Department of Information Engineering and Computer Science

Bachelor's Degree in
Computer Science

Final Dissertation

OpenRFSense - Explorative data analysis using software-defined radios

Supervisor
Fabrizio Granelli

Student
Leonardo Baldin

Academic year 2021/2022

Table of contents

Abstract

This thesis focuses on the design and implementation of a software-defined radio (SDR) platform for spectrum monitoring and analysis. The objective of the project is to provide a low-cost and scalable solution for real-time spectrum analysis and visualization. The project is based on open source technology and is aimed at being accessible and usable by individuals, businesses, and organizations which may need to monitor and understand the usage of the radio spectrum in remote locations.

This thesis consists of three logical macro-sections. The first section of the thesis provides an overview of software-defined radio technology and its role in modern radio communication systems. This includes a discussion of the hardware and software components used in SDR systems, as well as the algorithms and techniques employed to perform real-time spectrum analysis and visualization.

The second part of the thesis focuses on the design and implementation of the open source project "OpenRFSense", which was created and maintained by the author of this thesis. This section focuses mainly on the engineering of the various software components which comprise the project and the way such components are integrated.

Finally, the thesis evaluates the performance and accuracy of the SDR platform and its ability to provide a low-cost and scalable solution for spectrum monitoring and analysis. The conclusion provides recommendations for future work, including ways to further improve the accuracy and performance of the platform, as well as potential applications for similar SDR-powered software in various fields, such as wireless networking and radio communication systems.

Overall, this thesis aims to provide a comprehensive analysis of the design and implementation of an SDR platform for spectrum monitoring and analysis, as well as a useful example of an accessible, open source software suite.

1 Introduction

Dit is intro The collection and analysis of data samples from the radio frequency spectrum has always required considerable effort, even with partial automation, due to the inherent complexity and density of such data. Furthermore, complex systems based on analogic circuits were needed to process radio signals. This used to entail large investments in both manpower and capital at any scale, making spectrum analysis infeasible for hobbyists.

This thesis will elaborate on the potential of SDR technology and the value of open source software in advancing the field of radio communication systems.

1.1 Background and motivation

Software-defined radio (SDR) is a technology that has been gaining attention in recent years due to its ability to revolutionize the way we design and operate radio communication systems. The history of SDR technology can be traced back to the 1980s, when advances in digital signal processing (DSP) made it possible to perform signal processing in software. At that time, SDR was still in its early stages, and the hardware technology available was limited. In the early

1990s, the first SDR prototypes were developed [1], and the technology began to be used in military and defense applications.

As the technology progressed, the cost of digital signal processors and memory decreased, and software became more advanced. This allowed for the development of more modern SDR systems which could support a wider range of radio communication protocols and waveforms. In the late 1990s and early 2000s, SDR technology began to be adopted in the commercial sector for applications such as wireless networking and cellular communication systems.

Today, SDR technology is being used in a wide range of applications, from amateur radio to satellite communication systems, and is increasingly becoming more accessible to the general public. SDR technology is a key enabler for emerging technologies such as the Internet of Things (IoT) and 5G communication systems. The potential benefits of SDR technology are significant, including greater flexibility, reduced development time and cost, and improved spectrum utilization.

In today's world, radio frequency (RF) communication is the backbone of many critical infrastructures, with applications ranging from wireless networking to cellular communication systems. However, the increasing demand for wireless communication has led to a shortage of available spectrum, which can lead to interference and other problems. With this in mind, the motivation behind developing the OpenRFsense project is to create an experimental solution using modern SDR technologies for real-time spectrum monitoring and analysis.

1.2 Project overview

The OpenRFSense project leverages a small, high-availability application ("node") running on low-power devices (e.g. a Raspberry Pi or similar single-board computers) equipped with a SDR module and a backend service which provides centralized storage and remote control capabilities over said hardware.

The node application is designed to start on boot and run as long as the device is powered on. It's responsible for communicating useful data to the backend service on demand (such as device telemetry and radio spectrum recordings) and provide a web-based management interface aimed at facilitating normal device configuration without requiring external peripherals to be attached.

