAP resources using the GET method and stores it into the database and is retrieved by the collector for processing. Based on predefined thresholds and analysis of the retrieved data, the collector sends commands back to the CoAP resources to activate or deactivate insulin and glucagon pumps, via PUT methods, ensuring appropriate medication delivery.

```
//to read glucose level data with the help of  Coapthon library
get_path = "glucose/level"
response = client.get(get_path)

/ / to update insulin pump status (turn on/off) with the help of  Coapthon library
put_insulin_path = "glucose_control/insulin"
response = client.put(put_insulin_path, payload)
```

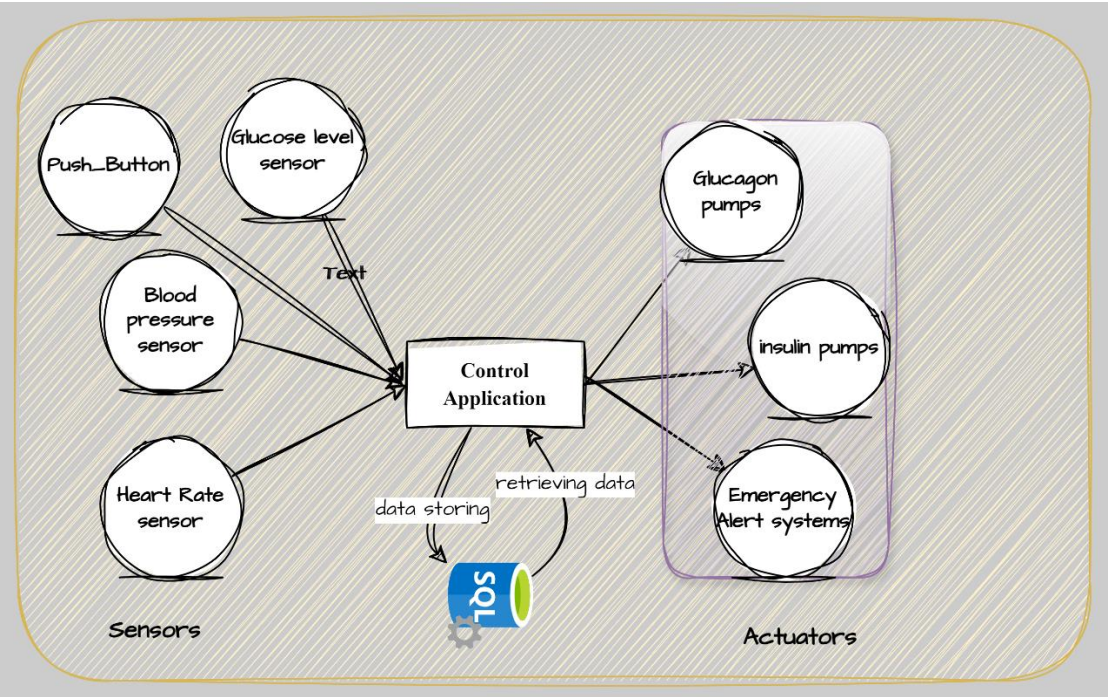


Figure 3: control application to read data from the sensors and send commands to the actuators

Another Python library I used to develop this controller application is `pymysql`. It is a popular python library for interacting with MYSQL databases. It provides a simple and efficient way to connect to MYSQL databases, execute, queries, and retrieve data.

```
// Database connection  settings
db = pymysql.connect(host="localhost", user="root", password="root", db="patientdata")
cursor = db.cursor()

//Example Query to store data into the database
def write_sensor_data(patient_id, glucose_level, incoming_timestamp):
query = "INSERT INTO glucose_monitoring (patientId, glucose_level, incoming_timestamp)
VALUES (%s, %s, %s)"
```

4.2. Data Encoding

The Cloud application for the system is developed using Python to send/receive data to/from the WSN. The data exchanged within the system is encoded using the JSON (JavaScript Object Notation) format, as the application domain and the kind of data exchange don't require a more structured encoding language like XML. For both application protocols MQTT and CoAP I used JSON data encoding.

```
// in MQtt data published in a JSON format
{
\"patientId\":%d,"
"\"ClientID\":\"%s\","
"\"heart_rate\":%d,"
"\"blood_pressure\":%d,"\
"\"button_status\":%d
}"

//In CoAP sends data in JSON format
"{\
"patient_Id\":%d,"
 \"glucose_level\":%d
}",
```

The above shows a JSON object with key-value pairs representing how data is published using JSON data object representation.

4.3. Application Protocols

In this system, I used MQTT and CoAP protocols, which are suitable for resource-constrained environments and their ability to support real-time data transmission. MQTT for cardiovascular emergency monitoring due to its inherited reliability and low latency characteristics. These features are critical for ensuring timely delivery of emergency alerts. While CoAP is used for continuous glucose monitoring, it offers a lightweight and efficient communication mechanism. Since glucose level updates are not as time-sensitive as emergency alerts, CoAP's direct communication style is well-suited for this purpose.

4.3.1. CoAP protocol

As I stated above, I choose the CoAP application protocol for glucose monitoring because it is mandatory to use both protocols and glucose level fluctuations are less time-sensitive than cardiovascular events.

