IPL RFFD
Capstone Project

By Joseph Vaughn

Abstract

Power lines and equipment cover every inch of modern society and are the building blocks of our modern life styles. However due to the size and sheer amount of equipment it is hard to detect when a piece of equipment is about to go bad. Having a sensor on every piece of equipment and on every mile of power line would be both expensive and difficult to monitor and coordinate. So a simpler solution is needed. When a piece of equipment fails it is quite easy to detect since when it fails chances are someone is without power now. So either by the few sensors that we have or by sour phone calls we will be informed that a piece of equipment is no longer functioning. But what if we could test for equipment before it fails. When a piece of equipment is failing it will make an electrical spark. These sparks aren't always large arches of lighting that you see in YouTube videos. Usually these sparks are just small short circuits inside the piece of equipment. The key for this project are two things, first that these sparks are consistent and unique, and second that these sparks cause radio frequencies.

Introduction

My project is based on the device that I have been given to work on, the RFFD which stands for Radio Frequency Fault Device. Essentially, this device uses radio waves to detect spiking electrical currents from equipment around the city, primarily targeting towards power lines. Whenever the device would detect a fault, it would record a dataset and send the data to the local server which would then be uploaded to a heatmap to display the information. I was sponsored by IPL to not only improve the accuracy of the device, but to also create a web server that can display the data they wish to see while the device is obtaining data from the average IPL truck drivers.

Background

The theory has tested to be true and a simplistic Raspberry pi detection device has been developed. The current system was developed by an IUPUI electrical engineering senior design team and has the ability to detect the radio frequency faults and add the GPS location of the fault to a heat map.

While the system is functional the software is a bit lacking. The previous projects were more focused on the hardware and theory then the software.

Luckily I knew a fellow colleague of mine that worked at IPL and was able to sponsor me for this project since it has been in the works for a while. Mostly IPL only affiliates with the Electrical-Computer Engineering department on campus, so my colleague wanted to see what would happen if a Computer Science student attempted to do research. As a Computer Science major, I have had a lot of experience in front end (client-side) programming, software engineering and some form of experience with environments that is in a C/C++ environment.

The first round of objectives at first was a bit steep at first since I was the only student working on this project. I had to make some compromises with both my IPL sponsor and my academic advisor to set a goal in mind for both sides. My primary objectives were pretty straight forward: improve the RFFD, create a stable web server for the company that can be passed off in future iterations and lastly provide as much feedback as possible from what I've observed and researched in the process for their team can improve upon.

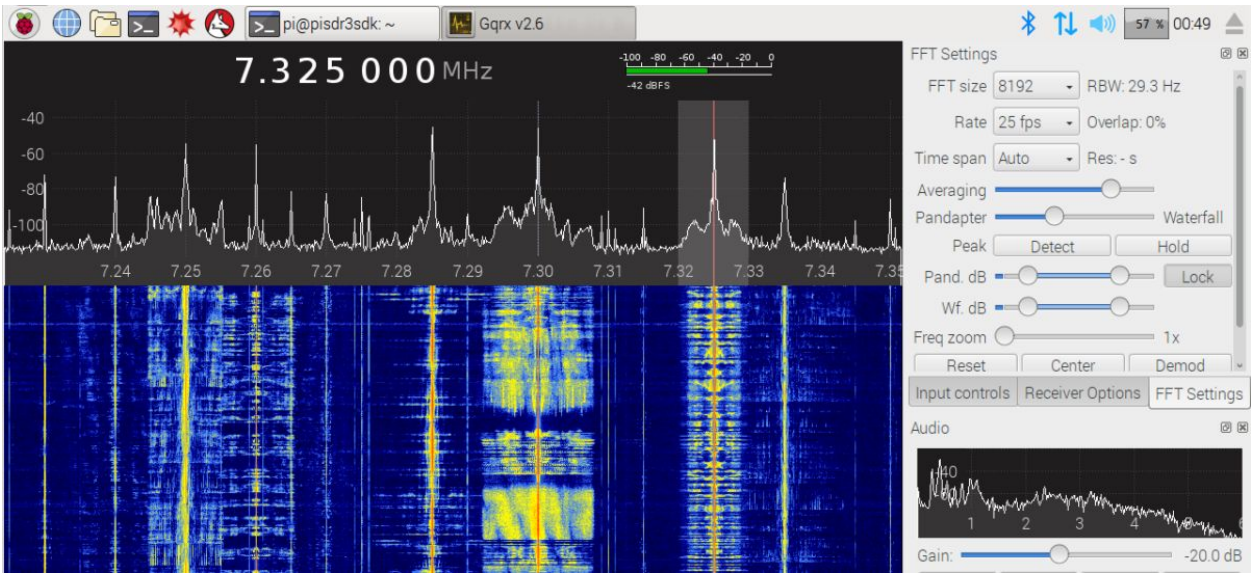
This is a growing technology within IPL, so they want to continually test and research this topic so they can see if it's worth investing in down the line. A device that's capable of tracking faulty equipment can obviously be a very powerful tool for the city. As you could imagine, this is a very significant item to research since it can prevent many power outages in the future and reduce the frequency at which they happen since they can detect a fault before it happens in real time, let alone they have data from prior experiences.