The backend service is responsible for receiving and storing spectrum data, while also providing a web-based user interface (UI). The service exposes a REST [2] interface to make data accessible to external services, while the UI allows users to manage the stored data and explore it by visualizing a playback of the recorded radio spectrum.

1.3 Related work

In recent years many SDR-powered projects have been created to monitor the spectrum in real time. One of such projects is Electrosense [3], developed independently by a team of researchers. Electrosense is a commercial product which includes a web-based visualization interface for spectrum data and receives such data by means of a custom-designed SDR board attached to a Raspberry Pi computer.

Another similar but simpler project is OpenWebRX [4], another web-based spectrum visualization platform which mainly focuses on real-time audiovisual analysis of data through a waterfall plot and noise sampling. The OpenWebRX project only supports visualizing a single data source at a time, but provides a free directory listing of all public machines which provide

radio data and a hyperlink leading to their web interface.

2 System architecture

Several different software components are required to ensure the system can:

- persist general data on disk, such as node telemetry and hardware information
- receive and store large volumes of radio signal samples organized in "campaigns"
- maintain a reliable and fault-tolerant connection between the backend and an unknown number of nodes

The currently implemented software architecture is summarized in Figure 2.1.

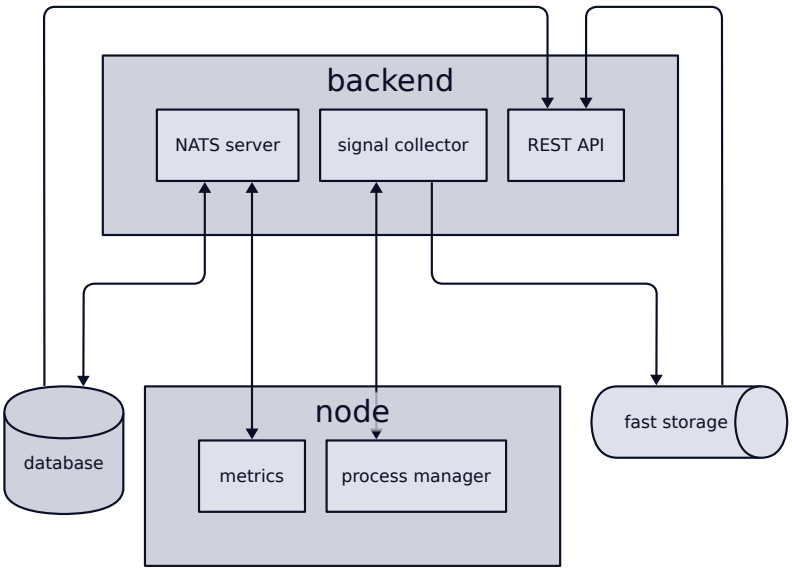

Figure 2.1: high-level system architecture

2.1 High-level components

OpenRFSense is built upon several different modern technologies, interconnected by mainstream networking protocols.

2.1.1 Hardware

OpenRFSense is designed to run on a variety of hardware platforms, including PCs, servers, and embedded systems. The hardware requirements depend on the specific use case and workload, but typically include a multi-core processor, sufficient memory and storage, and one or more software-defined radio (SDR) devices acting as signal receivers.

2.1.2 Software-defined radio

OpenRFSense relies on one or more SDR devices to capture and analyze radio frequency signals. The SDR devices are connected to the hardware platform via USB or PCIe interfaces and can be controlled using open-source software libraries such as GNU Radio. The SDR device is responsible for capturing radio signals from the environment.

2.1.3 Operating system

Thanks to the choice of programming language (Golang [5]), both the backend and node software can be compiled and deployed on most modern operating systems and hardware configurations (e.g. Windows on ARM, Linux on x86, MacOS on M1). The backend service is also developed with containerization in mind, making it possible to deploy the software securely and avoiding strict dependency on host system configuration.

2.1.4 Signal processing and analysis

OpenRFSense uses a variety of signal processing and analysis techniques to extract useful information from the captured radio frequency signals. This includes techniques such as Fast Fourier Transform [6] (FFT), digital signal processing (DSP), and machine learning algorithms. See Section 5.2 for a more in-depth explanation of the data analysis process.

