Cardiac Arrest Detector (Group 5)

Kyle Hanks
Electrical and Computer Engineering Department
George Mason University
Fairfax, Virginia, 22030
khanks@gmu.edu

Kyle Loyd Guthrie
Electrical and Computer Engineering Department
George Mason University
Fairfax, Virginia, 22030
kguthrie@gmu.edu

Shamili Mownika Tetali
Electrical and Computer Engineering Department
George Mason University
Fairfax, Virginia, 22030
stetali@gmu.edu

Sunanda Roy
Electrical and Computer Engineering Department
George Mason University
Fairfax, Virginia, 22030
sroy9@gmu.edu

Abstract— **Internet of Things is considered one of rapidly advancing innovative technologies that has influenced several areas, modelling a shift in traditional healthcare systems. Proliferation of IoT devices has enabled several opportunities to improve quality of life and providing methods to reduce mortality rates. Healthcare Domain is one important area requiring high standards of accuracy and precision in determining key metrics. The objective of this project is to design a Proof-of-concept prototype based on research with Connected Healthcare multisensory IoT system that extracts human body vital parameters and derive an early alarm of fatal cardiac arrest. The design goal primarily focuses interfacing various sensors and implementing data collection software methodologies by means of considering cost-effective techniques to potentially bring the technology closer to users and aid larger human population.**

Keywords—IoT, Arduino, Bluetooth, pulse, temperature, Android, Firebase

I. INTRODUCTION

Advancements in various fields of modern Computing and Communication technology has embraced new ideas in IoT design implementations. These innovations provided extensive opportunities to improving how traditional systems operated for betterment of developing enhanced lifesaving solutions. The importance of Healthcare and especially Cardiac monitoring systems has come a long way with extensive research put into characterizing cardiac abnormalities and human organ functions and tremendous studies have proved that timely monitoring of various vital parameters will have a lifesaving outcome. With the evolution of IoT it was possible to bring in these technologies together and implement medical solutions for proactive detection of health issues.

The high-level design idea on this project is to develop an embedded system using Arduino Nano and interfacing it to various sensors like pulse and temperature which are basic parameters to detecting anomalies in cardiac function. Essentially, the design aspect would leverage built-in low power Bluetooth communication method to integrating sensor data into android based application which would then establish a secure channel to cloud based solution for data acquisition and analysis.
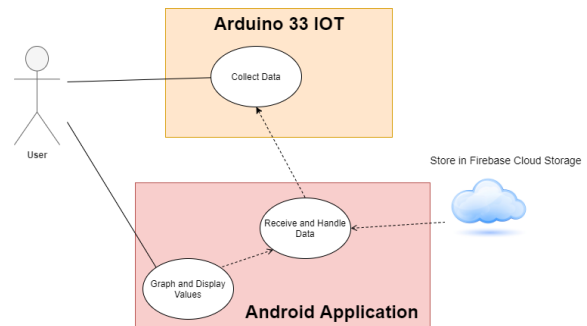
II. USE CASE DESCRIPTION

Fig. 1. Block diagram of key components

The use case in Figure 1 above serves as a framework for how we should approach designing our system. In general, use cases in IoT devices provide developers with clear goals for what the IoT device should be

capable of. Without this, it is hard to execute a clear vision for the design of an IoT system. Our use case displays our two main devices, the Arduino Nano 33 IoT and the Android Application, and shows how they interact with the user and with a cloud storage database.

As pictured, the Arduino takes sensor readings from the user and transmits them via Bluetooth to the Android application, which then processes cloud storage and user display. The Arduino's sensor integration process is where one of our main challenges related to the use case occurred. We attempted to use two different temperature sensors when implementing this design, however as stated in a later section, the sensor's readings were a bit inaccurate in that they had trouble sensing specifically body heat over the ambient temperature of the room. Between this issue and the need for an accurate pulse sensor, we were able to use the use case to inform our decision in determining which components to select and how to approach our solution.