There are many devices that IPL is currently researching or currently using today. I do not have the details of such products, but I can verify that using radio waves to communicate with their devices has always been used since the company first started. At this point, they're investing time in it to see if it's worthy investing.

Method, Implementation and Experiments

Over the course of the semester I have gone through many phases of research. The first portion of the project was to improve the RFFD within the software itself. I spent a solid amount of time figuring out how to operate the program(s) since it wasn't conventional to use. The only way I could actively work on the device is if I had an HDMI/VGA cord to plug into a display. Remote connections were very restricted and only in command line since it was Linux Debian 9 based. For a while, my sponsor and I tried to use a Raspberry Pi instead but the difference in operating systems and setting up the software from the tinker board was a lot difficult, so we stuck with the ASUS tinker board instead. At this time I had to familiarize

myself with the old documentation from the past group such as reading their journal entries and instructions on how to run the device. I spent many nights testing different equipment that was given to me, such as the difference between antenna and how different the data retrieval would look like if I used a modifier such as an RTL-SDR. In the meantime, I dived into how to properly read and observe radio waves through a program called GQRX:



Which is used to listen, observe, and modify the perception of radio waves as a whole. This would come into play later with the use of the software that was used to detect the faults on the spectrum.

Once I finally figured out how to properly run it, I started to implementing different ways to change it. One of the requirements of the project was to make it be able to function online and offline. Before, if it was not connected to wi-fi, it would go into a fail-safe mode and connect until it had a connection. Carefully, I modified how the code runs which was a giant while loop; the way I modified it was detect whether or not it has a connection at first. If it does, immediately upload data from the text file and send it the server program (receives data, writes data to json file, heatmap uses json file for points on the heatmap) on the local machine. If not, store it in the local text file and keep recording data. It was difficult to modify since I was dealing with someone else's code that had little to no comments with directions on what certain functions did so I had to teach it myself. Lastly, I also modified when the device should record data and that is constantly checking to see if the vehicle is going more than 5mph which is at running speed. It's not very useful if the vehicle stops for 45 minutes in a rural neighborhood or else you'll have a giant red dot on the heatmap of unnecessary data that is obviously false. This change was pretty simple to fix; before the start of every ping, I would use a function called getSpeed which used a calculation based on the current and last position based on the GPS coordinates and estimate the speed

at which the device was going. If the ping was more than 5 and it hit a fault, record the data. If not, continue on because most likely it got the first ping before stopping. At this point, I gave the device to my sponsor who mounted it to the IPL trucks for I can get some data to play with since the old data was highly inaccurate.

