Beehive Monitor

Tyler Lewis - Computer Engineering

June 2014

Project Advisor

John Oliver

Project Clients

Bill Lewis

Frank Czopek

Winter 2014 - Spring 2014

Table of Contents

Introduction

Project Overview

The objective of this project is to develop and build a system that can gather weight data from a managed honeybee colony and then transmit the data to the beekeeper so that the beekeeper can access the data from anywhere with an internet connection. Currently, the only way for a beekeeper to obtain information about their bee colonies is to physically be at the hive location and examine their hives by hand. It is necessary for beekeepers to visit their beehives to harvest honey because colonies that are oversupplied with stored honey tend to swarm, depleting the hive of its current queen bee and roughly half of its worker bee workforce. However, these trips are expensive and time consuming when the apiary is a long distance from the beekeeper's current location. For example, my client beekeeper spends up to $500 in travel and labor costs, in addition to a full day of work time, to make a trip to his beehive locations. This is very detrimental to his business if there is not a significant amount of honey to harvest. With the data the provided by the Beehive Monitor, a beekeeper will be able to remotely determine if they need to harvest honey that day or not, as a colony that has greatly increased in weight over time indicates a hive that is ready for honey harvesting. This will help the beekeeper to prevent unnecessary trips to their apiary and to make more productive use of their time and resources.

Clients and Partners

For this project, I am working with Bill Lewis, a long time professional beekeeper and owner of Bill's Bees, and his personal friend Frank Czopek, an engineer with Boeing Corporation, who originally proposed this project to me. Bill Lewis originally started beekeeping as a teenager to complete a Boy Scout merit badge. After obtaining his masters degree in mechanical engineering, Bill Lewis worked at Northrop, through which he met Frank Czopek. After years working as an engineer, Bill Lewis returned to his passion: beekeeping. Frank Czopek initially proposed this idea to me to help Bill Lewis save time and money travelling to and from the locations at which he keeps his bee colonies.

I made contact with Frank Czopek and received his notes detailing his ideas for the project as of January 4, 2014. Following my contact with Frank Czopek, I contacted Bill Lewis to determine what exactly he would like to see in this sort of system. During the communications, I made plans to keep them both updated on my progress and future project plans on a weekly basis and also to hand off the completed deliverable to Bill Lewis during the week of June 15th, 2014.

Stakeholders

Bill Lewis is a stakeholder for this project as the Beehive Monitor will directly benefit his company's productivity. By extension, all beekeepers who wish to monitor their beehives from a distance are also stakeholders for this project, as its completion would serve their interests as well. Frank Czopek is also a stakeholder for this project, as it has been his interest to construct a device such as this for several years.

Need Statement

Bill Lewis, owner and beekeeper of Bill's Bees, frequently moves his colonies to locations with large numbers of nectar producing plants in bloom, so that his bees can collect the nectar and deliver the largest honey crop possible. When his beehives are a long distance from his home, it would be very useful to him to have some way of getting information regarding the amount of honey stored in his colonies without having to go to the apiary himself. As a member of his family and someone who has worked for Bill's Bees previously, I decided to take on this project to use my computer engineering knowledge to benefit Bill's Bees.

Research Information

http://www.engadget.com/2012/09/04/raspberry-pi-getting-started-guide-how-to/

As I had never used a Raspberry Pi before, I found this guide for properly setting up a Raspberry Pi to be very useful. It provided all the information I needed to know from setting up the SD card for use by the Raspberry Pi to configuring user settings on the Raspberry Pi.

http://www.cplusplus.com/reference/

I used this website as a reference for C language library function calls and to confirm that I was setting the proper parameters and return types for every library function in my C program.

http://man7.org/linux/man-pages/man2/syscalls.2.html

I utilized this website as a reference for UNIX system calls and to ensure that I was setting the proper parameters and return types for every system call in my C program.

http://www.wikihow.com/Make-a-Raspberry-Pi-Web-Server

Having never set up an email server on any system, I found this guide to be very helpful in preparing the Raspberry Pi to send emails. The guide describes what files are necessary to install to prepare a Raspberry Pi to send emails.

http://www.nixtutor.com/linux/send-mail-with-gmail-and-ssmtp/

As I decided to use a Gmail account for my system's email address, this tutorial aided me in setting the necessary parameters on my Raspberry Pi to be able to access Gmail servers.

http://www.php.net/manual/en/funcref.php

Having never written any PHP code before this project, I needed documentation and examples to get started writing the PHP scripts to handle emails. This website provided documentation and examples for using all of the functions that I needed for the PHP scripts.

http://www.raspberrypi.org/forums/

This is the official Raspberry Pi forum board. This website provided all sorts of information about Raspberry Pi, from initial setup, to troubleshooting information, to which programming languages are optimal for implementing a simple email server.

