Detecting Pacemaker Failure Using Wearable Devices and a Smart Phone

Clinton Medbery, Aspen Olmsted
Department of Computer Science
College of Charleston, Charleston, USA

Abstract

In this paper we look at solutions for notifying patients and medical professionals in the event of a Pacemaker failure. HeartSync uses an iPhone to simulate Wearable Heart Rate Monitor and a Pacemaker data. HeartSync compares the data between the two sets based on the time the readings were taken. This two-factor system helps ensure that a patient's pacemaker is functioning correctly and then pushes this information to a server, where patients and healthcare professionals can view it without having to scrutinize it.

1. Introduction

People with heart problems use devices called pacemakers to stimulate their heart electrically. The pacemaker helps those with irregular or faulty heartbeats to keep a rhythmic pumping of their heart. It does this by housing a battery, a computer to keep the beats at regular intervals and monitor heartbeats, and wires that lead to different parts of the heart depending on the person's particular heart problem [1].

One of the major fears that a person, living with a pacemaker has, is a pacemaker malfunction. For example, leads can migrate, or erode [2]. In some cases, the pulse generator can malfunction. This can cause the pacemaker not to provide electric stimulus at the desired intervals, or it can keep the pacemaker from providing any electric stimulus at all. This can result in hospitalization, death, or a bevy of other medical related issues.

There are many existing applications like Remote-K-Viewer that help patients monitor their heart rate remotely through mobile devices [3]. People also use devices made by companies like Garmin, FitBit, and Jawbone to monitor their heart rate through a wearable monitor. HeartSync would monitor both devices heart rates and look for any discrepancies between the two sets of heart rate data. HeartSync could alert the user if it thinks it has detected a failure, as well as make the data available to the desired healthcare professional. This will give patients and healthcare professionals the important information that they need to make informed medical decisions.

2. Loading the control data

The application has been running on an iPhone 5S. The application uses Apple's HealthKit to store the heart rate monitor data. The user first authorizes HealthKit allowing the application to read and write heart rate beats per minute (BPM).

After authorizing HeartSync to use HealthKit, we load BPM for each minute of the day for today's date. We assume the pacemaker keeps the heart rate in a declined or relaxed state between the hours of 11 PM and 8 AM and keep the BPM at 70. The waking hours of the day are given a heart rate of 75 BPM. HeartSync talks to HealthKit, in the same way, a fitness application would, and stores the information on the phone for other HealthKit applications to access.

The pacemaker data is simulated by storing it in the iPhone's Core Data. Core Data is Apple's low-cost database for iOS applications. Both the pacemaker data and the heart rate monitor data get a heart rate entry for every minute of the day. Like the heart rate monitor data, the pacemaker data is set at 70 between 11 PM and 8 AM and 75 the rest of the day.

3. Data swap

As well as the control data, we also need some "Non-Working" data to simulate a pacemaker malfunction. We need to know what happens when things aren't working as they should. To do this, we assume that in the event of a pacemaker malfunction, the heart rate monitor would measure different BPM reading due to a more irregular heart beat caused by the malfunction.

This is done by rewriting the HealthKit data between the hours of 5 and 6 with heart rate data that is randomly generated, by the range of 70 and 80 beats per minute, simulating an irregular heart beat caused by a faulty pacemaker.

4. Data sync

After the data is loaded, we are left with a day's worth of sample heart rate monitor and pacemaker data. Our objective is to look at each BPM reading for each minute of the day and compare the two data points for any discrepancies. If the two BPM readings match, we create a new record that marks the reading as a matching record and eliminates the redundant records. Any readings that do not match are added to the array after checking the readings minute by minute.

After the readings are put through the data sync algorithm, we are left with an array or heart reading objects that have three enum flags from the three possible results of being put through the algorithm. The record could be a heart rate monitor reading, a pacemaker reading, or a reading from both. If the pacemaker and heart rate monitor are working correctly then, every reading in the array will be flagged to represent the fact that the reading came from both locations. We will know there is a discrepancy if the array contains objects or readings from either the heart rate monitor, the pacemaker, or a mix of both. When the data sync function is not on, HeartSync doesn't compare the sets of data and just loads an object for each reading into an array. The user will not be alerted to any discrepancy.