The next part that I wanted to focus on was improving the frontend and backend simultaneously. The way the heatmap.html worked was by making it locally hosted live by using the Node.js libraries. This was the only functions being used for Node.js, but essentially this is how I tested the heatmap while feeding it raw data in my own room to see it on the heatmap. Also in order to use all forms of the heatmap, I had to receive a google API key in order to use all functions in the map. This is crucial because I needed it to create the search bar. Luckily google has plenty of guides that tell you how to make a search box using their API so that wasn't too difficult. Now that I understood how the program worked overall, it was time to start improving it. One of the first things I wanted to do was to build a database for the data since they only use data points from a json file. They would like to be able to sift through past months at any given time, so a json file is not acceptable. This was when I started to research MySql; it has been a while since I've taken Database Systems taught by John Gersting so I had to freshen my skills in SQL. My first few issues were simply using MySQL workbench, which turned more into watching online classes learning how to use it properly such as creating UML diagrams, managing your servers, and creating your databases either through SQL entries or the "drag and drop" feature they have. After familiarizing with MySQL, I wanted to connect it locally through a web browser. This is when I realized that the project was going to take a turn so I ventured into the abyss of Server-Side programming.

Since I had to work with a database, I had to research programming languages in order to manipulate the data from within them which lead me to PHP, which was a language I had no experience with. When I started to practice with PHP, I realized it's not as easy to test such as Javascript and HTML where the web browser is essentially your "compiler" and ide. I had no way to test my PHP scripts so I uploaded them to Tesla, which does support php. These were simple scripts to test if they actually did function the way I wanted them to to get my feet wet. Not knowing the Node.js is actually a very good server side programming language, I still diverged more into php since my advisor knew a lot about the language and a website dedicated to his research.On my Node.js locally hosted server, it did not support my php applications (right now, I see how ironic it was) so I looked elsewhere to host my server since I really wanted to use php. I also discovered that Tesla technically does not support database work since it would have to request permission from the professor I would be working with every time I wanted to commit anything. I came across a peer that had taken server side programming classes and recommended me to use Apache, which is when I then discovered XAMPP which can easily connect your local web

server to your local database through 127.0.0.1. Finally, I did end up getting the database to connect to a very basic index that showed it successfully connected to the database and was also capable of doing sql queries with use of scripts.

Shortly after using Apache to test my methods, Spring Break had arrived and the next step of my project was coming close. I spoke with my sponsor and advised me to get it live. Him and I spoke to great lengths of what the provider should be and he was leaning more towards AWS since they are vastly known for being reliable and secure as a web host. Over spring break I looked into both AWS and Google Cloud to get an idea of what the costs would be to run a web server. Luckily they do offer free tiers but they will start to bill you if you do go over their specified limits. Ideally I wanted a database and a php supported website, which would have been expensive since allocated memory and security protocol to databases is not cheap. While researching AWS, they had many great tutorials and videos that demonstrated how to use php effectively in your websites which I found very helpful and used it to practice over the break.

After break, I had a few interviews lined up with some Indianapolis companies; one company wanted to know what my approach was for running a live website on my senior design project and offered me advice. He highly recommended a site called siteground which was only five dollars a month, supports php applications, has their own built-in MySQL support, Wordpress, and many more options for only that much. They even had a file manager system very similar to Tesla! This website also had many online tutorials if you did become a member, which I eventually did. (Currently using it as my online portfolio as I type this.) I had to learn the in's and out's of siteground, such as how to not use a WordPress generated site and do something simple for a project like mine. It honestly took about two weeks for me to realize it HAD a public_html section.

It was time to migrate my test files that I have been tinkering with in Apache over to siteground was I found out about the public_html side of the server. Once I successfully moved everything over, I started to create a very basic template for the front end of the website to give myself an idea of why to strive for. I created a Home, Heatmap, Severity and Admin tab. The home simply cited what the project was about, the Heatmap would display information that was gathered from the RFFD, the Severity was used to display the PHP scripts that I've been testing, and lastly the admin panel would be implemented to make it so the admin team can create, read, update and delete desired databases while also uploading images.

