UNIVERSITÀ DI PISA

IOT Project

Giulio Bello, Andrea Ruggieri

September 9, 2023

Chapter 1

Introduction

According to National Center for Health Statistics, these are the data gathered in past years about heart attack diseases in United States:

- About 695,000 people in the United States died from heart disease in 2021—that's 1 in every 5 deaths
- Coronary heart disease is the most common type of heart disease, killing 375,476 people in 2021
- Heart disease cost the United States about $239.9 billion each year from 2018 to 2019. This includes the cost of health care services, medicines, and lost productivity due to death

And about Heart Attack:

- In the United States, someone has a heart attack every 40 seconds
- Every year, about 805,000 people in the United States have a heart attack

Due to these impressive stats, project's aim is provide a smart solution to supervise the status of a patient under heart attack risk and following this idea three main parameters to be controlled were chosen:

- **Heartbeat**: irregular heartbeat is a crucial warning for people under heart attack risk in different situations
  - Decreased Heart Rate (Bradycardia): heart's ability to pump up blood is might be compromised
    Heartbeat value under 20bpm
  - Irregular Heartbeat (Arrhythmia)
  - Increased Heart Rate (Tachycardia): heart is beating too fast trying to pump more blood to compensate decreased oxygen delivery to the body's tissues.
    Heartbeat value over 250bpm
- **Oxygen**: is a value between 0 and 100, representing the percentage of oxygen in blood which is important because can provide insights into the heart's ability to pump oxygenated blood to the body's tissues. A drop

in oxygen saturation levels could indicate that the heart is not effectively pumping blood, which could be a sign of worsening cardiac function.
A value under 50% beware a potential critical situation

- **Troponin**: is a protein released into the bloodstream when there is damage to the heart muscle. It is highly specific to cardiac injury and is considered one of the most reliable markers for diagnosing heart attacks. Monitoring troponin levels allows healthcare providers to detect even small amounts of heart muscle damage

Based on heartbeat, oxygen and troponin values, the application must turn the activators on or off to take care of the patient in two different ways: automatically and manually. The former was thought as main feature, making easier the firsts steps to avoid an heart attack, while the second mode is useful for border-line situations in which, for example, doctors can decide to use (or not) a specific cure for the patient even is not the solution suggested by the data retrieved at the moment.

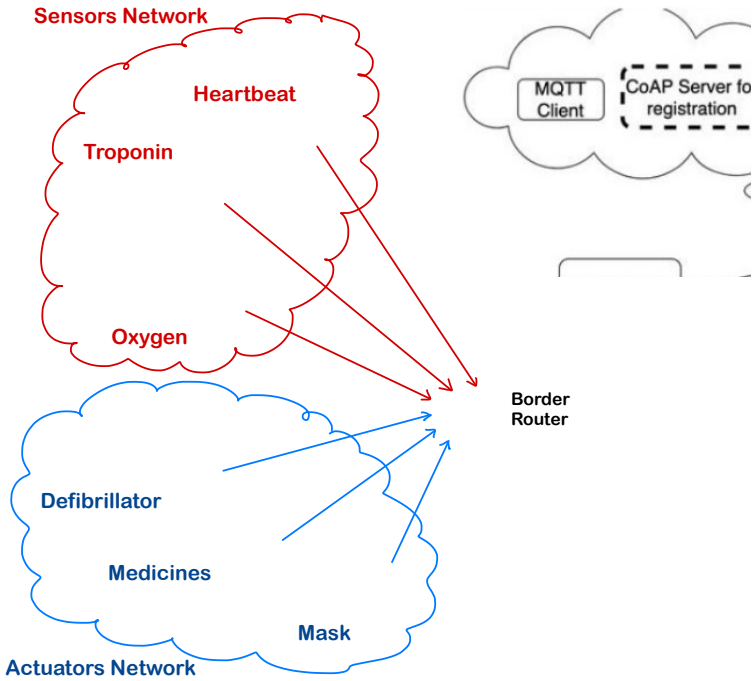

Figure 1.1: Overall application architecture

Chapter 2

Architecture

This version of the application, which will be accurately explained later, mainly consists of

- MQTT sensors
  MQTT is a publisher-subscriber topic-based application protocol. Using TCP, allows the information exchange between sensors and application thanks to a broker.
  MQTT sensors are:
  1. Heartbeat
  2. Oxygen
  3. Troponin
- CoAP actuators
  CoAP is an application protocol similar to HTTP, working over UDP by default, to provide methods GET, POST, PUT and DELETE. It exposes functionalities provided by things as resources that can be discovered and accessed in the same way browsers access HTTP resources (identified by URI).
  CoAP was used for the following actuators:
  1. Mask:
  2. Medicine
  3. Defibrillator
- Application side with Database, also using Grafana for data for a smart data monitoring

All the tools, IDE and programme languages used for the project were inserted into the following list.