Project Definition

Project Goals

- Construct a system that can weigh a beehive and communicate with the beekeeper.
- Save beekeepers' time and money by reducing the number of trips they must make to their apiaries and eliminating unnecessary trips altogether.

Project Objectives

- Use a scale with a capacity of at least 400lb with deviation lower than ±5lb to weigh a beehive.
- Use a serial connection to communicate between a Raspberry Pi and an external scale.
- Write commands to and read data from the scale.
- Be able to send and receive emails with a Raspberry Pi at least once a day, or faster if required.
- Send emails containing scale data to the user.
- Receive and parse emails containing commands from the user.

Project Deliverables

Upon completion of this project, I will have built a system consisting of a Raspberry Pi and a scale to measure the weight of a beehive. I will also have written the code necessary to obtain data from the scale either on a user specified interval or immediately upon user request, and then relay that information to the user via email. I will hand these items off to my client during the week of June 15th, 2014.

Marketing Requirements

1. The scale should have a maximum capacity greater than the weight of a beehive.
2. The system should be simple to set up and use.
3. The system settings should be customizable by the user, even when the program is running.
4. The system should be reliable to prevent unnecessary trips to restart it.
5. The system should have a fast response time when the beekeeper requests an update.
6. The system should not disturb the environment or the bees nearby while it is running.

Engineering Requirements

Category / Corresponding Marketing Requirement	Engineering Requirement	Justification
Functionality / 1	The scale must have a capacity of 400lb or higher	A bee colony will produce a maximum of 400lb of honey per, but as honey is harvested several times per year, the weight of the beehive will never exceed 400lb.
Usability / 2, 3	The system takes commands formatted liked those used in a Linux command line interface.	This type of interface is fairly simple to learn to use while passing all necessary information to the system in a concise manner.
Usability / 3	The system can change the base measurement time and interval measurement time while running.	Sometimes conditions vary and fewer or more frequent measurements would be more convenient for the beekeeper.
Reliability / 4	The system needs to be able to operate for a week straight without crashing.	Bee hives need to be checked for colony health reasons once a week, so this is the maximum amount of time the beekeeper would spend away from the apiary.
Usability, Reliability / 5	The system responds within 90 seconds to a request for an unscheduled update.	Users do not want to be kept waiting for data.
Environmental, Health and Safety / 6	Components used in the system must be Rosh compliant and cause no damage to bee colonies.	The Beehive Monitor aims to aid the beekeeper in maintaining their bees; harming the bees or the environment directly conflicts with this objective.

System Constraints

- The user must provide an internet connection to the Beehive Monitor for the program to function.
- For the current system model, the user must also provide the Beehive Monitor with a 120V AC power source.
- The system (except for the scale) must be able to fit inside of a beehive box so that it is protected from the elements and can be moved with relative ease.

Testing Criteria

Category	**Description**	**Justification**
Accuracy	This measures the deviation in the output of the scale.	Though it is not important to know exactly how much the beehive weighs, deviating from the actual weight of the beehive by a large percentage would impact a beekeeper's ability to make a good decision based on the data.
Reliability	This measures how long the system functions properly without being restarted.	If the Beehive Monitor program crashes before the beekeeper needs to make a weekly trip to their apiary, it is not achieving its original intention to save them time and money.
Response Time	This measures how quickly the system responds to user emails.	Whether or not an error is detected in emails sent by the user, a fast response from the system makes interacting with the device easier and a more pleasant experience for the user.

Project Planning and Design

Overview

At the start of this project, I met with my advisor, John Oliver, to plan out a timeline for the project tasks. This helped me to determine how much time I should spend on research and planning, code development, and debugging and testing. I also set up weekly meetings with Professor Oliver for both winter quarter and spring quarter so that I could update him on my progress, obtain guidance for my next steps, and get answers for questions I encountered.