HeartSync leaves us with four possible sets of data. One scenario is that the pacemaker is working, and we have a set of data with the data sync and one without. The other scenario is that the pacemaker is not working, and once again we have a set of data with the data sync and without.

5. Preliminary results

DATA SET 1 "Working" Data without Data Sync: The first data set we tested used the sets of data where all the BPMs matched, simulating a working pacemaker. We did not run the data synchronization algorithm on this set of data. The outcome was that although the points match visually, we are essentially left with two sets of redundant data.

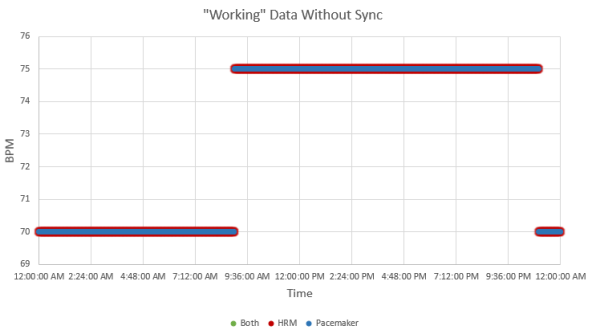

Figure 1. "Working" Data Without using data sync algorithm. All data matches and is redundant

DATA SET 2 "Working" Data with Data Sync: The second data set consists of the data where the pacemaker is working. The data is run through the data sync algorithm. The results are that the redundant data is eliminated. The new BPM readings are marked as matching both the heart rate monitor and the pacemaker.

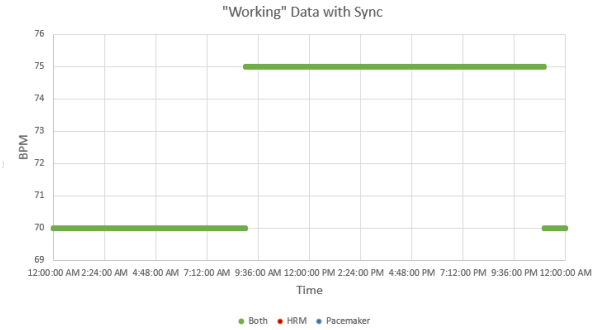

Figure 2. "Working" Data after using data sync algorithm. Since there are no discrepancies, all data is merged into one set

DATA SET 3 "Non-Working" Data without Data Sync: We have now switched out the heart rate monitor data, so there are random discrepancies between 5 and 6 PM. We once again load the data into a set without using the algorithm.

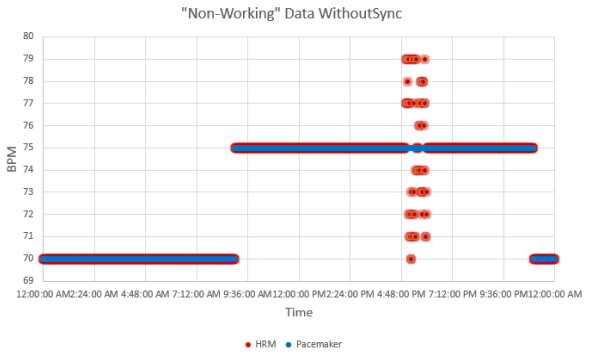

Figure 3. "Non-Working" Data without Data Sync algorithm. We can see the varying heart beats in this chart. However, all the data is being captured

DATA SET 4 "Non-Working" Data with Data Sync: The final set of data is the where HeartSync gets its main purpose. Here we are looking at a scenario where the pacemaker has failed, and a user is potentially using the app to find fault in their pacemaker. The data is merged where it matches, and the non-matching data is flagged.

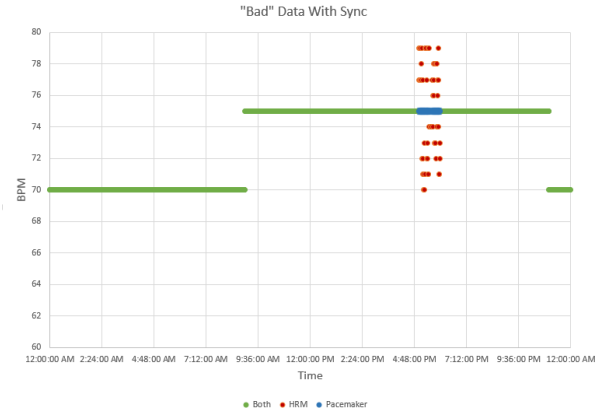

Figure 4. "Non-Working" Data with data sync algorithm. This data set leaves us with three sets of flagged data

6. Viewing and sharing the results

One of the problems that healthcare professionals face is the lack of access to the numerous points of health data that is tracked by users [4]. Although many health care professionals might have access to the pacemaker readings, they wouldn't have access to consumer wearable devices.