III.   RELATED WORKS

Over the past few years, several related IoT devices have offered solution to monitor, report, and display data relevant to preventing or predicting heart troubles in a user. In 2019, a wearable device was suggested that uses embedded ECG and temperature sensors to monitor the condition of the wearer's heart. [1] This data is then sent via Bluetooth to a smartphone to classify whether the wearer has any present abnormalities in the condition of their heart. The Android platform performs a quantitative heart rate analysis and displays the real-time plots of the ECG signal and body temperature. The app can trigger a warning if the ECG signal or body temperature reaches above a critical threshold. The app also has functionality for creating an account with useful patient data that can be expanded to include the ability for healthcare workers to remotely view the data collected by the app. This design served as a basis for our implementation. It has some similar aspects and functionalities but uses slightly different techniques. Over time, our implementation could be expanded to be more cost and energy efficient utilizing a different Arduino board and sensors than the one proposed in this paper.

In 2018, another wearable IoT device to detect cardiac issues was proposed by Ashwini K and Komala K. [2] The main goal of this implementation is to increase the accuracy of the ECG readings by increasing the

number of sensors worn. This device uses the AD8232, a three-lead ECG sensor that allows for an ultralow power analog-to-digital converter or an embedded microcontroller to acquire the output signal easily. The proposed model would also be able to give a voice output alerting the wearer to potential cardiac issues while simultaneously sending this data to a caretaker along with the location of the user if a serious issue occurs. The main shortcoming of this implementation is that the AD8232 sensor is not suited for constant wearing/monitoring throughout everyday situations. If a developer wanted to expand on the goal of making ECG readings more reliable using multiple sensors, they could start by improving placement and selection of the wearable sensors in the monitoring system. This approach could also be used to improve the reliability of our own device in future work.

IV.   IOT TECHNICAL SOLUTION

We propose building a hardware prototype along the lines of the experimental setup described in [1]. The goal of this prototype IoT device is to capture the instantaneous heart rate and body temperature readings of a human and perform computations which would result in the detection of an imminent danger of cardiac arrest in that person.
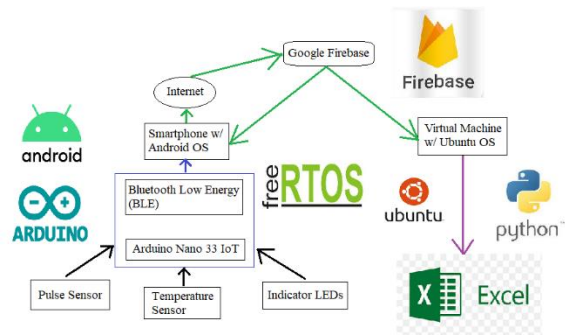
A.   System Architecture

Fig. 2.   Data flow diagram of entire solution

We split the entire architecture of the solution into five essential components:

1) *Sense*
2) *Compute*
3) *Transmit*
4) *Store*
5) *Retrieve*

The above components are described below in greater detail.

1) *Sense*

The following sensors have been used to capture user data:

a) *Tragoods pulse sensor/heart rate monitor*
ARDUINO LIBRARY: PulseSensor Playground
This module [2] can be used to detect our heart rate by clipping it to sensitive areas like an earlobe or fingertip. In our setup, we connect it to a fingertip. The captured signal is a raw voltage which can be plotted on any serial plotter to show the heart rate varying over time.

Fig. 3.   Components of pulse sensor package

The front side of the sensor (with the heart logo) makes the contact with the skin. This front side has a small round hole, which is where the LED (reverse mounted at the back side) shines through from the back. Just below this round hole is a little square, which is an ambient light sensor.
The LED shines light into the fingertip or earlobe, or other capillary tissue, and sensor reads the light that bounces back.

b) *FTCBLock TMP36 temperature sensor*
ARDUINO LIBRARY: TMP36
This module [3] is a low voltage and high precision temperature sensor, which is a possible candidate for reading body temperatures.

CHALLENGE: This sensor does not give stable and accurate values obfuscating our future calculations.

Fig. 4.   TMP36 temperature sensor

2) *Compute*
For detecting a heartbeat, we set a threshold value and classify any reading above that as a heartbeat and below as none.

