# ECG Meet BLE: A Method for Sending Realtime Analog Data Over BLE Advertisements

Alexander Redding
alexanderredding2022@u.northwestern.edu

## 1 Introduction

According to the 2019 Heart Disease and Stroke Statistics Update by the American Heart Association, nearly half of all U.S. adults have some type of cardiovascular disease [1]. This problem isn't going to disappear over night and our healthcare system must adapt to our country's growing medical needs.

Sometimes a patient may be experiencing symptoms of cardiovascular disease that appear randomly (ex, heart arrhythmias). Conditions like these may not immediately present themselves when the patient goes to see their healthcare provider. In that situation, a healthcare provider may direct the patient to wear an event monitor [2]. An event monitor is a type of portable ECG that records your heart activity whenever you are experiencing symptoms. The patient typically presses a button on the event monitor to begin a recording; the monitor is then returned to their healthcare provider for analysis.

Event monitors are the standard of care for random heart conditions and I think the standard could be set much higher. My project is an experiment focused on this initiative. An event monitor should be low power, low cost, and immediately transmit its data in realtime to a healthcare provider. I practically envisioned a "Life Alert but better." Following this, I developed a method of transmitting ECG data over Bluetooth Low Energy advertisements. My method could also have the potential to be extended to transmitting other forms of analog data.

## 2 Background

BLE advertisements were chosen as the transmission medium because as the name Bluetooth Low Energy implies, they are incredibly low energy. BLE radios are also low cost which would allow an event monitor utilizing BLE to be affordable. My project primarily focused on tackling the constraints of BLE.

A BLE advertisement is basically an efficient "hello" message to any nearby devices that may be listening. This "hello" contains some sort of information about the advertiser. The efficiency of sending advertisements comes at multiple costs; the two main ones concerning the viability of my project are the relatively low range of advertisements and their tight payload size.

In commercial application, I believe range limitations could be tackled through better utilization of BLE scanners already in wide circulation: smartphones.

A small payload budget doesn't have a solution quite as simple. Disregarding any headers and addresses in a BLE advertisement, there are only 31 bytes available to the user. To put this into perspective, a single high resolution ECG value could be within the integer range of 1000 - 2000 (values are later converted to millimeters for relative analysis). To represent a number this large, a 16 bit integer would have to be used in order to maintain appropriate byte alignment. A 16 bit integer has the size of 2 bytes, meaning that if you could use all 31 bytes in a payload, you would only be able to fit about 15 full raw values. According to a study conducted by researchers at Dong-a University, ECG readings should be sampled at 250-Hz minimum for heart rate variability analysis [3]. That means one would have to take and transmit 250 ECG readings per second. Coupled with the 31 byte payload size, this is very inefficient if only 15 values can be sent at once (every 20ms at minimum).

Sending raw ECG values is a naive approach and we can get a better idea how to improve once we start taking some notes on ECG data set trends.

## 3 Design and Implementation

Raw analog values can be noisy and "jumpy". Filtering the data can allow for much more consistency regarding the rate of change between consecutive data points. If ECG values are changing at a consistent rate, perhaps it could be possible to only transmit the changes between values instead of the values themselves (given that the changes are significantly smaller in size). This would allow for a much higher throughput than with sending the full raw values.

Additionally, the idea of focusing on value changes could be extended if the data set tends to have a low standard deviation. If so, we could also also possibly focus on the difference between filtered ECG values and a known mean; thus potentially lowering the amount of information we have to send per value as well.

For development I used the DFRobot *Analog Heart Rate Monitor Sensor*, an Arduino Nano for initial ECG tests, two nRF52840DK boards for BLE communication, and a human test subject (myself). I also focused my project's goal away from commercial application and instead limited it to achieving efficient transmission of ECG data from a single BLE advertiser to a single scanner.

An Arduino Nano was used for the sake of simplicity in reading analog values from the heart sensor and building an initial understanding of a raw ECG data set. I also created a Python script for calculating a small statistics suite on the data and a second generated set consisting of the differences between neighboring values.

Inspired by a prior research study which found that weighted moving average filters can be effective in removing noise from an ECG, I also created another data set on which I applied an exponential moving weight average (to better account for sudden changes) [4]. For the filter, I just used the existing Arduino EWMA library.