The network architecture for the CoAP protocol is designed as shown in Figure 4, the CoAP network connects the glucose sensor and actuators (alert system, insulin pump, glucagon pump) via the border router. The glucose sensor periodically transmits readings to the border router using CoAP, which relays the data to the application. Based on these readings and predefined thresholds, the application sends commands back through the border router to the actuators, regulating insulin/glucagon delivery or triggering alerts as needed.

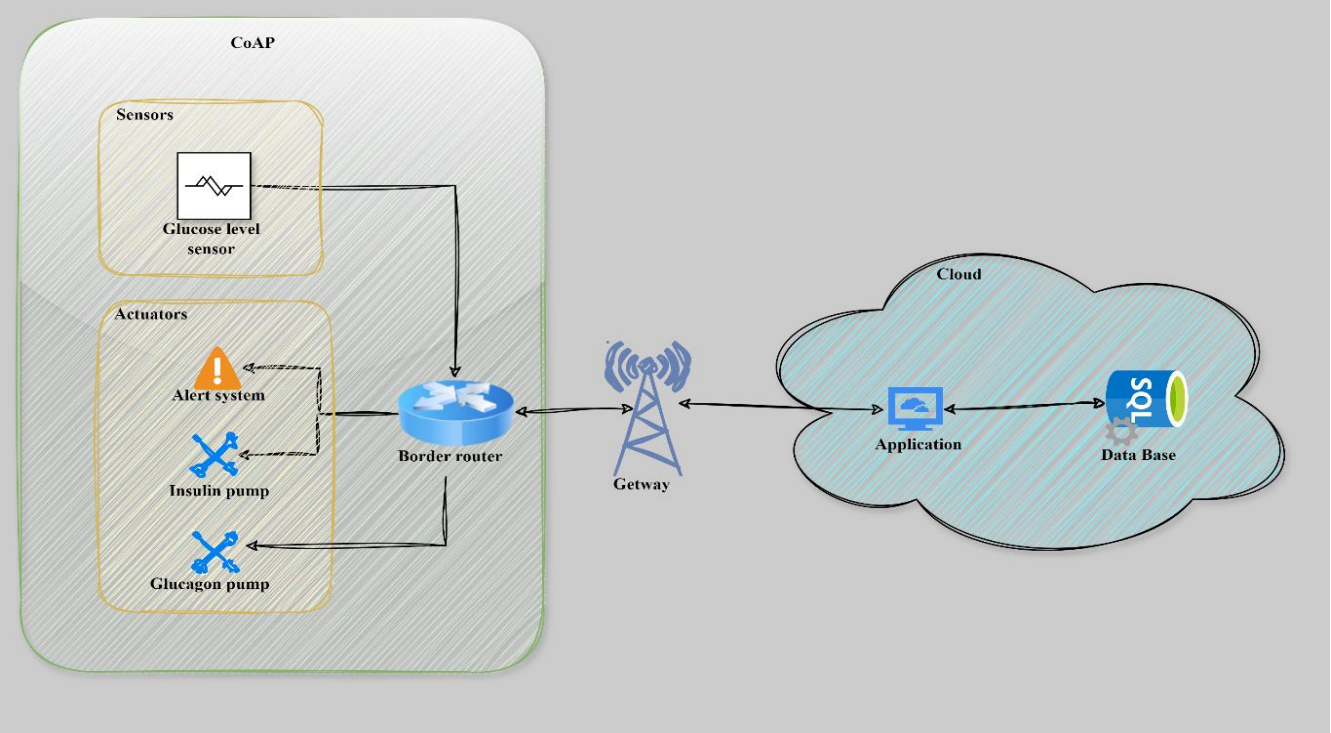


Figure 4: This picture depicts the CoAP protocol architecture in WSN

Each end device in the CoAP network exposes resources through a server. In this case, the Glucose Level Server acts as the primary server, providing addresses to access resources on each end device. The cloud application statically registers these addresses, enabling it to access resources directly.

The following code snippet demonstrates how the **glucose_monitoring_server (CoAP server)** initializes and activates each CoAP resource, making them accessible to the network

```
//Activate CoAP resources
coap_activate_resource(&res_glucose_sensor, "glucose/level");
coap_activate_resource(&res_insulin_control, "glucose_control/insulin");
coap_activate_resource(&res_glucagon_control, "glucose_control/glucagon");
coap_activate_resource(&res_alert_control, "glucose_control/alert");
```

4.3.2. MQTT Protocol

Since CoAP is used in managing continuous glucose monitoring, the system also uses the Message Queuing Telemetry Transport (MQTT) protocol for the critical task of cardiovascular data transmission. MQTT's inherent reliability, low latency, and publish/subscribe communication model make it well-suited for real-time health data that demands immediate attention.