Project Schedule

The planning phase of the project lasted from 1/7 until 3/21, during which time I composed the project charter, completed the hardware and software block diagrams, and obtained the necessary hardware components. The code development phase lasted from 3/22 until 4/24, during which time I researched which programming languages and function libraries would be the most efficient in my code and then implemented my solution. Finally, the debugging and testing phase lasted from 4/25 until 5/24, during which time I performed stress tests (by sending multiple emails to the system in a short period of time while it was also taking a measurement from the scale) and long duration testing (running the device under normal operating conditions for a week at a time to check for memory issues). After completing my own testing, I gave my clients access to the system to allow them to evaluate and experiment with the system for themselves.

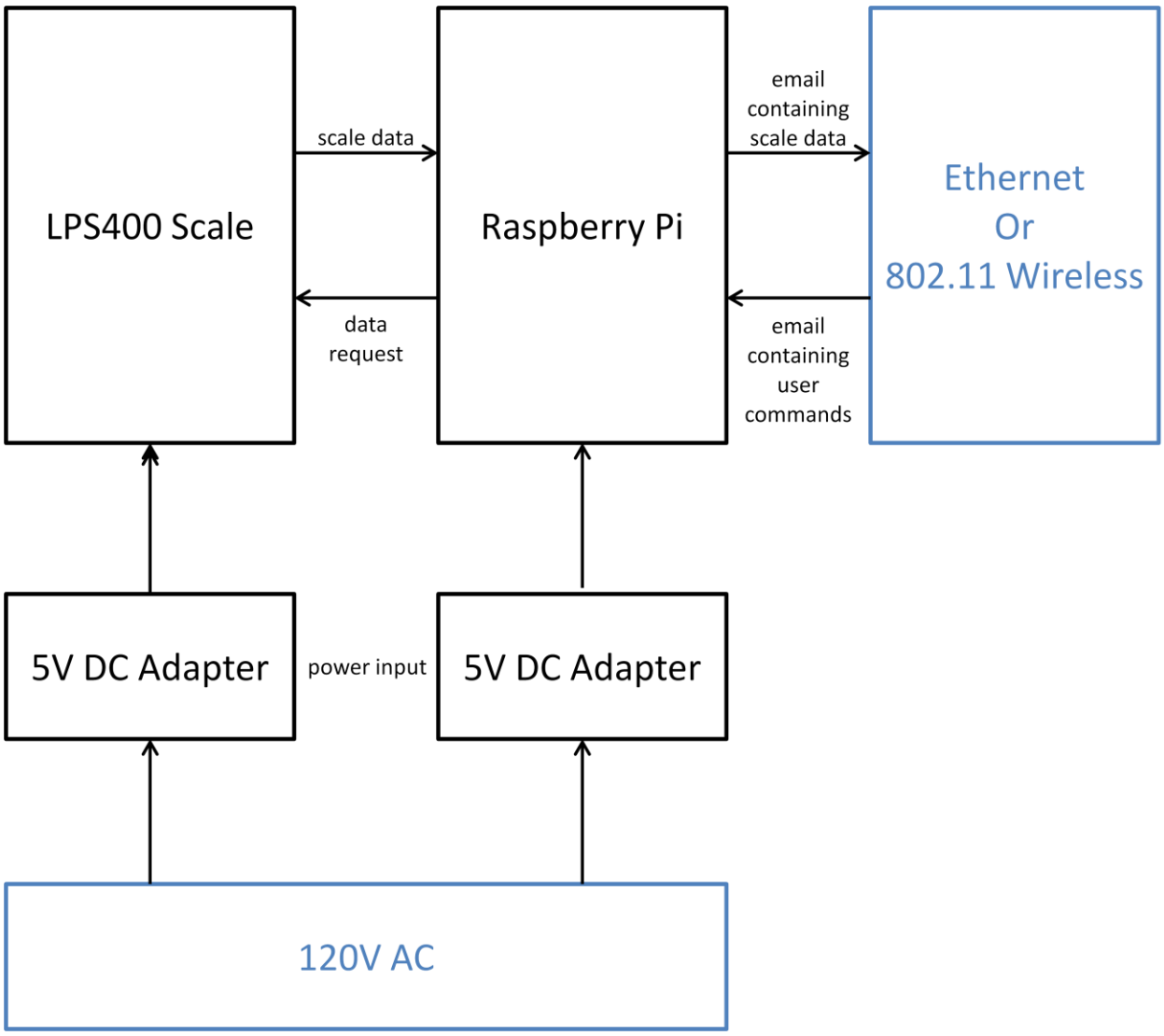
Hardware Components and Justification

There are two main hardware components in the Beehive Monitor system: a computer that runs Beehive Monitor software and a scale that obtains the weight of the beehive. For a visual description of how the physical system is set up, see the Hardware Block Diagram on the next page.