While making these tabs I still was extensively learning proper php coding and ended up having to settle with a simple php code that displays all of the rows in the database from the backend. I did want to go more in detail of an advanced filter that lets the team view the lists such as most relevant, highest,

lowest, and especially by time with different months or weeks. It was also a challenge to properly make the heatmap extract data from the database so I still used the json file for my presentation; the php script kept breaking and would have difficulty getting the right data. I also was pressed for time so I was unable to create a fully fleshed out admin page.

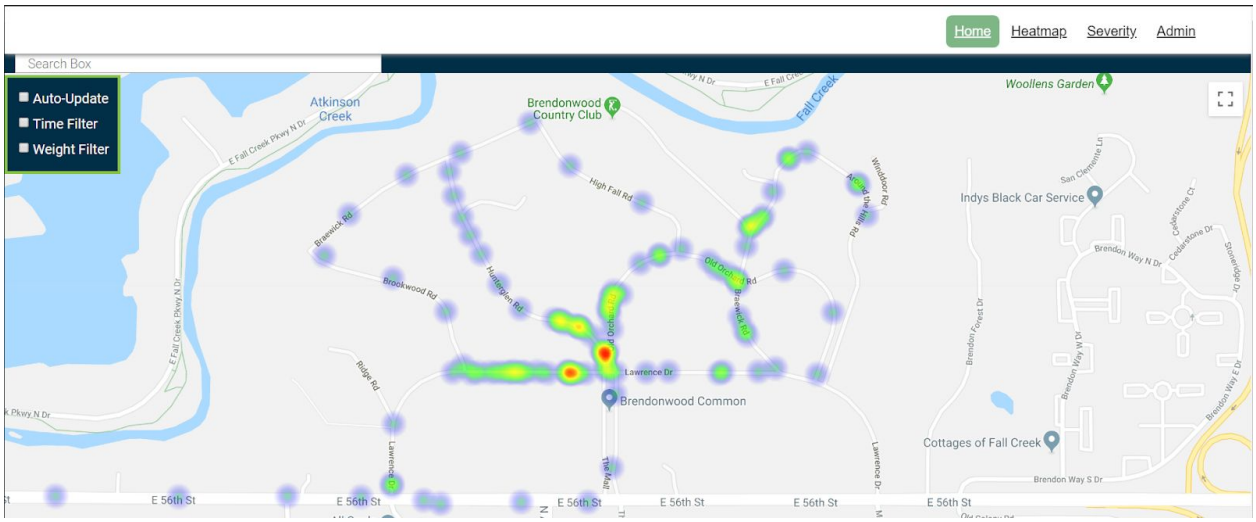
Concluding Thoughts

Overall, the project was a success to me personally because at the start from day one, I wanted to challenge myself on creating and running website on my own, unlike in the past it was for a class. I knew there were certain things I may not know but many familiar concepts to help me through the way. If I had to say the percentage of how "complete" my project is, would be about 70% mostly because there were a few things that I really wanted to fix in the backend, such as the heatmap obtaining data from the database, a complete admin page, and a less buggy heatmap. For the actual device itself I also wanted to investigate more into a backup data plan for the tinker board in case the data had malfunctioned. The most challenging parts of this project was essentially managing each portion of it which was one of the reasons why I felt pressed for time. I had the frontend, backend and the device itself to manage, let alone I had no server side experience before this so personally if this was a two person or third person project instead of just me, I honestly believe this would have turned out phenmonimally.