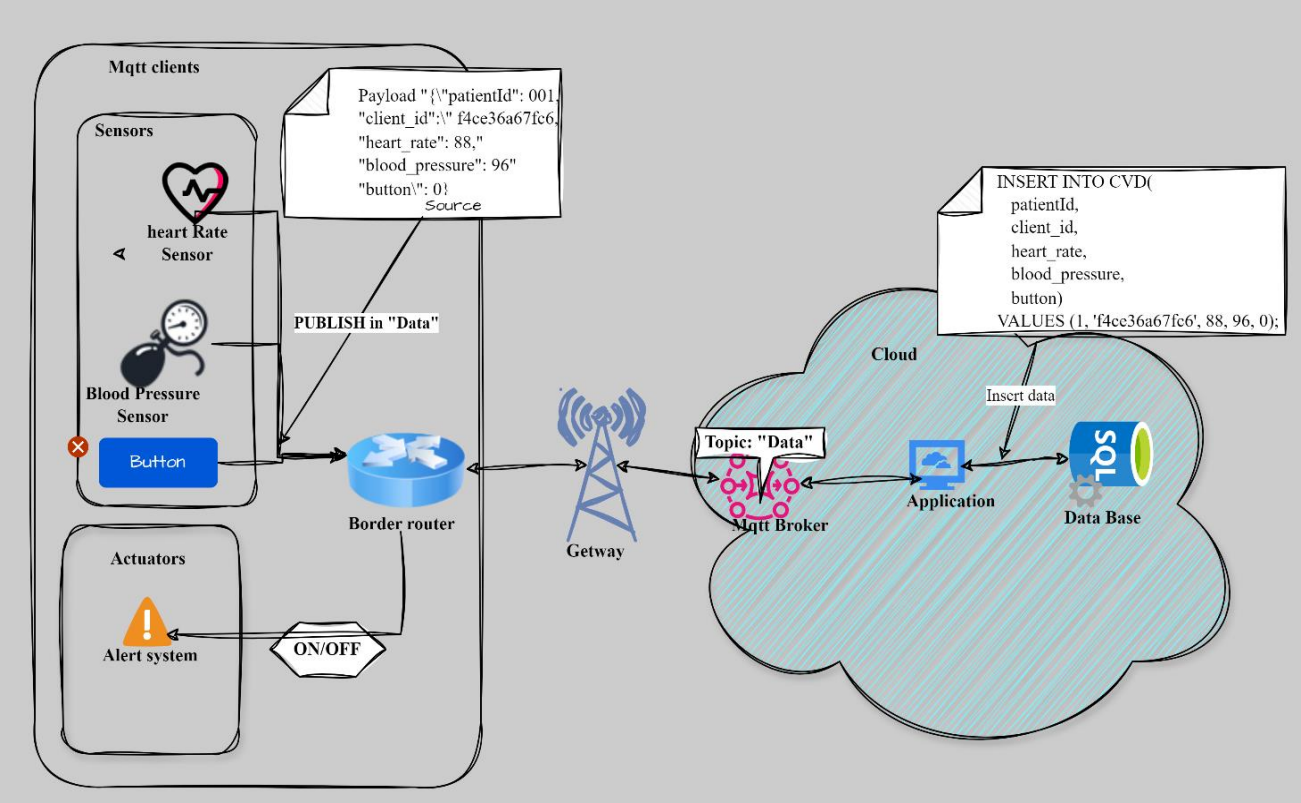


Figure 5: WSN architecture of the usage of Mqtt Protocol

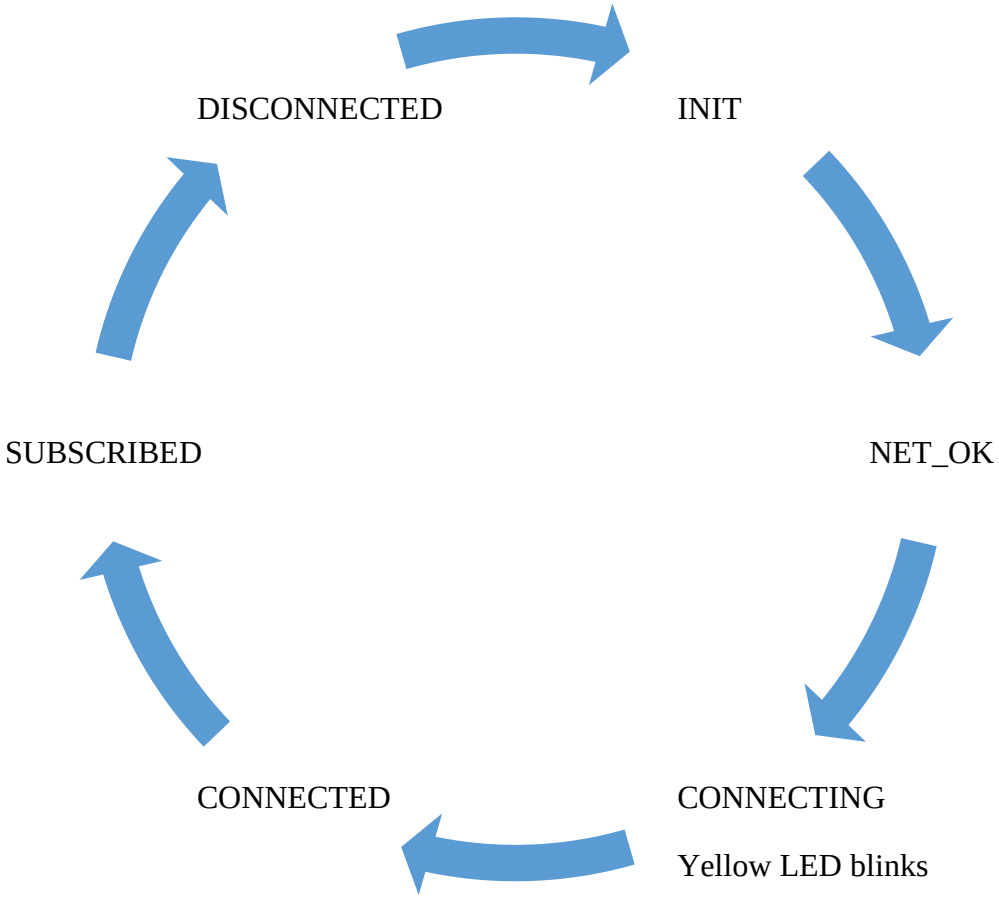
Figure 5 depicts that, the heart rate sensor, blood pressure sensor, and the emergency button act as MQTT clients, they publish data on a specific topic to the MQTT broker. The broker, a central message hub, efficiently transmits this data to the subscribed client application in the cloud and vice-versa. To achieve this I used **Device3** as MQTT Sensor Node, By exploiting the many network libraries that Contiki-NG has, the client node can be easily configured, as the network stack is handled in the lower layers.