- Virtual Box to let Ubuntu 18.04 run smoothly
- Contiki: open-source OS for resource-constrained networked embedded systems
  - Programmed in C

  - Support for several platforms
  - Event-driven kernel, protothreads (event management support and sequential flow of instructions that can be interrupted to wait for events or conditions)
  - Dongle and Launchpad support
  - Cooja, a sensor simulation environment
- Visual Studio Code, provides support for the language needed
  - C: Border Router, CoAP actuators and their resources, MQTT sensors
  - Java: Application, request from application for operations on DB
  - MySQL: creation and access to DB
  - Bash: Script for smart start of the application
  - JSON: data encoding
- Seven sensors, each one is a Dongle Nrf52840 even programme can also works with Launchpad and other sensor types
  - One Border Router
  - Three sensors for heartbeat, oxygen, troponin
  - Three sensors for the actuators mask, medicine and defibrillator

The whole application code is available at **https://github.com/AndreaRuggieri/IOT**

2.1 MQTT Sensors

The application was tested using three sensors:

1. Heartbeat sensors, which publish on topic "cardio", has following values:
   - Heartbeat value is good: $30 < HB < 220$
   - Heartbeat value must be controlled for possible dangerous (CTR_HB): $20 < HB < 30$ or $220 < HB < 230$
   - Heartbeat value is dangerous (DNG_HB): $HB \leq 20$ or $HB \geq 230$
2. Oxygen sensors, which publish on topic "oxygen", has following values:
   - Oxygen value is good: $30 < OX < 220$
   - Oxygen value must be controlled for possible dangerous (CTR_OX): $50 \leq OX < 60$
   - Oxygen value is dangerous (DNG_OX): $OX \leq 50$
3. Troponin sensors, which publish on topic "troponin", has following values:
   - Troponin value is good: $0.01 < TRP < 0.07$
   - Troponin value must be controlled for possible dangerous (CTR_TRP): $0.07 \leq TRP < 0.1$
   - Troponin value is dangerous (DNG_TRP): $TRP \geq 0.1$

Due to the lack of real measurements stuff, as heart rate monitor, pulse oximeter and cardiac troponin test, values are not gathered from real patient's body but automatically generated to simulate as good as possible a realistic situation.

2.2 CoAP Actuators

CoAP actuators expose resources used after a CoAP call and are the sensors corresponding to real actions a med equipment can perform trying to solve patient's issues.
Following scheme was adopted to simulate real hospital situation:

1. No issues: All values are in good ranges, there are no even possible dangerous at the moment
2. Oxygen_value in CTR_OX:
   Actuator Mask ON LOW
3. Oxygen_value in DNG_OX:
   Actuator Mask ON HIGH
4. Cardio_value in CTR_HB XOR Troponin_value in CTR_TRP:
   Actuator Medicine 1 ON LOW
5. (Cardio_value in CTR_HB AND Troponin_value in CTR_TRP) OR
   (Cardio_value in DNG_HB XOR Troponin_value in DNG_TRP):
   Actuator Medicine 1 ON HIGH
6. (Cardio_value in DNG_HB XOR Troponin_value in DNG_TRP)
   AND (Cardio_value in CTR_HB XOR Troponin_value in CTR_TRP):
   Actuator Medicine 2 ON LOW and Actuator Defibrillator ON LOW
7. (Cardio_value in DNG_HB AND Troponin_value in DNG_TRP):
   Actuator Medicine 2 ON HIGH and Actuator Defibrillator ON HIGH

2.3 Application and Database

2.3.1 Application

Application logic was wrapped into *Coordinator* and *RemoteControlApplication*, which have both threads starting from *Main* class.

- *Coordinator* handles two different sides
  1. Extending *MqttCallback*, exposes a CoAP Resource in order to perform MQTT sensor registration with methods *connectionLost*, *deliveryComplete* and *messageArrived*, invoked every time a message arrived, which receives as in input a String, representing the topic, and an MqttMessagge object, containing sensor id and relative value. If the message is correctly received, data are stored into the Database.
  2. Extending *CoAPResource*, may override handleGET, handlePUT, handlePOST and handleDELETE but for this aim only the last two were needed. *MyCoapResource* class exposes a resource is always active while the application is on and allows the actuator to sign in (POST) to the database. There's also the DELETE method implement to remove a useless actuator from DB upon client request.
- *RemoteControlApplication* perform periodic data retrieval from database:

- Access database to retrieve sensor's values
- Analyses values and for every situation (already explained just above) the corresponding action to be performed
- Every time an actuator have to change its status, a put request is called from *ActuatorClient* class, a CoAP Client instance which only perform the request to turn ON or OFF the actuator, but the code corresponding to the action is in the specific actuator class
- The class also contains
  * Threshold values for every sensor and relative methods to change these
  * Code to access DB every time a status change is requested or an additional information is needed
  * An internal Hash Map to simplify handling process for active or not actuators

A Command Line Interface is also provided to handle these requests:

1. Retrieve information about the status of the actuators
2. Set the dangerous thresholds
3. Activate a specific actuator for a specific patient

This last option was implement thinking about a common situation in which the Med equip makes a decision despite data retrieved by sensors but using other parameters for a more complete analysis of the patient's status.