From the start of the project, I planned on using a Raspberry Pi for the system's computer. Its small physical size makes the Raspberry Pi perfect for this project because it can easily fit inside of a beehive box and leave plenty of room for other components. In addition, the Raspberry Pi has a 700MHz processor, 512MB of RAM, and 8GB of memory space, which is more than enough capacity for the Beehive Monitor. From the point of view of a computer, the Beehive Monitor only creates, deletes, opens, closes, reads, and writes to files, none of which are very demanding tasks. The Raspberry Pi is also compatible with both Ethernet and wireless internet connection, allowing the user to choose whichever one best suits their needs. Finally, the Raspberry Pi is very well documented and has an active FAQ board and forum, which was quite useful to a first time user such as myself.

For the scale component of the system, I originally planned on using load cells to build my own scale, because of their accuracy and weight capacity. However, upon determining their cost, I decided they would be far too expensive for this project. When I relayed this information to my client, he agreed and a Brecknell LPS400 postal scale was procured for the project instead. I actually had no part in making the decision to use this scale, but as the purchase was already made, I accepted the change in my project specifications and verified that the LPS400 would meet the 400lb capacity requirement, and also that it would be possible for me to interface it with the Raspberry Pi.

Hardware Block Diagram



Setting up the System

1) Start the system and all of its components. Place all of the components except for the scale in a protective box. Place the box containing the system components in location desired for the beehive. Center the scale on top of the box.

2) Center a bottom-board on top of the scale.

3) Place a beehive on top of the bottom-board.

4) This is the completed setup. Shield the sides of the scale if deemed necessary.

Software Design and Justification

For the software portion of the Beehive Monitor, I decided on a highly modular approach, breaking the software into three separate programs. This allowed for much easier debugging and testing, as a component can be tested on its own before being added to the main program. In addition, smaller pieces of code make it much faster and easier to track down errors; for the same reason, it will be much more simple to modify the code in the future to improve or add to the system. For a visual description of how the software is set up, see the Software Model in Appendix A.

Initially, I planned on writing the entire project in the C programming language. C is a great language for interfacing with hardware, as it is very memory efficient and runs much more quickly in comparison to interpreted languages (such as Java) or scripting languages (such as PHP). In addition, C is a personal comfort choice of mine because I have used it for the past five years. Because writing my own program for sending and receiving emails would be very unpleasant in C, I planned on using a prewritten email program such as 'mail' or 'mailx' to handle this portion of the project. However, I experienced many difficulties in getting these programs to behave exactly as I desired when sending and receiving emails, so I abandoned my attempts to utilize them and decided to write my own code instead. This meant that I would need to choose a different programming language to write this part of the code, as I did not want to write it in C. After searching the Raspberry Pi forums to determine what programming languages would be best for implementing a simple email server, I came up with two candidates: Python and PHP. After doing some research into both languages, I chose to use PHP because its functions were all very well documented and explained with examples. In addition, PHP syntax is similar to that of my comfort programming language, C.

System Test Plan

1. While writing the code, perform sanity checks and functionality testing on each module before adding it to the system.

2. Upon adding a new code module to the system, perform regression tests to verify that all previous code still functions as expected.
3. Change the weight on the scale between measurements to verify that the scale does record different weights.
4. Change the weight on the scale while it is taking a measurement to verify that the recorded data is accurate.
5. Send email requests with incorrect parameters and formatting to verify that the system will not crash.
6. Perform a short duration stress test during which many email requests are sent to the system over a short period of time while the scale is also taking a measurement to simulate the maximum burst amount of work the system will have to perform.
7. Perform a long duration test under normal operation conditions to verify that the system code has no memory leaks. During this test, do not vary the weight on the scale so that it is possible to notice deviations in the scale output.