Mqtt Sensor node (Device 3)

This Sensor node is implemented with state machine with 6 states:

1. INIT: this state is the initial state of the sensor node. Upon powering up, the node initialize itself internally for the connectivity.

2. NET_OK: in this state, the node attempts to establish a connection to the network
3. CONNECTING: in this state, the sensor node tries to establish a connection with MQTT Broker. At this state, Yellow Led blinks.
4. CONNECTED: in this state the sensor node is ready to subscribe to topics and start exchanging messages
5. SUBSCRIBED: in this state, the sensor node receives messages on the topic it subscribed. In this system, the sensor node to the "Emergency Aler" topic
6. DISCONNECTED: this state is when a loss of connection with the MQTT broker. in this case, the sensor node is programmed to automatically attempt reconnection using the "mqtt_disconnect(&conn)" command.
7. the sensor node tries to reconnect. I used this code to reconnect,"mqtt_disconnect(&conn);"



RPL Border Router and MQTT Broker

The Border Router used in the project is the rpl-border-router example provided by Contiki-NG. No modifications were done in the device, as it is only intended to provide the WSN access to the external network. For the MQTT Broker system, the Mosquitto MQTT Broker software was used, as part of the project specifications. The broker then automatically sets the topic where the sensor nodes are publishing and retransmit the messages the subscribers. In the project deployment the only subscriber to this topic will be the Cloud Application.

4.4. Data Storage (MySQL Database)

The system has MSQL database to store patient data for both real-time monitoring purposes and medical analysis. The database is designed with two primary tables.

4.4.1. Glucose monitoring table

```
mysql>  SELECT * FROM glucose_monitoring;
+-------+-----------+---------------+---------------------+---------------------+
| id    | patientId | glucose_level | incoming_timestamp  | record_created      |
+-------+-----------+---------------+---------------------+---------------------+
| 12418 |         1 |            97 | 2024-06-22 09:47:28 | 2024-06-22 09:47:27 |
| 12419 |         1 |           247 | 2024-06-22 09:47:33 | 2024-06-22 09:47:33 |
| 12420 |         1 |           237 | 2024-06-22 09:47:39 | 2024-06-22 09:47:38 |
| 12421 |         1 |           227 | 2024-06-22 09:47:44 | 2024-06-22 09:47:43 |
| 12422 |         1 |           217 | 2024-06-22 09:47:49 | 2024-06-22 09:47:49 |
| 12423 |         1 |           207 | 2024-06-22 09:47:55 | 2024-06-22 09:47:54 |
| 12424 |         1 |           187 | 2024-06-22 09:48:00 | 2024-06-22 09:47:59 |
| 12425 |         1 |           177 | 2024-06-22 09:48:05 | 2024-06-22 09:48:05 |
| 12426 |         1 |           167 | 2024-06-22 09:48:10 | 2024-06-22 09:48:10 |
| 12427 |         1 |           157 | 2024-06-22 09:48:16 | 2024-06-22 09:48:15 |
| 12428 |         1 |           147 | 2024-06-22 09:48:21 | 2024-06-22 09:48:21 |
| 12429 |         1 |           137 | 2024-06-22 09:48:27 | 2024-06-22 09:48:26 |
| 12430 |         1 |           127 | 2024-06-22 09:48:32 | 2024-06-22 09:48:31 |
| 12431 |         1 |           117 | 2024-06-22 09:48:37 | 2024-06-22 09:48:37 |
| 12432 |         1 |           107 | 2024-06-22 09:48:43 | 2024-06-22 09:48:42 |
+-------+-----------+---------------+---------------------+---------------------+
15 rows in set (0.00 sec)

mysql>
```

This table is a structured repository for glucose monitoring data collected by the Smart Diabetic Patient Monitoring System.