2.1.5 User interface and management

OpenRFSense provides a web-based user interface for configuring, managing, and visualizing the captured data. The user interface includes features such as real-time spectrum displays, signal analysis tools, and alerting mechanisms. The system also provides APIs for integration with external tools and services.

2.2 Backend architecture

The backend service is a monolithic application which combines user interface and data management code. It can be broken down into several modular components which all depend only on shared configuration in the form of a file or environment variables on the host system.

The internal messaging service maintains a constant communication stream between the backend and the various nodes deployed by the user. It provides a scalable and fault-tolerant solution for message delivery to the remote nodes. It is documented in Chapter 3.

Another critical component is the web-based user interface (UI) which is the main interactive mean for the user to communicate with the other components inside the backend service. An in-depth explanation, complete with other high-level considerations, is contained in Chapter 4.

2.2.1 REST API

To allow external access to stored data, a REST (REpresentational State Transfer, defined in [2]) API is provided by the backend. Such a system allows authorized applications to query data through a standardized interface. Currently, the following data can be fetched from the backend:

- signal measurements taken by a specific node and belonging to a certain campaign
- a list of all nodes currently connected to the messaging system
- node metrics and system status for a specific node

The API can also actively request actions to be carried out by the nodes. To ensure bad actors cannot arbitrarily send command requests to the system, all requests which perform an action that can change the state of the system require a form of authentication. The currently implemented authentication method is Basic HTTP Authentication (a Web standard, see [7]).

2.2.2 Signal measurements storage

Since radio signal data can be dense and structured as large inbound TCP packets, a storage layer capable of extremely fast writes to memory is needed. The BadgerDB [8] key-value store was chosen due to the maturity of the software and it being natively developed in Golang, providing easier integration into the existing backend code. BadgerDB is capable, in optimal conditions (sufficient RAM and a modern solid state disk), of writing several gigabytes of data per second to disk. In practical testing, BadgerDB has proven more than sufficient for handling raw inbound signal data from several nodes at a time.

2.2.3 Database access layer

A support database is used to store persistent data, which is required to survive service outages or is generally better kept for an unspecified amount of time. PostgreSQL [9] was chosen mainly due to the maturity and notoriety of the project. From the project website:

> *PostgreSQL is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads.*

PostgreSQL is widely available and easily deployable even using container technology. A native Golang implementation of the database connection and communication protocol is used internally to store data such as past node metrics received through the messaging system, or signal measurement campaign data (start date and time, end date and time, etc.). Such data is then available for querying through the REST API documented above (in Section 2.2.1).

2.3 Node architecture

The node software is developed upon much the same technologies as the backend to ease integration between the two and reduce development overhead. Being developed to run natively on embedded devices as a self-contained application, several software components are needed to fullfil all required functions.

The application is available both as a standalone, statically-linked executable and a more user-friendly prebuilt system image based on the Raspbian OS. The system image comes pre-configured with required system components such as:

- a SystemD [10] service which starts the node management software on boot
- the OpenRFSense node management application
- the external, low-level program which interfaces with the SDR hardware (`orfs-sensor`)
- some useful signal decoding software, also part of the project

The following sections describe the main components of the node management software.

2.3.1 System metrics collector

A critical component for the node software is the metrics collector. This component is responsible for providing a current snapshot of the status of both hardware and software of the system, which will then be stored by the backend. The metrics message format and message exchange is documented more extensively in Section 3.1.

2.3.2 Process manager

The node software relies on an external process for data acquisition and delivery. Not being originally developed with integration in mind, such a program is managed as a zero-knowledge black box, executed as a child process by generating command-line arguments and passing them to the program.