I ran the statistics suite on both a raw and filtered ECG data set of myself and obtained the following:

| Raw ECG Dataset (10s, 200-Hz, Arduino Nano) | | | | | |
|---|---|---|---|---|---|
| | **Min.** | **Median** | **Max.** | **Mean** | **STDev** |
| **Original** | 209 | 356 | 541 | 347.46 | 45.79 |
| **Filtered** | 319 | 344.91 | 367.97 | 347.24 | 7.31 |

| Differences Between Neighboring Values | | | | | |
|---|---|---|---|---|---|
| | **Min.** | **Median** | **Max.** | **Mean** | **STDev** |
| **Original** | -108 | 0 | 113 | 0.03 | 30.07 |
| **Filtered** | -4.56 | 0.07 | 3.6 | 0.02 | 1.38 |

The filter helped significantly with reducing the range and variability regarding the differences between neighboring values. An integer difference with the range of [-5, 4] can be contained with only four signed bits (with a range of [-8, 7]).

| | | | |
|---|---|---|---|
| -8 | 4 | 2 | 1 |

**Four Signed Bits**

I then continued the statistical evaluation by interfacing the heart sensor and filter with an nRF board. Similar results were obtained, however the raw values had to be scaled by 1/6 in order to achieve these similar statistics (very little resolution was lost).

| Raw ECG Dataset (1min, 250-Hz, nRF52840DK) | | | | | |
|---|---|---|---|---|---|
| | **Min.** | **Median** | **Max.** | **Mean** | **STDev** |
| **Filtered** | 274 | 316 | 340 | 313.07 | 11.83 |

| Differences Between Neighboring Values | | | | | |
|---|---|---|---|---|---|
| | **Min.** | **Median** | **Max.** | **Mean** | **STDev** |
| **Filtered** | -4 | 0 | 7 | -0.0001 | 1.31 |

Having certified that I could limit the differences between neighboring ECG values to a size significantly smaller than their original, I designed a payload structure to utilize these neighbor differences.

## 3.1 Payload Structure

In order to maximize the number of goodput per payload, I decided to forego the flags data type at the beginning in order to save 3 extra bytes. Although possibly frowned upon, the Bluetooth Core Specification states that "If the Flags AD type is not present in a non-connectable advertisement, the Flags should be considered as unknown and no assumptions should be made by the scanner"; thus making its exclusion perfectly legal since the advertiser is non-connectable [5].

The only Bluetooth data type I used was Manufacturer Specific Data and I used the company ID 0xFFFF (allocated for internal testing). After accounting for the Length, Type, and ID fields; there are 27 bytes left for our use. I used the 27 bytes to create the following ECG payload structure:

| **Count** | **1st Value** (dif. From 300) | **2nd** Value | **3rd** Value | ... | **51st** Value |
|---|---|---|---|---|---|
| 1 byte (Unsign.) | 1 byte (Sign.) | 4 bits (Sign.) | 4 bits (Sign.) | | 4 bits (Sign.) |

### 3.1.1 Count Variable

The count byte (unsigned 8 bit integer) is to keep track of the ECG payload ordering and it also helps with accounting for lost payloads. This is done by the scanner keeping a global count variable. The count variable gets incremented each time a new payload arrives with a larger count than the global variable. If the new payload has count greater than the global + 1, then the scanner outputs 51 "null" values per each lost payload. This shows the user that some payloads were lost and it also maintains the right time scale for following data. Both count variables also appropriately account for unsigned integer overflow and the count resets to 0 and continues.

### 3.1.2 First Value

The first ECG value in the payload is contained by a signed 8 bit integer that represents its difference from 300. I chose 300 because it's very close to the expected average of the dataset and it's also just a pleasing even number. The scanner takes this difference, adds 300 to it, and then outputs the full ECG value. The full value is encoded vice-versa (full - 300).

Given the range of a signed 8 bit int [-128, 127], the first value has an effective range of [172, 427]. This well encapsulates the expected minimum and maximum range of the filtered nRF dataset [274, 340].

### 3.1.3 2nd, 3rd, …, 51st Values

The following values are each represented as their respective difference from the previous value. This can be achieved with only 4 signed bits per value because the difference between two neighboring values is expected to fall within the range of [-4, 7], fitting snugly within the [-8, 7] range of 4 signed bits.