CHALLENGES: Obtaining readings in the expected range required setting the correct threshold value and sampling interval, so the correct number of heartbeats are detected per unit time interval. We solved this problem by observing the signal waveform to detect threshold and surveyed online implementations for setting the sampling interval. As a part of this verification, we added support for computing the Beats Per Minute (BPM) periodically during a user session. Our obtained BPM values were in the expected normal range (~60).

The temperature readings can be obtained using the associated library functions or manually calculated (ensure Arduino Nano 33 IoT operating voltage is factored into calculations).

3) *Transmit*
The computed sensor data are transmitted to the receiver device (a smartphone) using the built-in Bluetooth Low Energy (BLE) module on Arduino Nano 33 IoT device.

4) *Store*
The smartphone device (running Android OS) stores these sensor readings in real-time into a cloud database (Firebase).

5) *Retrieve*
The data stored in the cloud database is queried back by our Android app with the future goal of plotting them into a user-friendly waveform for the convenience of the end user.

V. IMPLEMENTING BLE COMMUNICATION AND FIREBASE STORAGE

A. Overview of BLE

BLE is a wireless personal area network technology which provides a low power and low-cost interface for data transmission. Using BLE, data can be transmitted up to a range of 100 m and Android apps can communicate to edge devices having stricter power requirements such as proximity sensors, heart rate monitors, and fitness devices.

B. Overview of Firebase storage

Firebase is a cloud platform developed by Google for developing mobile and web applications. It provides end-users a means to store their data on a cloud server in real-time and perform different analytics on user data. Firebase provides numerous platforms for supporting a wide range of user requirements like performing user authentication, storing data into a real-time database, setting up a cloud server, executing cloud functions and implementing machine learning applications.

C. Implementing BLE

In this project, BLE has been used to transmit sensor data from an Arduino Nano 33 IoT acting as the peripheral device (client) to a Samsung Galaxy M11 smartphone acting as the central device (server). Arduino's FreeRTOS libraries have been used to implement multithreaded reading of sensor values.

1) CLIENT side

Sensor readings from pulse sensor and temperature sensor are captured continuously and sent in parallel over two respective threads to the server end. These 2 sensor readings are transmitted as Arduino BLE objects of type *BLEIntCharacteristic* to the Android app listening on server side.

2) SERVER side

The server side comprises of an Android app developed in-house using Android Studio software on Windows 10 platform. This app captures the exported Arduino BLE characteristic readings and converts them to Android objects of type *BluetoothGattCharacteristic* for subsequent processing.

D. Implementing Firebase storage

In this project, the Cloud Firestore database (server) provided by Firebase has been used to receive real-time data from an Android app running on a smartphone (client). This stored data is also read back by the Android app for possible future applications.

1) CLIENT side

Due to the asynchronous nature of the Android API (which reads the incoming sensor data over multiple threads), the data are stored to Cloud Firestore storage asynchronously as well.

2) SERVER side

For each continuous execution of the experimental setup, all the incoming data from Android data are stored as a single collection of multiple documents. Here, each document represents an individual sensor reading.

E. Benefits of this communication interface

The BLE communication protocol allows for the transmission of sensor data from client to server with extremely low latency, which may not be possible if implemented using other communication protocols like Wi-Fi, Bluetooth (BLE's ancestor), etc. In addition, the low power consumption of BLE meets the requirement of a robust IoT device driven by power supplies of low range.

F. Algorithm

1) Edge device (Arduino Nano 33 IoT)

Input: Sensor readings: pulse (p_raw) and temperature (t_raw).

Output: BLE characteristic objects of sensor readings: p_char and t_char.

Procedure:

1. Initialize a BLE service object (srv).
2. Initialize BLE characteristic objects (p_char and t_char).
3. Add p_char and t_char to srv and advertise srv.
4. Create separate RTOS threads for reading respective sensor values (p_thread and t_thread).
5. Optionally create RTOS thread for monitoring tasks usage.
6. Schedule all RTOS tasks for continuous reading of sensor values at fixed intervals.

2) Gateway device (Smartphone with Android OS)

Input: Number of samples (set of pulse and temperature readings) to be captured from peripheral device (Arduino) - N.

Output: Cloud Firestore collection (user session) of multiple documents (sensor readings).