After HeartSync gathers and extrapolates the data, the user will be alerted if there are any extraneous readings, meaning a possible pacemaker failure.

HeartSync can then POST the data to a.NET RESTful API to that is made available to store heart beat data in a Microsoft SQL server.

Without the data sync, doctors or healthcare professionals would have to examine the data, looking for any possible data discrepancies that might arise. This would be costly, and HeartSync eliminates these unnecessary steps for these professionals. Without the API, they might not receive the data at all.

7. Potential issues and solutions

One of the issues with HeartSync is that it only checks for exact matches in the heart rate. If there is any difference at all in the heart rate, then the data will be flagged as being possibly problematic. We could create a simple algorithm to provide the program more affordance in determining any issues of the heart rate. For example, we could allow the heart rate monitor reading to be within a few beats of the pacemaker reading. However, is this providing us with a better alert system? And what if we get a new device? HeartSync would need a solution more robust to ensure that its users remain healthy.

More than likely, each device will have it's variance in the accuracy of the heart rate monitor readings. This leaves us a few options moving forward in trying to make sure that HeartSync doesn't produce any false alarms or even worse, not alerting the patient to a possible pacemaker failure.

One of the solutions is, to properly test each of the devices that HeartSync supports. Devices like the FitBit, Apple Watch and Jawbone could be tested in different situations such as a person being active, sleeping, or idle and we could come up with the proper amount of deviation for each device. When a person starts up HeartSync, the application could then detect, or ask for the device being used and then set the algorithm to allow for however many heartbeats it needs to account for the discrepancy. The problem with this solution is that the numbers will constantly have to be updated as new models are released. Also, sometimes the variances might differ depending on the user. For example, a user with hairy arms or excessive weight might get a different reading than others users wearing the same device. It would potentially become a cat and mouse game for the developers.

Another solution would be to provide HeartSync users with some server side functions that would allow HeartSync to make more intelligent decisions about the validity of the heart rate data coming from iOS. A server side application can store all of your heart rate data for you and analyze that data to make sure that your pacemaker is performing correctly. Instead of looking for exact matches heartbeat for a heartbeat, the server side application could look for disturbing trends that could signify a pacemaker problem. For example, what if we had an average variance of 4 heart beats a minute for 6 months and an average variance of 8 beats per minutes a week following that time interval? We could assume that there is an error with either the pacemaker or the monitor and once again alert the user to a possible conflict.

8. Conclusions

In this paper, we propose a method for patients with a pacemaker to test the functionality of their equipment on the go. Patients will receive an early alert to a pacemaker malfunction as well be able to push this data to a remote server where doctors and healthcare professionals can view heart rate data that it is pertinent to them, and make informed decisions about a patient's health. We also talked about the possible problems with the current iteration of HeartSync. We also put forward some possible solutions to this problem like server side machine learning to account for data variances in wearable heart rate monitors.

9. References

[1] How Does a Pacemaker Work? - NHLBI, NIH: 2015.https://www.nhlbi.nih.gov/health/health-topics/ topics/pace/howdoes. Accessed: 2015- 08- 18.

[2] Enes Elvin Gul and Mehmet Kayrak - Common Pacemaker Problems: Lead and Pocket Complications, Modern Pacemakers - Present and Future 2011, Prof. Mithilesh R Das (Ed.), ISBN: 978-953-307-214-2, InTech

[3] Remote-K-Viewer Technology: Managing Cardiac Devices Remotely Via an iPad: 2012. http://www.eplabdigest.com/articles/Remote-K-Viewer -Technology-Managing-Cardiac-Devices-Remotely-iPad. Accessed: 2015-08-21.

[4]Chen, C. et al. 2012. Making Sense of Mobile Health Data: An Open Architecture to Improve Individual- and Population-Level Health. J Med Internet Res. 14, 4 (2012), e112.