4.4.2. Cardiovascular monitoring table

```
mysql>  SELECT * FROM cardiovascular_monitoring;
+------+-----------+-------------+------------+----------------+--------+---------------------+---------------------+
| id   | patientId | client_id   | heart_rate | blood_pressure | button | incoming_timestamp  | record_created      |
+------+-----------+-------------+------------+----------------+--------+---------------------+---------------------+
| 7125 |         1 | f4ce36a75fc6 |         92 |            130 |      0 | 2024-06-22 09:50:39 | 2024-06-22 09:50:38 |
| 7126 |         1 | f4ce36a75fc6 |         92 |            130 |      0 | 2024-06-22 09:50:39 | 2024-06-22 09:50:38 |
| 7127 |         1 | f4ce36a75fc6 |        103 |            112 |      0 | 2024-06-22 09:50:44 | 2024-06-22 09:50:43 |
| 7128 |         1 | f4ce36a75fc6 |        103 |            112 |      0 | 2024-06-22 09:50:44 | 2024-06-22 09:50:43 |
| 7129 |         1 | f4ce36a75fc6 |         99 |            131 |      0 | 2024-06-22 09:50:49 | 2024-06-22 09:50:48 |
| 7130 |         1 | f4ce36a75fc6 |         99 |            131 |      0 | 2024-06-22 09:50:49 | 2024-06-22 09:50:48 |
| 7131 |         1 | f4ce36a75fc6 |        116 |             87 |      0 | 2024-06-22 09:50:54 | 2024-06-22 09:50:53 |
| 7132 |         1 | f4ce36a75fc6 |        116 |             87 |      0 | 2024-06-22 09:50:54 | 2024-06-22 09:50:53 |
| 7133 |         1 | f4ce36a75fc6 |        109 |             92 |      0 | 2024-06-22 09:50:59 | 2024-06-22 09:50:58 |
| 7134 |         1 | f4ce36a75fc6 |        109 |             92 |      0 | 2024-06-22 09:50:59 | 2024-06-22 09:50:58 |
| 7135 |         1 | f4ce36a75fc6 |        114 |            124 |      0 | 2024-06-22 09:51:04 | 2024-06-22 09:51:03 |
| 7136 |         1 | f4ce36a75fc6 |        114 |            124 |      0 | 2024-06-22 09:51:04 | 2024-06-22 09:51:03 |
| 7137 |         1 | f4ce36a75fc6 |         59 |            134 |      0 | 2024-06-22 09:51:08 | 2024-06-22 09:51:08 |
| 7138 |         1 | f4ce36a75fc6 |         59 |            134 |      0 | 2024-06-22 09:51:08 | 2024-06-22 09:51:08 |
| 7139 |         1 | f4ce36a75fc6 |         69 |            114 |      0 | 2024-06-22 09:51:14 | 2024-06-22 09:51:13 |
| 7140 |         1 | f4ce36a75fc6 |         69 |            114 |      0 | 2024-06-22 09:51:14 | 2024-06-22 09:51:13 |
+------+-----------+-------------+------------+----------------+--------+---------------------+---------------------+
16 rows in set (0.00 sec)

mysql>
```

This table is a structured repository for Cardiovascular Monitoring data collected by the Smart Diabetic Patient Monitoring System.

5. Result Demonstration

This Smart Diabetics Monitoring system carries out mainly two main tasks. One controls glucose levels and the other monitors cardiovascular diseases. We will see in details below:

5.1.Glucose level monitoring

The Glucose Level controlling is implemented by cooperating CoAP Sensor node, cloud application and database. The sensor node exposes resources of sensors and actuators such as alert resources, insulin/glucagon resource and glucose sensor resource. The cloud application also access these resources via internet to collect sensor reading data and to implement controlling commands. The cloud application uses GET method to access and collect glucose levels data from the sensor node. And also uses PUT method to send commands for to the actuators. from the sensor node via the internet and store them into the database. to the Border Router and registers itself with the Collector using a CoAP request. Following successful registration, the sensor exposes a resource that accepts GET requests for data retrieval and responds using JSON format. To simulate the sensor's response to a changing environment, the sensor also reacts to PUT requests, simulating glucose level changes when the insulin or glucagon actuators are activated.

In essence, the Cloud application interacts with the sensor node in two primary ways: One is Data Retrieval when the collector sends GET requests to read the sensor's current glucose level readings and gets responds using JSON format. The application stores this data in the database. The second is Actuator Simulation, when the controller application sends PUT requests to the sensor node to simulate the effects of insulin and glucagon pumps, allowing for system testing and validation. For clarification check the diagram in Figure 6.

Actuators (insulin pump, glucagon pump, Emergency Alert)

The actuators play a crucial role in regulating glucose levels and triggering emergency alerts when necessary. They are integrated with the Glucose Level Server as resources, enabling coordinated responses. Upon registration with the Collector, each actuator exposes a resource that accepts PUT requests to activate or deactivate its functionality.

The Cloud application assumes responsibility for sending these activation/deactivation commands based on real-time glucose level data. If the Collector detects that the glucose level has crossed predefined critical thresholds, it sends the appropriate commands to activate the insulin or glucagon pumps. Similarly, it manages the Emergency Alert activation based on critical health metrics or user-initiated requests.

Actuators are simulated using LEDs that indicates the status of the Actuators:

- GREEN LED: Indicates that all actuators are currently inactive, signifying a normal and stable health status.
- YELLOW LED: Illuminates when either the insulin pump or the glucagon pump is actively regulating glucose levels.
- RED LED: Signals the activation of the Emergency Alert system, indicating a potentially

critical situation

To sum up, the actuators help to regulate blood glucose levels and trigger emergency alerts. The integration with the Collector through CoAP resources and LED status indicators facilitates the simulation how the system works.