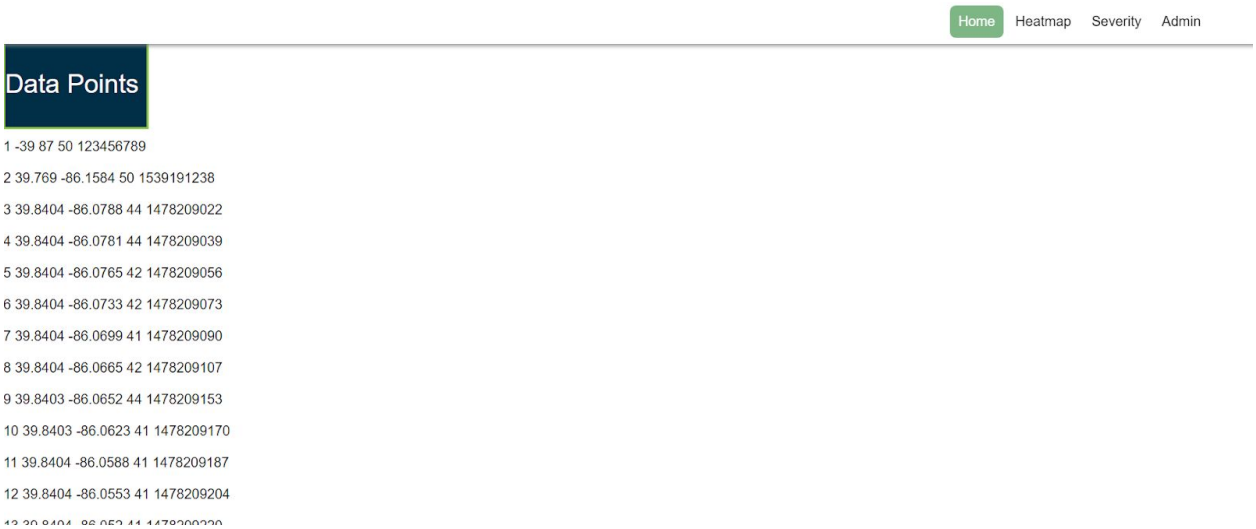
In the end, I did end up creating a good website for them to use, but I do plan on continuing my work with them in the summer since I did learn and strive to improve and learn more from this project since it's a real life example of how to design in a full stack manner.

www.ipl-rffd.com: is the website:

Heatmap in action from an IPL truck working in a residential neigborhood. Red indicates a high weight signal which means it's higher than the set threshold, which is 41 mHz in this demo.



Severity page: Although not completely what I wanted, I did show the internal database does look like and demonstrated that I can connect and manipulate data in php.



Data points being received (ignore the 0.00000, it bugged out while on campus. More in the video demo)