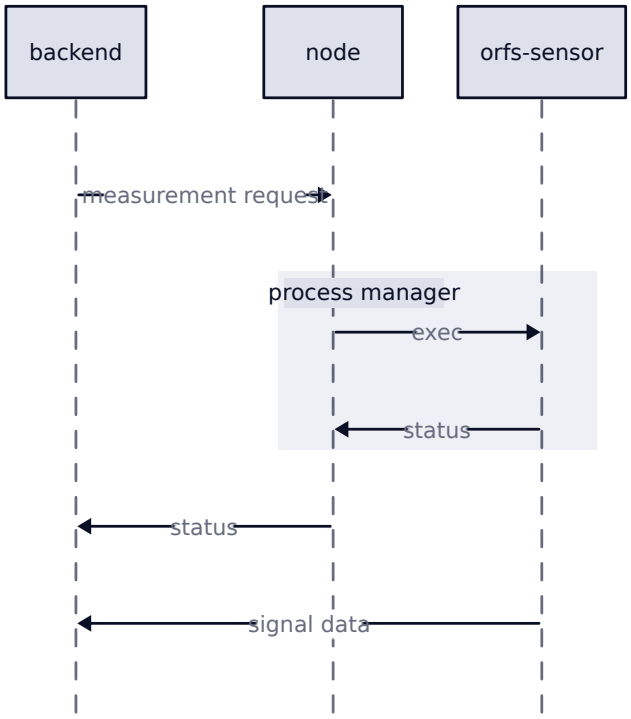

Figure 2.2: measurement process flow

2.3.3 User interface

Much like the backend, the node provides users with a web-based UI to manage system functionality. Additionally, the UI lets users configure and reboot the system without needing an external monitor or peripherals for easier system administration of embedded devices. A description of the node UI with screenshots is contained in Section 4.2.4.

The interface is only accessible on the same network as the node, to avoid exposing administration functionality to the public internet. Additionally, for thirty minutes after the system is booted, the UI is accessible by connecting to the device through a temporary WiFi access point if the hardware is capable of creating one such connection.

3 Service messaging

In OpenRFSense, NATS messaging [11] is used to enable communication between the backend and the nodes. NATS (Neural Autonomic Transport System) is a lightweight messaging system that is commonly used for machine-to-machine (M2M) communication in IoT (Internet of Things) applications. A messaging system such as NATS is based on message delivery guarantees and optional message retention for later delivery. These capabilities ensure service message resiliency through high-latency connections and in-flight data loss.

The backend acts as the NATS server and the nodes act as NATS clients which connect

to the server. When a node is powered on, it connects to the NATS server and subscribes to a "subject", which is a named destination to which messages are published. The node can subscribe to one or more subjects depending on its capabilities. Some subjects currently being used for service messages are:

- `node.all` for node system metrics (see Section 3.1)
- `node.all.aggregated` and `node.all.raw` for backend-to-node signal recording commands (see Section 3.2)
- `node.$hardware_id.$command`, where `$hardware_id` is the node's unique identifier and `$command` is a specific action being requested

3.1 Node metrics

The backend, when requested through the REST API or the web UI, sends a "status request" message to a specific NATS subject (to which all nodes are required to listen on). All nodes then start collecting internal metrics (such as CPU usage and temperature, memory usage etc.) and respond on the same subject with the collected data, encoded in JSON (JavaScript Object Notation) format [12].

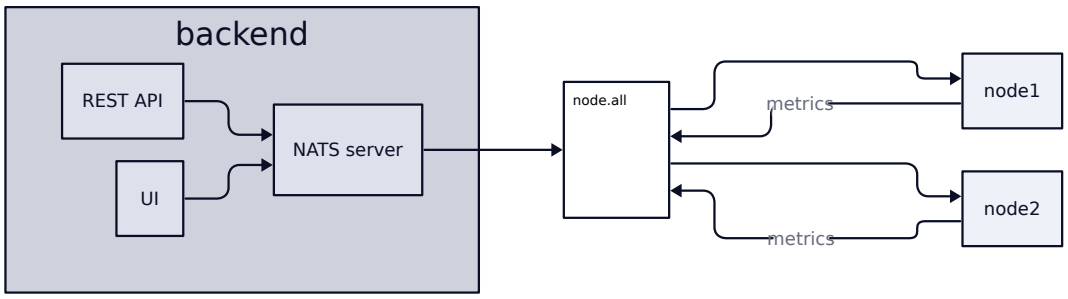

Figure 3.1: node metrics request flow

3.1.1 Metrics contents

A metrics object contains useful system data such as:

- the system hostname
- a unique hardware ID derived from the motherboard model
- the motherboard/hardware model of the system
- the system's online time (uptime) in milliseconds since it was powered on

Additionally, a variable-content section is reserved for less crucial information. This section is never required nor checked by the backend, but its data, if present, is shown to the user through the web UI. Currently, such data is provided by small standardized software modules ("providers"), which are called by the node management software if present. Some currently implemented providers are:

- node geographical location in GeoJSON [13] format
- filesystem usage

- memory (RAM) usage
- network connections
- current node availability (free, busy, handling errors, etc.)

3.1.2 Message encoding

As stated above, node metrics are encoded in JSON format to be easily accessible by external services, since JSON is a well-established industry standard data format. An example of the metrics data encoded in textual JSON is Listing 3.1.

Listing 3.1 example metrics object in JSON format

```
1 {
2   "id": "kgslnximugwhsfnwbjknwbv",
3   "hostname": "raspberry",
4   "model": "Raspberry Pi 3B",
5   "uptime": 186339250000000,
6   "providers": []
7 }
```

3.2 Commands

The backend can also publish messages to node-specific subjects. This enables the backend to remotely control the nodes and send configuration updates or commands. The messages can be one of two types:

1. empty messages sent on a specific NATS subject ("pings")
2. JSON-encoded messages containing relevant context for the requested action

When a node receives a remote command message, it processes the command and sends a response message back to the backend. The response message can contain any relevant information about the status of the command, such as success or failure, and any associated data. Generally, nodes send their system metrics as response but it is not a strict requirement.

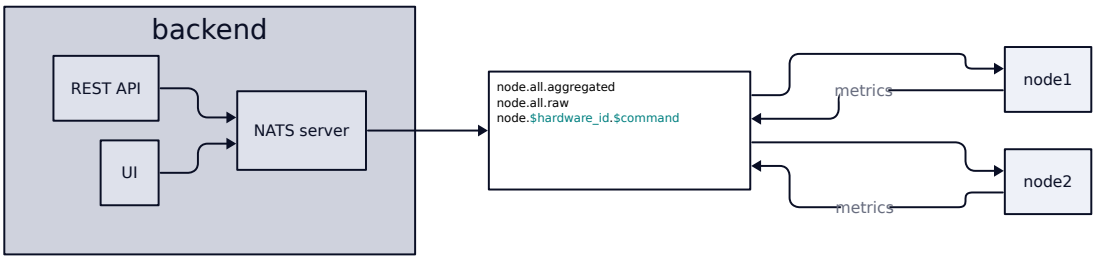

Figure 3.2: sensor campaign command flow

4 User interface

The user interface plays an essential role in enabling users to interact with the system effectively. A well-designed user interface can make it easy for users to navigate between pages and perform several complex tasks related to node system and data management.

The UI is served to the user as a collection of web pages by the backend service. The pages are generated on demand with up-to-date information and take advantage of modern web technologies.

4.1 Design principles

Several design principles guided the development of the user interface, mainly:

1. Consistency: Elements such as fonts, colors, and icons are consistent throughout the interface, making it easier for users to navigate between UI sections and use the system.
2. Simplicity: The user interface is simple and easy to use. The simplicity principle ensures that the interface is not cluttered with unnecessary elements, making it easy for users to find what they need.
3. Feedback: The user interface provides users with immediate feedback to let them know that their actions have been registered. This principle ensures that users are not left wondering whether their actions have been successful.
4. Accessibility: The user interface is designed to be accessible to all users, including those with disabilities. This principle ensures that users with different abilities can use the system without encountering any usability barriers.

4.2 Interface components and structure

The UI is aimed at users with a certain technical background and previous knowledge of the system, but it is designed to be easily navigated through by any given user.

