Demystifying Software Defined Radios using an approachable Docker Image

Kade Cooper, Jing Guo, Pieter Bork, Carlos Lawrence, and Jinpeng Miao
Interdisciplinary Telecom Program
University of Colorado Boulder
Boulder, CO, U.S.
{Kade.Cooper,Jing.Guo,Pieter.Bork,Carlos.Lawrence,Jinpeng.Miao} @colorado.edu

Abstract—The use of SDRs is non-trivial and the learning curve along with the time overhead required to retrieve useful data from an SDR is a deterrent to users regardless of their skill level. Basic FM radio applications require the user to boot to a live image, configure drivers/libraries, or install specialized packages for a variety of platforms. Most current packages do not offer a comprehensive solution and most ability of everyday users to access the RF spectrum. Although one device that reduces overhead costs and increases the ability of everyday users, developers, or security professionals access to a single platform which drives, so data is persistent between scans. SUPERFREQ has setup and maintain its own database saved to a user’s hard drive, so data is persistent between scans. SUPERFREQ has addressed these issues by allowing everyday users, developers, or security professionals access to a single platform which produces actionable information using a SDR.

The rest of the paper is organized as follows: Section II provides a history of the SDR and background information for our study; Section III elaborates on the design and test of SUPERFREQ; Section IV discusses the results of our current design; Section V concludes the paper and discusses potential future work.

I. INTRODUCTION

A Software Defined Radio (SDR) is a dynamic piece of hardware that uses a general purpose CPU and software to interpret and decode RF signals on a range of frequencies. This means the single SDR is capable of capturing and interpreting medium and high-frequency signals, also referred to as intermediate frequencies (IF). Its core idea is to use a broadband “digital / analog” converter close to the antenna to digitize the signal as early as possible, implementing processing functionalities in software whenever possible. In short, SDRs are a relatively new wireless communication architecture based on digital signal processing (DSP). For an SDR system to be useful as an adaptable future-proof solution, and to cover both existing and emerging standards they must be reconfigurable, intelligent, and programmable. The SDRs replaces expensive hardware and multiple antennas with a single device that reduces overhead costs and increases the ability of everyday users to access the RF spectrum.

Despite the advantages of SDRs, current software in SDR space is underdeveloped and is not simple to deploy or use. With this in mind, we aim to develop a user-friendly package, called SUPERFREQ, composed mainly of two parts. Firstly, we use HackRF, a device used to intercept wide range of frequencies, and create an easily deployable package that allows for collection and display of WiFi information on 2.4GHz and 5GHz frequency bands along with collection of Bluetooth and Zigbee information on the 2.4GHz frequency band. From this work, a user would connect to our Docker image via their preferred web browser to view graphs of all packets and their respective information in a easy to understand format. This additional technology was aimed to help end users gain deep insight into various radio defined networks. Overall, we hope our Docker Image aptly named SUPERFREQ, has helped layed down the basic foundation for people curious in this technology to expand it to LTE and beyond.

Index Terms—Docker Image, GNU Radio, HackRF, LTE, SDR
who has continued progress to this day [4]. In 2001 (when the SDR architecture was first patented [5]), Eric Blossom created the GNU Radio Companion to help make common SDRs more accessible to dedicated programmers which was followed in 2004 by the first commercially approved SDR [6], the Anywave, used by cell phone carriers. Mitola continues to drive the field forward [7] and in the past few years, many more SDRs have become available allowing anyone with an interest to acquire the necessary hardware.

The underlying interest and advantages SDR has brought lies primarily in three key areas: greater interoperability, reduced obsolescence, and lower barrier cost. [8] Since SDRs provide greater interoperability, they allow for multiple devices using different channels and bands of frequencies to be accessed via a single point allowing for sectors such as public safety better means to communicate through disaster responses. [9] SDRs reduce obsolescence of the hardware itself since they are meant to be expandable through means of continual software additions. On this note, SDRs provide economies of scale in terms of cost, size, and weight while requiring low energy use to operate through various computer interfaces. [10] In addition, a common interface to SDR technology has been through USB connection to Linux operating systems because of open source libraries such as GNU-Radio and ease of access. [11]