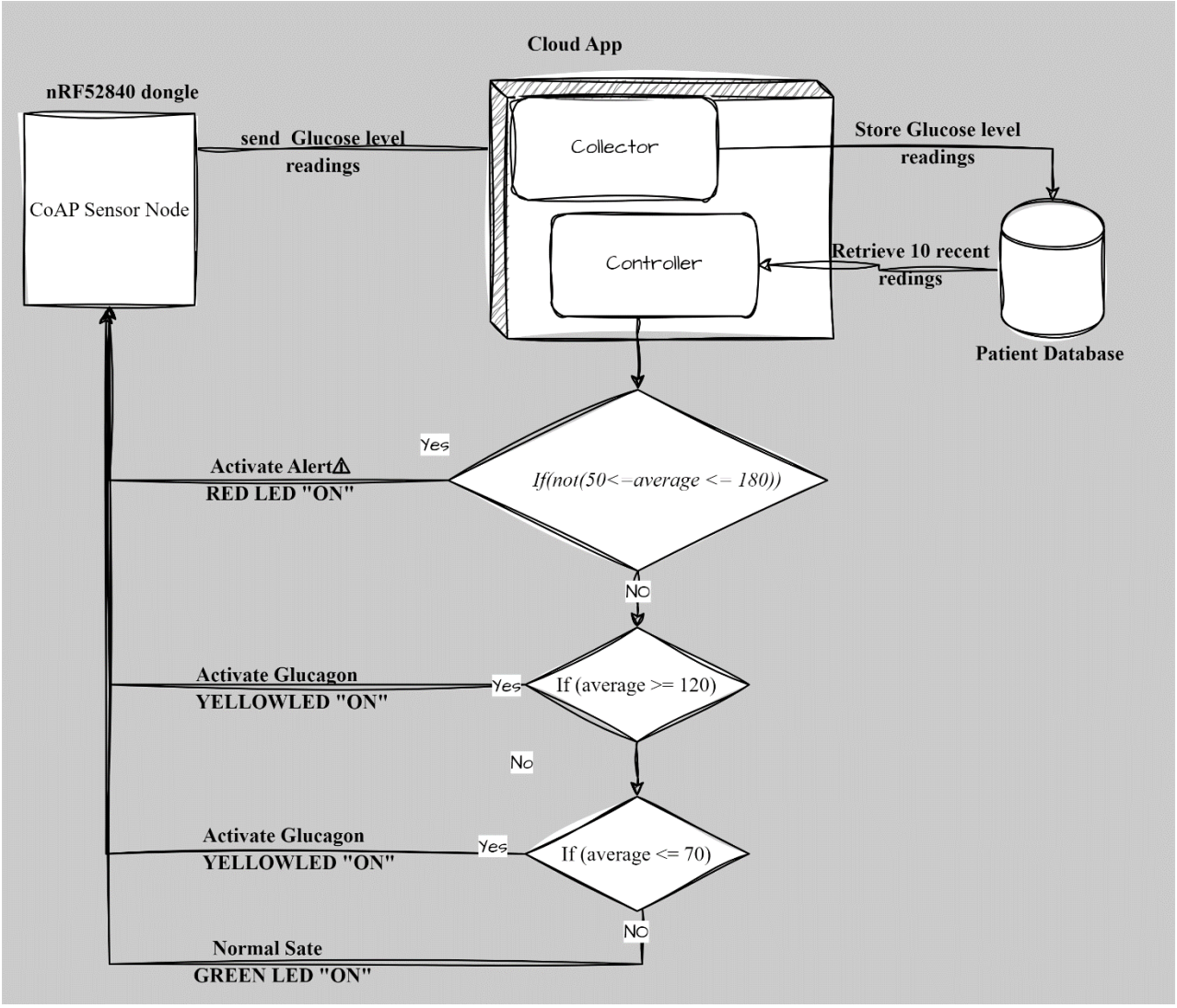


Figure 6: Diagram depicts how the smart diabetic monitoring system is controls the glucose level

5.2.Cardiovascular monitoring

The cardiovascular monitoring also implemented using Mqtt sensor node, Mqtt broker and like glucose monitoring module it uses cloud application and database. In this case, the communication among the sensor node and the application is via intermediary called Mqtt broker. They both publish and subscribe to the topics in the broker. For example the sensor node subscribes to the "emergency alert" topic to receive command public by the cloud application.

Emergency Alerts: The system can automatically trigger alerts based on the calculated average exceeding thresholds. Overall, the MQTT network expands the system's capabilities by monitoring vital signs and facilitating emergency responses.in this demo as we see in figure 7, the emergency actuator for this demo is Red LED. if the calculated average exceeds set threshold the RED LED is "ON" instead of Emergency alert.