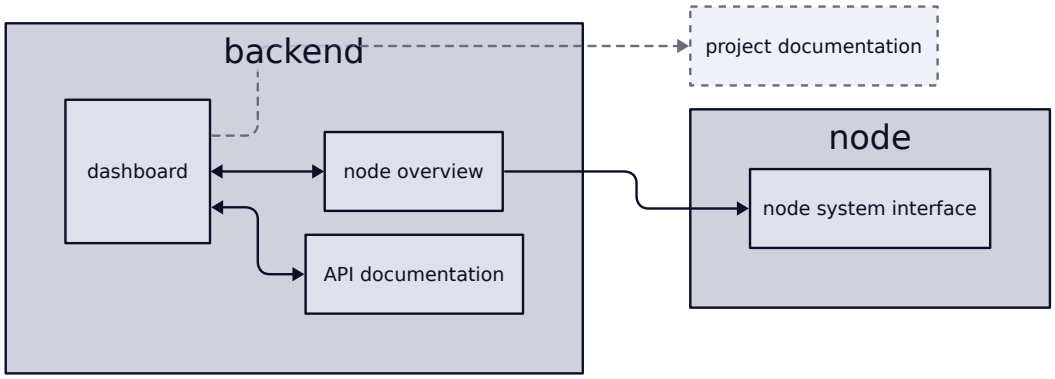

Figure 4.1: UI hyperlink structure

4.2.1 Dashboard

The dashboard acts as the homepage of the UI. It is the first page served by the backend on the root address and is the first interactive element the user will be shown on connection to the system.

The first element presented by the UI is a table containing useful information about all nodes connected to the backend, such as node location and unique identifier. Each node can be selected with a checkbox to take part in a measurement campaign (configurable through a pop-up modal dialog). Every row also contains a link to a page which visualizes a node's metrics and location in more detail (see Section 4.2.2).

The homepage of the UI provides a map of the world, with the locations of all the sensors in the OpenRFSense network indicated by markers. Clicking on the markers yields basic information about each node, such as its ID and the last time it was active, and a link the node-specific overview page.

4.2.2 Node overview page

A node-specific overview page is also generated for each node connected to the backend and the messaging system. This page generally contains all required node metrics returned by a request to the node, as well as more detailed node system information if available (see Section 3.1). The node overview page also contains a map but only showing a single marker corresponding to the node being queried and centered on the node's location, if provided.

A table containing a list of all past and ongoing measurement campaigns is also provided to the user, with management controls on each row to download the signal data in various formats or delete it from the storage. The controls are non-interactive if the campaign is still ongoing.