Procedure:

1. Establish connection to the peripheral device.
2. Initialize a counter for tracking the number of samples read.
3. Convert each (Arduino) BLE characteristic object to a (Android) BluetoothGatt characteristic object.
4. Establish connection to the Firebase instance.
5. On successful completion of a read characteristic function, convert to desired format and store into the cloud database.
6. Final collection consists of N sensor values.

VI. RESULTS

A. *List of deliverables*

A. Arduino code and serial log
B. Android app and logcat log
C. Demo videos
D. Fritzing diagram
E. Google Firebase to MS Excel conversion
    a. Python script to extract data from the cloud and plot them into a graph.
    b. Google cloud authentication key (JSON file).
    c. Input text file (contains names of collections to be plotted).
    d. Example output graphs for N = 5 and 100.
    e. Associated LINUX commands

B. *Selected outputs*

```
*****************************************************
Free Heap: 9280 bytes
Min Heap: 9280 bytes
*****************************************************
Task              ABS              %Util
*****************************************************
Task Monitor      156              <1%
IDLE              9896176          64%
Task Temperatur   31124            <1%
Task Pulse        5409532          35%
Tmr Svc           22               <1%


*****************************************************
Task            State   Prio    Stack   Num     Core
*****************************************************
Task Monitor    X       1       136     3
IDLE            R       0       126     4
Task Pulse      B       3       194     1
Task Temperatur B       2       188     2
Tmr Svc         B       2       118     5

*****************************************************
[Stacks Free Bytes Remaining]
Thread Pulse: 194
Thread Temperature: 188
Monitor Stack: 136
*****************************************************
```

Fig. 5. Example of RTOS task usage

```
1   19:53:54.207 -> BPM  60
2
3   19:53:58.798 -> Body temperature is now: 63
4   19:53:58.798 -> Body temperature updated
5
6   19:54:01.054 -> Heart rate is now: 508
7   19:54:01.054 -> Heart rate updated
8
9   19:54:00.753 29093-29112/com.example.ble_test
    I/MainActivity$gattCallback: Debug: Temperature value 1: 63
10  19:54:01.877 29093-29112/com.example.ble_test
    I/MainActivity$gattCallback: Debug: Pulse signal value 1: 508
11
12  19:54:02.313 29093-29093/com.example.ble_test
    I/MainActivity$gattCallback$1$onCharacteristicRead: Debug:
    Temperature DocumentSnapshot added with ID:
    7LoQ6bODaz70zewXq6ly
13  19:54:02.455 29093-29093/com.example.ble_test
    I/MainActivity$gattCallback$1$onCharacteristicRead: Debug:
    Pulse DocumentSnapshot added with ID: 6RwzU86kuvoNmWqtQYxe
14
15  19:54:07.565 29093-29093/com.example.ble_test
    D/MainActivity$gattCallback$1$onCharacteristicRead: Debug:
    Pulse 6RwzU86kuvoNmWqtQYxe => {pulse signal=508, sample
    index=1}
16  19:54:07.572 29093-29093/com.example.ble_test
    D/MainActivity$gattCallback$1$onCharacteristicRead: Debug:
    Temperature 7LoQ6bODaz70zewXq6ly => {sample index=1, body
    temperature=63}
```
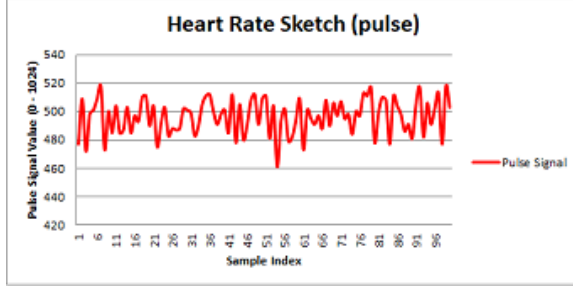
Fig. 6. Data flow of a sample sensor reading

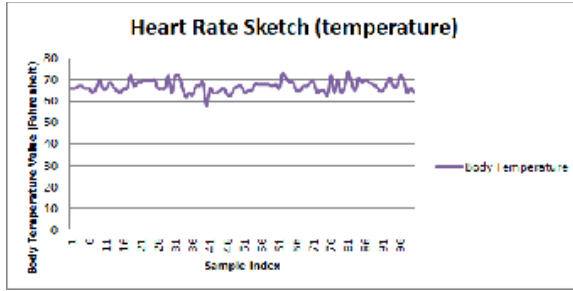Fig. 7.   Pulse signal variation (N = 100)

Fig. 8.   Body temperature variation (N = 100)

C. Future scope

The accuracy of sensor readings can be improved with further market survey of IoT devices, although the base IoT device of Arduino Nano 33 IoT will always be preferred over its higher footprint counterparts.