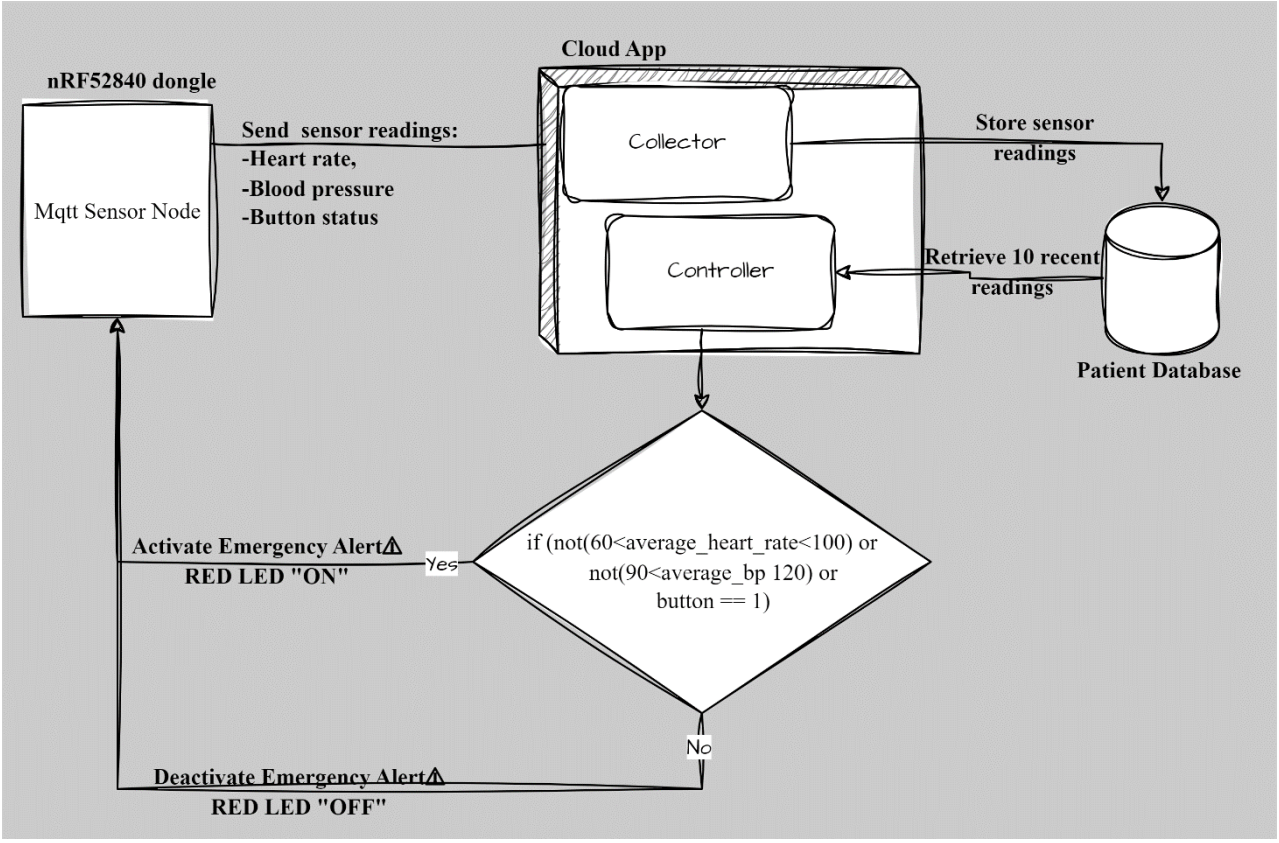


Figure 7: Diagram depicts how the smart diabetic monitoring system controls the cardiovascular disease

6. Conclusion

The Smart Diabetic Patient Monitoring System is a personalized and proactive healthcare for diabetic patients. This ingenious system leverages the power of Internet of Things (IoT) protocols like CoAP and MQTT to provide real-time, accurate monitoring of critical health parameters like blood sugar levels, heart rate, and blood pressure.

The system's architecture is a testament to its efficiency. It utilizes a dual-network approach, with each network optimized for specific functionalities. The CoAP network prioritizes crucial tasks like continuous glucose monitoring and actuator control, enabling precise insulin and glucagon medication through automated pumps. This ensures timely medical intervention, potentially preventing life-threatening situations.

Meanwhile, the MQTT network focuses on the continuous monitoring of cardiovascular health, a crucial aspect for diabetic patients who are at a higher risk of heart-related complications. The system can detect potential issues before they escalate by constantly collecting and analyzing this data. All collected data is carefully stored in a central MySQL database. This not only facilitates comprehensive data analysis for long-term monitoring but also allows for real-time identification of concerning trends in vital signs. Furthermore, the system can generate alerts for emergency situations, significantly enhancing patient safety.