Ergo, SDR technology has moved towards ease of use and this project continues to push that boundary. The first big step in the field towards ease of access for SDRs was the release of GNURadio [12] which moved developers from pure hardware programming to a simple user interface. In addition to project development software, many people have developed individual projects like BLE Dump [13] (packet capture using a SDR for Bluetooth), BTLE Decoder [14] (packet capture using RTL-SDR for NRF24 and Bluetooth Low Energy), and the RFSec Tool Kit [15] (a collection of Radio Frequency tools). Furthermore, progress is being made to simplify and collect the underlying protocols in a reusable library for SDR development [16]. While these other projects contribute to the utility of SDRs, they are still hosted on version control servers and require the user to configure the underlying systems to enable their use.

To achieve the above aims, we need to discuss some background topics. In this section, we will lay out information regarding hardware, software, and database implementations. Exploiting this knowledge, we can continue to carry out our work.

A. GNU Radio

GNU Radio is a representative project for software radios in the open source world. Its emergence has enabled the open source world to break the monopoly of traditional communications giants and allows people to freely understand any details of the entire communications system. GNU Radio implements most of the modules required for software radios, completes the buffering and scheduling of sampled data streams, and is collectively maintained by the open source community. GNU Radio software is very comprehensive for RF front-end hardware support, such as USRP, HackRF, BladeRF etc. GNU Radio performs all the signal processing. You can use it to write applications to receive data out of digital streams or to push data into digital streams [17], which is then transmitted using hardware. Since GNU Radio is software, it can only handle digital data. Usually, complex baseband samples are the input data type for receivers and the output data type for transmitters. Analog hardware is then used to shift the signal to the desired center frequency. A developer is able to implement real-time and high-throughput radio systems in a simple-to-use and rapid-application-development environment.

B. HackRF

HackRF is a fully open source hardware platform. The schematics, PCB design, operating system drivers, and microcontroller firmware are all released under the GPL agreement, which is the main reason we chose this radio for our project. The technical specifications and abilities of the HackRF are as follows:

- **Wide Operating Frequency Range.** HackRF operates from 1 MHz to 6 GHz, a wider range than many SDR peripheral available today. This range includes the frequencies used by most of the digital radio systems on Earth. [18]
- **Transceiver.** HackRF can be used to transmit or receive radio signals, however it is only a half-duplex link, so it can transmit or receive, but cannot do both at the same time. Full duplex operation can be achieved by using two HackRFs.
- **Portable.** The HackRF is small, lightweight, and is powered via USB allowing for easy transportation.
- **Wideband.** The maximum bandwidth of HackRF is 20 MHz, about 10 times the bandwidth of TV tuner dongles popular for SDR [19]. This means that HackRF could be used for high speed digital radio applications such as LTE or 802.11g.
- **Open Source.** The most important goal of the HackRF project is to produce an open source design for a widely useful SDR. All hardware designs and software source code are available under an open source license.

![HackRF One.](image)
C. SQLite

When deciding which database to use in SUPERFREQ, the most important factors are that the solution is lightweight and portable. We chose SQLite to implement a SQL database over a NoSQL database because our data is structured and has a well-defined schema. For SQL databases, PostgreSQL and MySQL are the most widely supported and commonly used, but they also have significant installation and deployment overhead. To overcome these issues, we used SQLite, a relational database management system that uses a single file and a built-in python library to interact with tables and data in a simple manner. This solution allows us to interact with a SQL database with simple commands from our application and has proved to be as user-friendly and efficient as advertised.

![SQLite](image)

D. Protocol Capture

An SDR can receive and interpret a large variety of protocols, but prioritizing specific protocols was necessary in this project due to time constraints. As our team is new to SDR development, we decided to start with the protocols that were the most well documented and had a significant amount of support: IEEE 802.11 (WiFi) and IEEE 802.15.4 (Zigbee). Previous development work was also done on IEEE 802.15.1 (Bluetooth), and was our third choice for integration. We weighed and selected each protocol based on multiple factors shown in Fig. 2.