Advanced data analytics using state-of-the-art machine learning techniques can be performed on the output waveforms, to detect abnormalities in the human body's vital parameters.

VII.   CONCLUSION

As healthcare needs are increasingly concerning and monitoring human body vitals becomes an important paradigm, we attempted to demonstrate the IoT capabilities in the ehealth domain. This experiment primarily focused on design and development of an integrated smart IoT system to predict and monitor health abnormality in a user. We managed to create a low power communication channel between smart IoT device and smartphone application and were successfully able to connect to Google firebase to store the values to derive the value prospects of this study.

VIII.   DISCUSSION

A. RTOS task usage

The sample output in Fig. 5 shows that the major chunk of non-idle execution time of the microcontroller is consumed by the pulse sensor reading task (35%). While temperature sensor reading task consumes negligible share of the microcontroller's execution time. This observation directly correlates to the assigned priorities of these sensor reading tasks, pulse reading task having higher priority than temperature reading task. On the contrary, we find that the monitor task consumes the most amount of stack space followed by temperature reading task. The least amount of stack space is occupied by the pulse reading task, indicating an efficient software implementation of their driver libary.

B. Data flow of sensor readings

Fig. 6 shows the typical flow of information from the edge device to the cloud storage via the gateway device. This output shows one of the two implemented ways of retrieving stored data from Firebase. Each entry in the log is taken from either the serial output of Arduino IDE or logcat output of Android Studio IDE while the desired number of sensor readings are being captured. The individual entries are arranged in increasing order of their timestamps to signify the continuity of data flow from one node to another node in the system architecture.

NOTE: To avoid transmission of redundant data, the sensor readings are transmitted via BLE to smartphone only if there is a change in read value. This is expected to optimize the power consumption of the resource constrained Arduino Nano 33 IoT device.

Line 1 shows the current value of BPM. Lines 3 and 4 show an updated temperature reading. While lines 6 and 7 show an updated pulse reading. Lines 9 and 10 show the values received by the Android App. Lines 12 and 13 show these values being written to Firebase. While lines 15 and 16 show these vales being retrieved from Firebase.

C. Pulse signal and body temperature variation

The waveforms shown in Fig. 7 and Fig. 8, show the pulse signal and body temperature varying within a stable range of low and high values, indicating a healthy human heart rate and body temperature.

REFERENCES

1. AKM Jahangir Alam Majumder, Yosuf Amr ElSaadany, Roger Young, Donald R. Ucci, "An Energy Efficient Wearable Smart IoT System to Predict Cardiac Arrest", Advances in Human-Computer Interaction, vol. 2019.
2. Ashwini K, Komala K, "Wearable Heart Attack Detector using IoT", International Journal of Engineering Research & Technology. 2018.
3. Pulse Sensor Playground. [https://github.com/WorldFamousElectronics/PulseSensorPlayground]
4. TMP36 Temperature Sensor. [https://learn.adafruit.com/tmp36-temperature-sensor]
5. The Ultimate Guide to Android Bluetooth Low Energy. [https://punchthrough.com/android-bleguide/]
6. Android Developer Guides. [https://developer.android.com/guide]
7. Add Firebase to your Android project. [https://firebase.google.com/docs/android/setup]
8. Cloud Firestore. [https://firebase.google.com/docs/firestore]
9. Things You Should Know About Bluetooth Range [https://blog.nordicsemi.com/getconnected/things-you-should-know-about-bluetooth-range]
10. Python Client for Google Cloud Firestore [https://googleapis.dev/python/firestore/latest/index.html]