The 2nd through 51st values are decoded by looping through all of the values and keeping track of the previous value. The previous first value gets initialized as the first value + 300. The second value is the previous value + the second value difference. The new value gets outputted and the loop continues. The encoding works very similarly as well, with the difference of the previous value - next value being saved instead.

Using a 4 bit signed integer was tricky since that's not a standard C data type. Instead, I created a byte-sized Payload_t struct that contains two signed 4 bit integers using bit fields to specify their width. Consequently, two encoded values have to be inputted and removed from a payload buffer at once. Thankfully, this doesn't present any problems.

## 3.2 Task Pipeline

The tasks of both the advertiser and the scanner can be summarized with their respective pipelines.

**Advertiser Pipeline**

Get raw heart sensor data, apply EWMA filter, insert into a full value buffer

Remove values from full value buffer, encode values following ECG payload structure

Get encoded payload, update advertising payload to transmit new data

**Scanner Pipeline**

Receive advertisement, check count, output null values for lost payloads

Decode payload, output full values

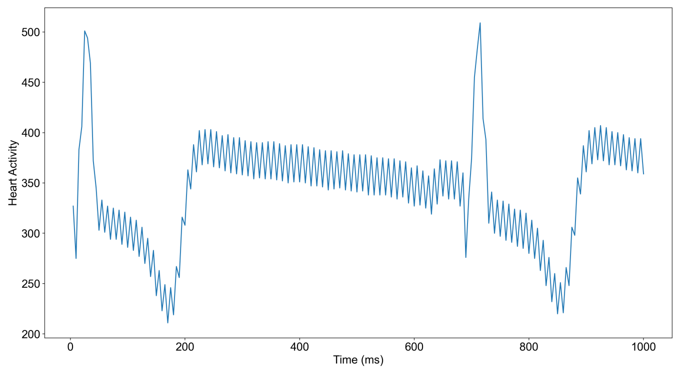
## 3.3 External Libraries

I created two external libraries for the nRF boards. One library is a simplified port of the Arduino EWMA library. The other library contains the Payload_t definition, a struct to contain relevant buffers (actual values, payload) and their sizes, and functions representing each task in the pipelines. I initially believed that I would have to use an OS (Zephyr) to implement the pipelines, but creating organized functions help to significantly simplify task execution and I realized that an OS was unnecessary. All of the code I used can be found on this repository: https://github.com/alexredd99/ECG-Meet-BLE.

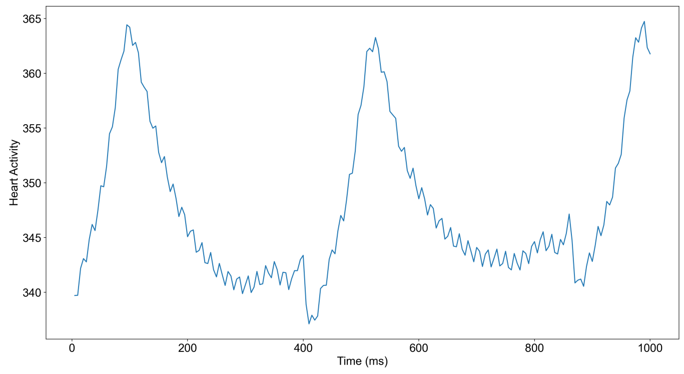
## 4 Evaluation

In the most basic sense, my realtime ECG transmission method worked. Here are four examples comparing the resolution of four different ECGs. Each sample contains one second of heart activity.