System Testing Results

1. When writing new code modules, I added debugging statements to give me information regarding how the code was running. Each of the three modules spent a large amount of time in this step due to syntax errors and stability issues. 'beehive.c' spent the largest amount of time in this step because of the large number of system calls that it uses; the output of each system call must be checked before proceeding in the program's execution, otherwise it could crash and would have to be manually restarted by the user. Though I checked both 'rcv_email.php' and 'send_email.php' for the same issues, they are not as important as the stability of 'beehive.c', as 'beehive.c' will restart these programs as needed. Once I was confident that a particular code module was causing no errors, I removed these statements and added the module to the main system program.
2. When I added a new code module to the main system program, I would run the entire program for several hours while periodically printing out debugging information to make sure the addition of the new module did not cause any bugs. Once I was confident in the

functionality of the software, I disconnected the monitor from the Raspberry Pi and proceeded to testing the functionality of the system.

3. When changing the weight on the scale between measurements, the next email did reflect the change in weight.
4. When changing the weight on the scale during a measurement, the scale would wait for the reading to stabilize before sending that data to the Raspberry Pi, resulting in the correct weight being displayed in the email.
5. I sent emails containing misspelled and unknown commands for this current model of the Beehive Monitor, which were ignored. I also sent commands with incorrect numbers of parameters. If there were too few parameters or incorrectly formatted parameters, an error message was sent back to me by email. If there were too many parameters, the system took accepted the required number and ignored the remainder.
6. When performing the stress test, I sent many emails all within sixty seconds (the time that 'rcv_email.php' sleeps after checking the inbox) for each command, including some with unknown commands and/or incorrect parameters, to check all possible forms of user error and to see if the system could handle conditions much worse than the expected normal operating conditions. When this test succeeded, I tried it again while 'beehive.c' was taking a measurement from the scale to simulate the worst case scenario. This test also succeeded.
7. Finally, I tested the system under normal operating conditions for an entire week to simulate how a beekeeper might actually use the device. The purpose of this test was to make sure the 'beehive.c' was not leaking memory, which would eventually cause the system to stop functioning and require a manual restart. In addition, I used this time to note the deviation in the scale's output, which ended up being ±2lb at the worst. A beehive typically weighs well over 100lb, so this 2lb fluctuation will have no effect on a beekeeper's ability to make a sound decision based on the data received from the Beehive Monitor.
8. Although not part of my original test plan, after completing my tests, I gave my clients access to the system, along with a User's Guide (see Appendix C). My intention in doing this was that they might find an error that I had overlooked or suggest some potential changes to make the Beehive Monitor easier to use. They did not have any criticisms about how the system functioned or the email updates they received, so at this point I concluded the testing phase.

Closing Statements

Sustainability Impact

The only maintenance that the Beehive Monitor requires is the removal of dust and other contaminants. As most of the components of the Beehive Monitor are housed inside of a protective box, they would only need to be checked for dust, which could be removed with pressurized air. As the weight sensor of the scale is located directly under the beehive and is more exposed than the other components, it is recommended that the sensor be wiped off with a damp towel periodically. This system maintenance could be performed by the beekeeper when they visit their apiary.

The Beehive Monitor indirectly affects the sustainability of fossil fuel resources by saving the beekeeper trips to a distant apiary. The weight data that the system sends to the beekeeper lets the beekeeper know if it is necessary to harvest honey that day. Any trip that they do not have to make gives the beekeeper more time to accomplish something else, in addition to consuming no fuel and producing no carbon emissions.

Some future upgrades to the Beehive Monitor include additional sensors such as a temperature sensor internal to the hive (to help gauge colony health) or a bee counter at the entrance of the hive (to track colony activity). Some upgrades to the user interface include additional commands that could be sent to the system by email (to improve the user experience with the device).

Some challenges that will occur when upgrading the device include the lack of additional USB ports; additional devices must be compatible with the general purpose input/output pins on the Raspberry Pi. Each additional data sensor also requires additional code in the main program to read data from that sensor. Adding new commands to the system requires that the program that monitors the Beehive Monitor's email address be modified.

Health and Safety Impact

The Beehive Monitor does include small parts that could pose a choking hazard for small children, as well as larger and heavier parts that could cause bodily harm if dropped on fragile body parts.