E. App Packaging

At the inception of SUPERFREQ, we had seen other apps successfully use the AppImage packaging structure to create all-inclusive tools that were easily deployable and multi-platform. As we began development, we soon found out that while this solution may work for smaller applications and simple interfaces, it was not suitable for an app with large dependencies like GNU Radio as well as running a web server. After seeking alternative solutions, we quickly discovered Docker as an alternative that allowed us to pre-build all of our necessary packages and remove all dependencies on specific operating systems.

F. Docker

Docker is an open source application container engine that enables developers to package their applications and dependencies into a portable container and then publish to any popular Linux machine for virtualization [20]. Containers provide privilege and process separation while utilizing the host machines hardware, allowing them to read data from the HackRF as well as host a web server on a local socket.

A complete Docker image has the following components: DockerClient, Docker Daemon, Docker Image, and Docker Container. Docker uses the client-server architectural paradigm, managing and creating Docker containers with remote APIs. [21] Docker containers are created from Docker images. Clients and servers can run on either a single machine or through a socket or RESTful API.

Fig. 2. SQLite.

Fig. 4. Docker architecture.

III. PROJECT DESCRIPTION

In the initial stages of the project, we realized that one of the main difficulties of developing and running software for a SDR is configuring the environment. The options available were to use Pentoo Linux, which came with GNU Radio and some base packages pre-installed or to download GNU Radio on another Linux distribution and build the dependencies from

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IEEE 802.11 (2.4 GHz &amp; 5 GHz)</th>
<th>IEEE 802.15.1 (Bluetooth)</th>
<th>IEEE 802.15.4 (Xbee)</th>
<th>LTE/GSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Simplicity</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Interoperability</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Ease of Use</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Security Consideration</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Cost</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>1.0</td>
<td>4.0</td>
<td>3.6</td>
<td>3.0</td>
</tr>
</tbody>
</table>
scratch. With the objective to enable users to utilize our application in whatever environment they prefer, we pursued the latter option and found that building dependencies from source was a process that took hours and required significant technical knowledge. In order to achieve the goals of the project, we decided that neither of these solutions was acceptable because we wanted the application to be portable and usable on a wide range of systems and we wanted initial setup to be quick and easy.

![Concept of operations (CONOPS)](image)

**Fig. 5.** Concept of operations (CONOPS) shows the initial ideas of the SUPERFREQ Docker Image. While the core functions of the HackRF, database, terminal, and Flask framework are still present in the final product, changes to these operations were made. Notably, the App Image was replaced naturally by the Docker Image and Command Line/Flask interface were catered to such as change. As a side note, the Attacker in the network was meant to showcase the security aspect in using a HackRF. Due to time constraints and network complexity, it was determined too expensive to have an auto generated security report.

Our solution was to utilize a Docker container which pulls from a pre-built Docker image that is publicly accessible on Dockerhub that contains a GNURadio installation as well as all tools necessary to receive and interpret signals from the HackRF. The SUPERFREQ Dockerfile automatically downloads this prebuilt image and installs the SUPERFREQ app, which allows the Docker image to avoid building GNURadio and other C libraries, providing significant time savings. Once the SUPERFREQ Docker image is built and run, SUPERFREQ launches a web server to listen port 80 and provides the user direct access to the SUPERFREQ application on their local machine.

The web interface enables users to perform scans of WiFi, Bluetooth, and Zigbee protocols using the HackRF on the 2.4GHz and 5GHz bands. Once a scan has been completed, scan results are automatically aggregated by SUPERFREQ, stored in a SQLite database, and displayed in the GUI presented by the Flask web server. Results stored in the database are persistent as long as the Docker container is not removed from the host machine, and the default landing page for SUPERFREQ will allow users to view results in raw and graphical formats for any previously run scans. Optionally, an advanced user may launch a shell inside of the SUPERFREQ docker container for access to the SUPERFREQ terminal. This terminal allows users to run individual scans, print out results, and run additional filters on data directly via the command line.

**Fig. 6.** Functional block diagram. The FBD presents the importance of the backend components to the SUPERFREQ Project. The single most critical backend components for the Docker image itself was the SQLite database because of its use by both the command line interface and web (Flask) interface. The other critical component outside the Docker image was the data pathway between the HackRF, GNURadio Libraries, and ability to eventually write data to the database.