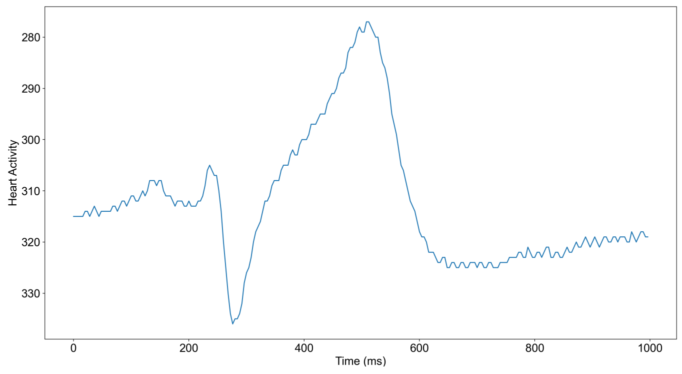
**Raw ECG (200-Hz, Arduino Nano)**



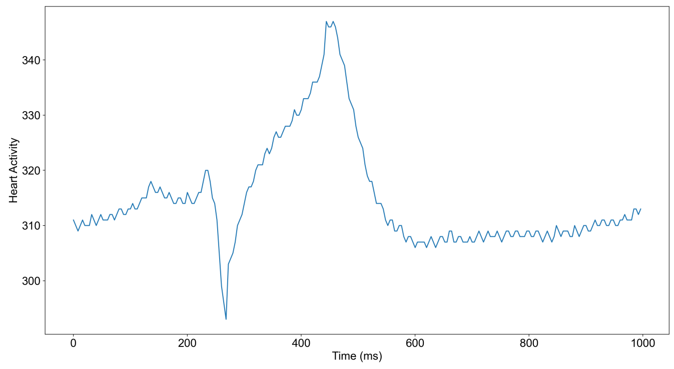
**Filtered ECG (200-Hz, Arduino Nano)**



**Filtered ECG (250-Hz, nRF52840DK)**

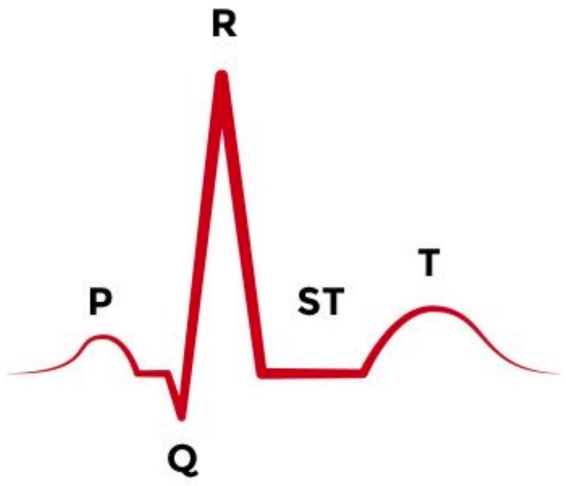


**Received Filtered ECG (250-Hz, nRF52840DK)**



Each valley and peak seen here corresponds to a single heartbeat. This diagram from the Advanced Cardiovascular Life Support Medical Training website highlights the important features of an ECG [6]:

**ECG Diagram**



I'm not a doctor so I'm not going to attempt to explain what each feature is, but these features coupled with the frequency of heartbeats can tell a healthcare provider a significant amount regarding a patient's heart health.

Most importantly, these features can be seen on every ECG I recorded. The last three ECGS have an inverted P wave but I think that's due to user error (putting on the ECG electrodes incorrectly). The first two ECGS also look somewhat difference then the last two because my heart rate happened to be faster that day for some reason.

The last ECG is one that was decoded and outputted by the nRF BLE scanner. Since a single ECG payload contains 51 values and we're sampling at 250-Hz (250 samples per second), that one second of ECG data is the combination of almost 5 distinct payloads.

## 4.1 Reliability

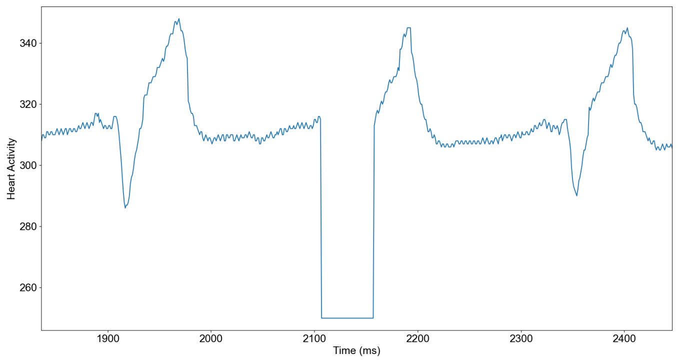
Given the high amount of payloads needed to transmit ECG data in realtime, the question of reliability becomes incredibly important. Obviously we want to limit the amount of lost payloads as much as possible, but it isn't the end of the world if a few get lost. What matters most is that a healthcare provider can study the general trend of their patient's heart activity. If a disease is afflicting one's heart, the symptoms will more than likely show up in multiple heartbeats (not just one).