2.3.2 Database

Database class provides methods to handle the database. By default, it contains following tables:

1. Actuators: empty as the application starts, an actuator must be registered in there to be used and cancelled from table if is not used. Every entry is composed by the actuator IP, its type and status
2. Sensors: a table for each sensor containing id, value and timestamp in every entry

Please note that the id in sensor field is a `varchar` starting with a char, which can be "o", "t" or "c" where every letter stands for a different sensor, followed by tree numbers representing the patient identifier.
Database provides methods to:

- Create a connection between application and DB
- Insert, replace or delete an actuator
- Retrieve actuator IP having corresponding type
- Retrieve actuator status (useful for command line interface option)
- Update actuator status
- Retrieve sensor values

2.3.3 Grafana

Grafana is an analytics web platform that allows to analyze data from various sources. Available at http://localhost:3000/ address by web browser, due to the simply usage because only requires database connection, provides many different ways to monitor data using both simple user interface or writing complex queries looking for specific data.

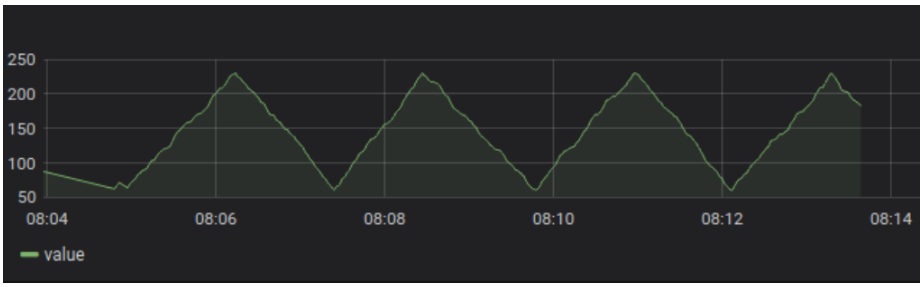

Figure 2.1: Heartbeat values over time

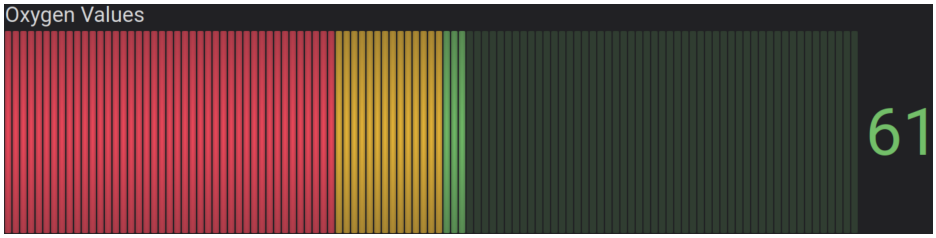

Figure 2.2: Oxygen values over time

These are two of many possible example of data visualization for a smart monitoring of sensor values.

Chapter 3

Workflow and future updates

3.1 Workflow

This section shows the application workflow, which works as it follows:

1. Sensors start metering simulation and send values to `Coordinator.java`
2. `Coordinator.java` upload values received on DB which stores ID, timestamp and value for each one
3. Actuators registration on DB via `Coordinator.java`, which stores IP, type and status for each one
4. `RemoteControlApplication.java` starts periodic data retrieval and analysis from DB
5. Thanks to the sensors value analysis, the application decides to turn ON or OFF actuators
6. During the entire application run, there's a Command Line Interface available

3.2 Future updates

3.2.1 Sensors Power Management

Considering power management, one future update for energy saving and harvesting might be the introduction of an algorithm which decrease number of detection if the values sensed are quite far from threshold value.

3.2.2 ECG sensor

Since ECG is one the most important value for heart health status monitoring, one suitable idea to improve heart-attack finding efficiency might be use another sensor for ECG. Unlike other sensors used for the project which all have easy

and fast way to examine heartbeat, oxygen and troponin, ECG exam is quite complicated so the sensor can give much less values to analyse and write the code to simulate ECG value once in a while. ECG analysis requires a classifier, which takes as input current values and gives as output positive value if these are ok, zero in other case.