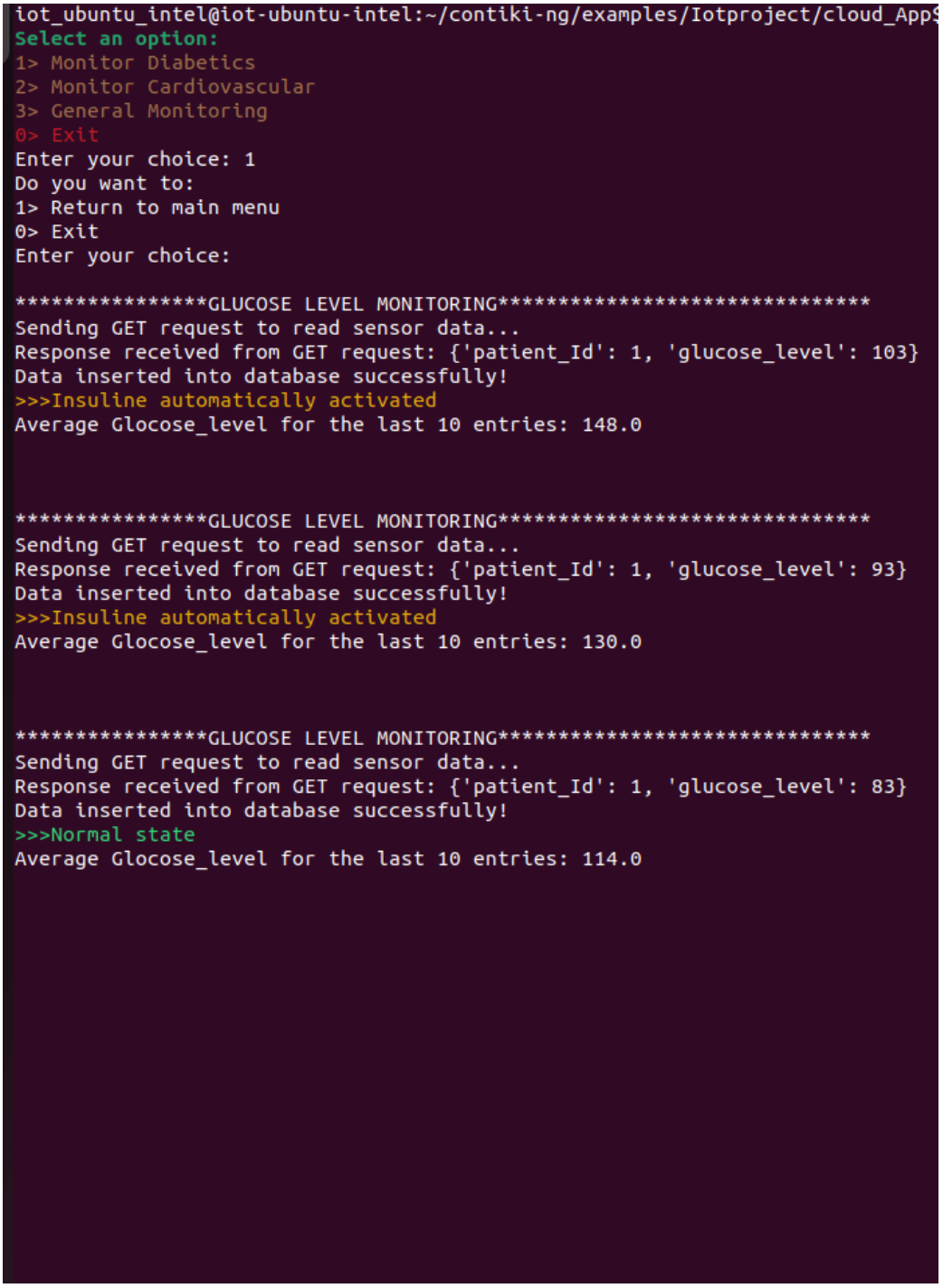
Through advanced data analysis and real-time monitoring, the Smart Diabetic Patient Monitoring System goes beyond simply managing diabetes; it empowers patients to take control of their health and improve their overall well-being. This project serves as a shining example of how technology can be harnessed to deliver personalized healthcare solutions that are both proactive and preventative.

7. Appendices

```
 # Check emergency
average_bp = get_average_blood_pressure(patientId)
average_heart_rate = get_average_heart_rate(patientId)
if ((average_heart_rate > 100 or average_heart_rate < 60) or (average_bp > 120 or average_bp < 90) or (button == 1)
    app_mqtt_client.publish("Emergency_Alert", payload="ON")
    print(f"Alert is activated!")
    if (average_heart_rate > 100 or average_heart_rate < 60):
        print(f"Average Heart Rate for the last 10 entries: {average_heart_rate}")

    if ((average_bp > 120 or average_bp < 90)):
        print(f"Average Blood Pressure for the last 10 entries: {average_bp}")

    if (button == 1):
        print("The Emergeny button is pressed.")

    alert_active = True
    time.sleep(2)
else:
    if alert_active:
        print("Turning alert OFF")
        app_mqtt_client.publish("Emergency_Alert!\n", payload="OFF")
        alert_active = False
    else:
        if (not alert_active):
            print("normal/stable state \n")
```

This Python code checks the cardiovascular status by setting heart rate and blood pressure thresholds.

Cloud application controlling terminal:



Cardiovascular monitoring terminal

```
*****************Cardiovascular Monitoring************************
Received message on topic: Heart/Data
Data inserted into database succefully!
>>>Normal state


***************GLUCOSE LEVEL MONITORING*****************************
Sending GET request to read sensor data...
Response received from GET request: {'patient_Id': 1, 'glucose_level': 158}
Data inserted into database successfully!
>>>Alert is calling!
Average Glocose_level for the last 10 entries: 203.0


*****************Cardiovascular Monitoring************************
Received message on topic: Heart/Data
Data inserted into database succefully!
>>>Normal state


***************GLUCOSE LEVEL MONITORING*****************************
Sending GET request to read sensor data...
Response received from GET request: {'patient_Id': 1, 'glucose_level': 138}
Data inserted into database successfully!
>>>Alert is calling!
Average Glocose_level for the last 10 entries: 192.0


*****************Cardiovascular Monitoring************************
Received message on topic: Heart/Data
Data inserted into database succefully!
>>>Normal state


***************GLUCOSE LEVEL MONITORING*****************************
Sending GET request to read sensor data...
Response received from GET request: {'patient_Id': 1, 'glucose_level': 128}
Data inserted into database successfully!
>>>Alert is calling!
Average Glocose_level for the last 10 entries: 181.0


*****************Cardiovascular Monitoring************************
Received message on topic: Heart/Data
Data inserted into database succefully!
>>>Normal state
```