Still, the necessity of high throughput creates concern. In order to maximize reliability I decided to focus on having low advertisement intervals in order to retransmit payloads as much as possible, instead of focusing on sending repeated data. An ECG payload contains 51 values and each value is recorded 4ms apart. Thus, it takes 204ms to create a new payload. I found that using the lower advertisement interval of 21ms yielded the lowest payload loss. This gives the opportunity to send a single payload nearly ten times while it's preparing the next payload.

In an experiment in which I transmitted nearly 17 seconds of ECG data, a total of 83 distinct payloads were sent. Out of those 83 payloads, only 4 were lost. This yielded a 4.8% miss rate. 4.8% isn't perfect, but it's something I'm satisfied with. It should be noted that this experiment was conducted under default transmission settings and the advertiser and scanner were about a yard away from each other. If I had more time I'd like to experiment more with longer distances and further investigate different advertisement intervals with the addition of a value buffer in the advertiser.

Here's what a dropped payload looks like in an ECG. A null value of 250 was used.

**Example of a Dropped Payload in an ECG**



## 4.2 Energy Efficiency

In order to analyze the energy efficiency of my advertisement scheme, I utilized the Nordic BLE Power Profiler. Under default transmission settings, an nRF board can be expected to draw an average current of 444μA on advertising alone (not accounting for running the heart sensor). In terms of practical application, an ideal advertiser should have the capability of running for at least 24 hours straight on a CR2032 battery (220mAh). Nordic recommends reducing battery life expectancy by 30% to account for environmental factors. Consequently, a good heart rate sensor should draw less than 6.55mA in order to achieve a day of battery life.

## 4.3 Future

In addition to more reliability testing, I'd also like to interface a BLE scanner with something like a Raspberry Pi in order to send ECG data directly to a web server. One of the most challenging aspects of this project was displaying the received ECG data in realtime. Trying to connect a Python script to an RTT session proved to be buggy and unreliable. Instead, I collected all of my data by simply copying the values that the scanner outputted through RTT and saving them in a separate file. I'd also like to experiment with having a varying quantity of each device (multiple advertisers and multiple scanners).

In addition to not being a doctor, I'm also not an electrical engineer. However, I do think that my method of transmitting ECG data in realtime could be adapted to other sources of analog data. The only requirement in this regard is that the dataset fits the value constraints of the ECG/analog payload (either with or without filtering). Given that, the only other possible constraint would be the sampling frequency if it happens to be high.

## REFERENCES

[1] American Heart Association News. 2019. Cardiovascular diseases affect nearly half of American Adults, statistics show. (Jan. 2019). Retrieved March 16, 2021 from https://www.heart.org/en/news/2019/01/31/cardiovascular-diseases-affect-nearly-half-of-american-adults-statistics-show.

[2] John Hopkins Medicine: Event Monitor. Retrieved March 16, 2021 from https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/event-monitor.

[3] Ohhwan Kwon, Jinwoo Jeong, Hyung Bin Kim, In Ho Kwon, Song Yi Park, Ji Eun Kim, Yuri Choi. 2018. Electrocardiogram Sampling Frequency Range Acceptable for Heart Rate Variability Analysis. *Healthcare informatics research*, 24(3) (July 2018), 190-206. DOI: https://doi.org/10.4258/hir.2018.24.3.198.

[4] Xiao Hu, Zhong Xiao, Ni Zhang, Xiaoming Han. 2012. [Correction of electrocardiogram signal baseline wander based on statistically weighted moving average filter]. *Sheng Wu Yi Xue Gong Cheng Xue Za Zhi*, 29(1) (Feb. 2012), 51-4. PMID: 22404006.

[5] Supplement to the Bluetooth Core Specification. *Bluetooth SIG Proprietary*, (Dec. 2019), 12.

[6] The Basics of ECG. *ACLS Medical Training*. Retrieved March 16, 2021 from https://www.aclsmedicaltraining.com/basics-of-ecg/.