**IV. RESULTS**

![Docker Container Parsing RF Data.](image)

The SUPERFREQ project was successful in creating an easy-to-use environment to enable actionable data collection from a SDR. The web interface brings the immense range of usable options for the SDR in to a manageable list which a user can easily click and run. Beyond the front end development, the back end design of SUPERFREQ demonstrates modular use of an SDR and provides the framework for adding more features or frequencies in a user friendly python environment. Finally, the data received is accessible by a wide range of users, displaying graphically and pre-formatted in the web...
interface while still being available in raw data, packet capture, and database formats for more advanced users.

As demonstrated in Figure 7, the Docker Container is platform independent and handles the data collection and processing of this project. The three protocols we set out to scan (WiFi, Zigbee, and Bluetooth) are successfully parsed by the container, giving more information about the networking devices present.

On the other hand, Table I shows the captured scan data as it is written in the database. The data here is meant to be more comprehensive so a user could have more details on the inner workings of the network. Ideally, this information would be used for network administration but, could also be used to learn how each radio frequency sends out data according to its standards. For example, Table I shows how WiFi sends out its

<table>
<thead>
<tr>
<th>SSIDs per Channel</th>
<th>Packets Per Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As an example, Figure 8 highlights how such graphs would display information regardless of which underlying operating system the end user is working on.

Table I: WiFi Data Table.

<table>
<thead>
<tr>
<th>SSID</th>
<th>MAC 1</th>
<th>MAC 2</th>
<th>Frequency</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Graph](image.png)

Fig. 8. ChartJS Graphs Displaying Scan Data.

As an example, Figure 8 highlights how such graphs would display information regardless of which underlying operating system the end user is working on.

Table I: WiFi Data Table.

<table>
<thead>
<tr>
<th>SSID</th>
<th>MAC 1</th>
<th>MAC 2</th>
<th>Frequency</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

On the other hand, Table I shows the captured scan data as it is written in the database. The data here is meant to be more comprehensive so a user could have more details on the inner workings of the network. Ideally, this information would be used for network administration but, could also be used to learn how each radio frequency sends out data according to its standards. For example, Table I shows how WiFi sends out its corresponding data in each packet (the count is used primarily for graph output).

As showcased by the figures above, the GNURadio library, Flask micro web framework, and Docker container were vital pieces of underlying technology that allowed for a complex, interactive app environment to function cohesively. Thus, the driving goal of our project - to demystify SDR hardware and software through an approachable integrated environment - has been achieved.

Recall the biggest issue facing SDRs today: technical expertise to use the hardware in an intuitive fashion. SDRs solo aim has been to provide an easily accessible single point of control over various radio frequencies. While this has been accomplished in prior research, it has lacked both quickly deployable equipment and platform independency. However, SUPERFREQ has dramatically improved on both these issues and opened a path for others in the open source community to expand upon this work.

V. CONCLUSIONS

In this paper, we integrated various tools and libraries to create a software package named SUPERFREQ, which can collect a variety of wireless communication information, store the data, and display the data in an intuitive manner. Instead of spending hours building dependencies or booting into a live image operating system, users can easily build the SUPERFREQ image with Docker using our Dockerfile. Through a few simple steps, users are able to effectively retrieve and visualize collected data using an SDR. Our study shows that our project has significantly reduced the technical threshold, allowing users to use SDR more easily and efficiently. Furthermore, we are exploring the possibility to integrate: LTE/GSM frequencies, user preferences and settings for each frequency scan capture (e.g. let the user input a range of frequencies they wish to capture), and transmission of previously discussed frequencies to the SUPERFREQ app.

ACKNOWLEDGEMENT

We thank Joe McManus, our academic advisor, who gave us a lot of technical guidance and advice. We also would like to thank our course instructor, Dr. Kevin Gifford, for many valuable observations and useful comments. This work was supported by the Interdisciplinary Telecom Program.

REFERENCES


[15] RFSec-ToolKit, cn0xroot, github.com/cn0xroot/RFSec-ToolKit


[19] M. Ossemann. Kickstarter. Transmit or receive any radio signal from 30 MHz to 6000 MHz on USB power with HackRF. Jul 31, 2013