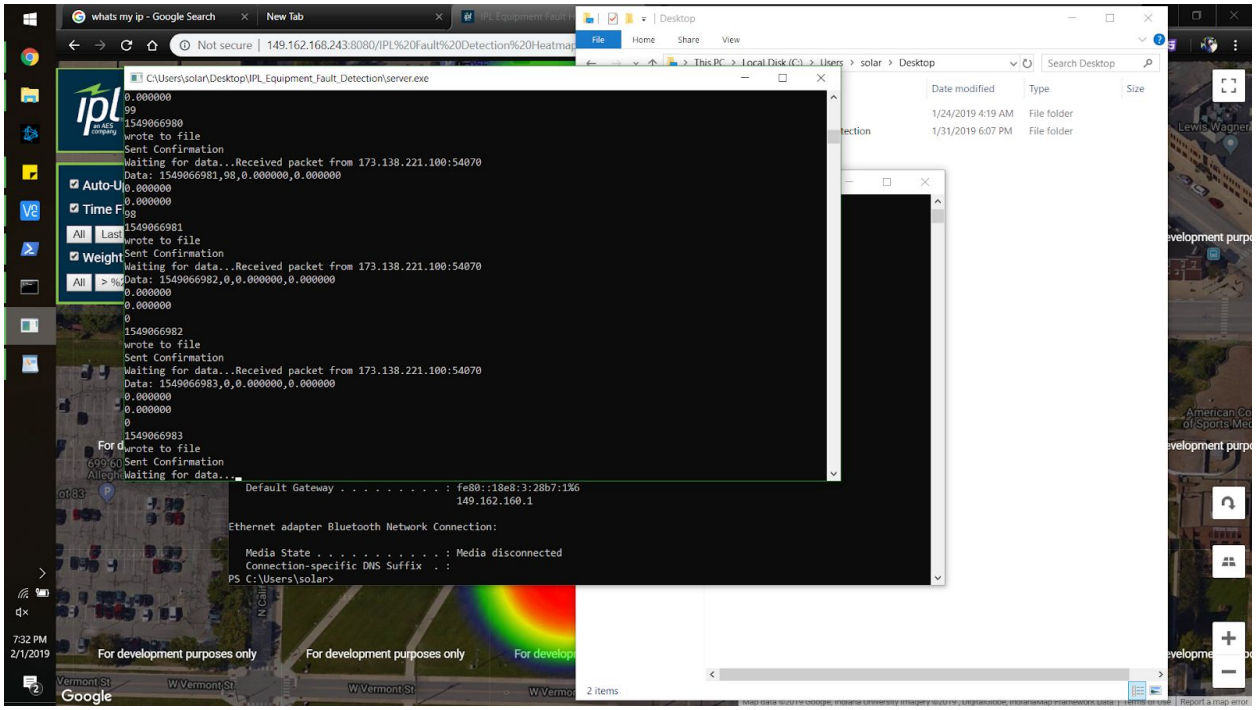


Image of the Siteground Public HTML page, this is where I stored the front and backend

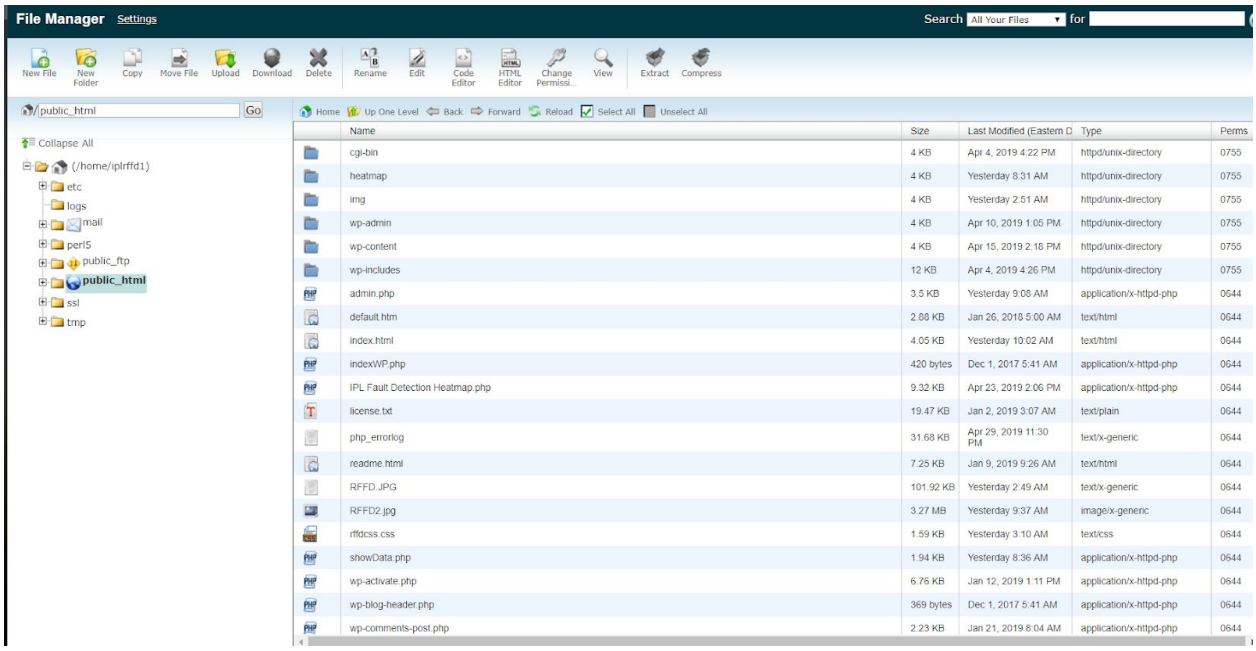


Image of the database within Siteground's MySQL server (using phpMyAdmin)