Social and Political Impact

Using the Beehive Monitor with an email address that is not your own, particularly with a low measurement interval, will cause that email address to receive a large number of emails from the Beehive Monitor. Actions such as these that cause unwanted spam are subject to the email spam legislation in the country of use.

Conclusion

When beginning this project, I had very detailed knowledge of most of the C programming language, which I used to write the Beehive Monitor's main executable named "beehive". However, though I was aware of its existence, I did not have much prior experience using the <termios.h> library, which allows a program to interface with external devices (in this case the scale). This project helped me to understand more about the use of this library.

Prior to this project, I had no knowledge of the PHP language. I am now comfortable with my ability to research and use PHP in inter-process communication, as well as to send and receive requests via email.

Finally and most importantly, I feel like I gave gained experience that will be useful in a future career by working on a project from the planning phase, through the development and testing phases, and finally to the deployment phase. In addition, I have developed and produced a completed product that will be immediately applicable and beneficial to my client.

Appendix A - Software Model

```
beehive.c
// check user parameters
// fork and exec rcv_email.php
// open and configure connection to scale
// sleep until base measurement time
while(1) {
    if(rcv_email.php has terminated) {
        // fork and exec rcv_email.php
    }

    // request data from scale
    // read data
    // write data to out_email.txt

    // check in_email.txt for parameter changes
    // change parameters
    // delete in_email.txt

    // fork and exec send_email.php

    // sleep for interval time or until next SIGINT occurs
}


rcv_email.php
while(1) {
    // open connection to email inbox

    for(all emails) {
        if(email subject is recognized user command) {
            // write email body to in_email.txt
        }
    }

    // delete all emails
    // sleep for 60 seconds
}


send_email.php
// read data from out_email.txt
// send email containing data
// delete out_email.txt
```

Appendix B - User's Guide

Starting the System:

1. Navigate to the directory "Desktop/beehive_monitor".
2. Enter any of the following commands:
   ./beehive
   ./beehive [0 - 23] [0 - 59] [0 - 24] [0 - 59]
   ./beehive [0 - 23] [0 - 59] [0 - 24] [0 - 59] [location]

If entering your own parameters, the first four parameters must be integer values in the ranges specified above. The first and second parameters are the hour and minute values of the initial measurement time. The third and fourth parameters are the hour and minute values of the interval of time between measurements. The final parameter is an optional string of characters consisting of one word to help identify the location of the device (if a person happens to have more than one Beehive Monitor).

Communicating with the System:

email address of the device: **********@gmail.com [I gave the clients the actual email address of the Beehive Monitor at that time, but I blocked it here for privacy reasons].

The subject field of the message should be one of the commands (listed below without the double quotes). Some commands require additional data to be present in the body field of the email. Commands that are spelled incorrectly currently produce absolutely no response.

commands:

- "measurenow" – This command will cause the device to send an email update on demand. The delay on the return message can be up to 90 seconds. No message body is necessary for this command.

- "changetime" – This command will cause the base time and interval time for data measurement to change AFTER the next data measurement. It requires 4 parameters separated by any number of spaces in the email body. The minimum interval for data measurement is 0 hours, 5 minutes. The maximum interval for data measurement is 24 hours, 0 minutes. All parameters must begin with a numerical character, but may be followed by any number of non-numerical characters, which will be ignored. There are some examples listed below. Any invalid parameters will cause an email alert.

  1. Parameter 1 should be an integer from 0 to 23. This represents the hour for the base time of the first data measurement.
  2. Parameter 2 should be an integer from 0 to 59. This represents the minute for the base time of the first data measurement.

3. Parameter 3 should be an integer from 0 to 24. This represents the hours for the interval between data measurements.
4. Parameter 4 should be an integer from 0 to 59. This represents the minutes for the interval between data measurements.

"0 0 24 0" would take a measurement every 24 hours starting at midnight.

"f0 0 24 0" is invalid, but "0fhg 0 24a0 0fff" is exactly the same as "0 24 0 0".

Example Beehive Monitor Email Message:
location - Wed May 21 01:40:00 2014
Weight: 16.2

Next Data Measurement: Thu May 22 01:40:00 2014
If you do not receive an email at that time, restart the device.