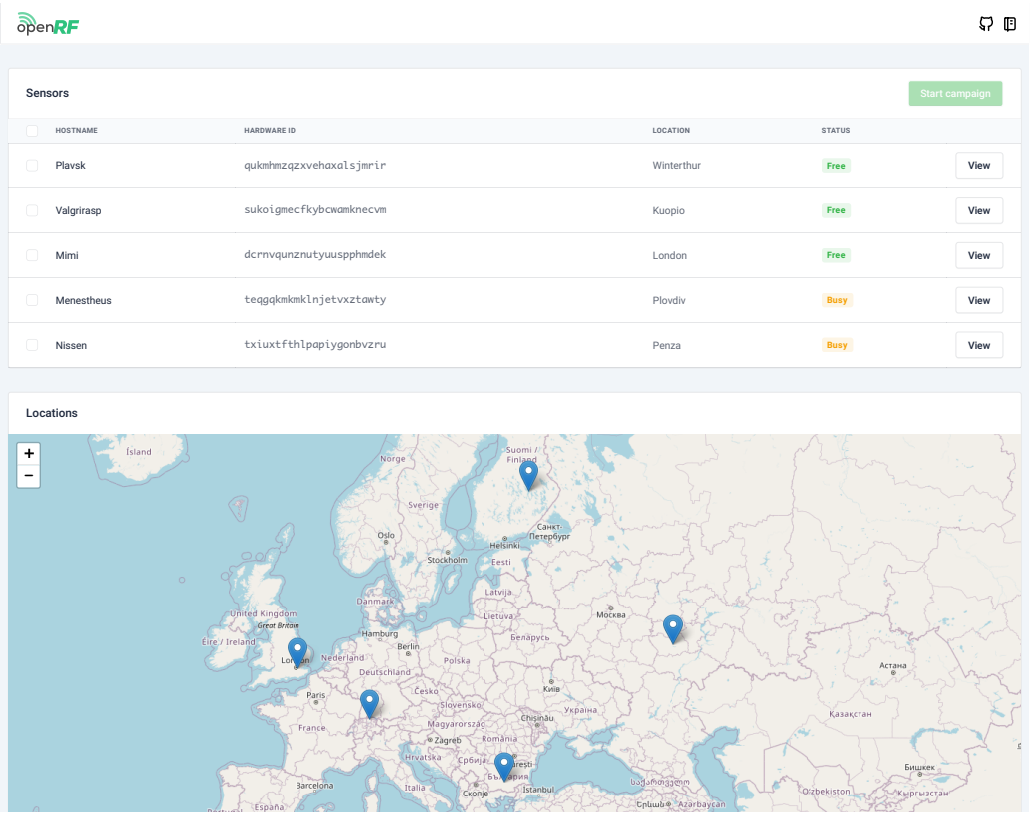

Figure 4.2: Dashboard

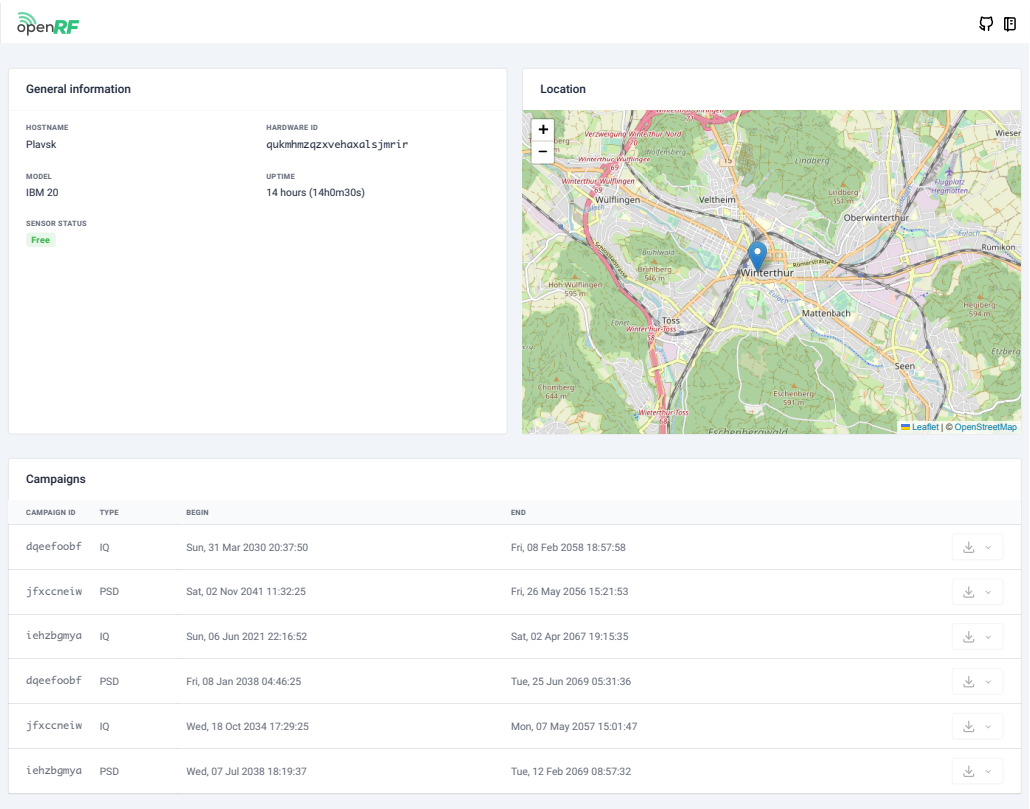

Figure 4.3: Node overview page

4.2.3 API documentation

To make the REST API (defined in Section 2.2.1) easily accessible to users and external applications, a documentation page is automatically generated using Swagger (OpenAPI specification version 2.0 [14]). The generated page contains all the necessary documentation to query data and send commands to the backend, complete with examples and request and response formats in textual JSON.

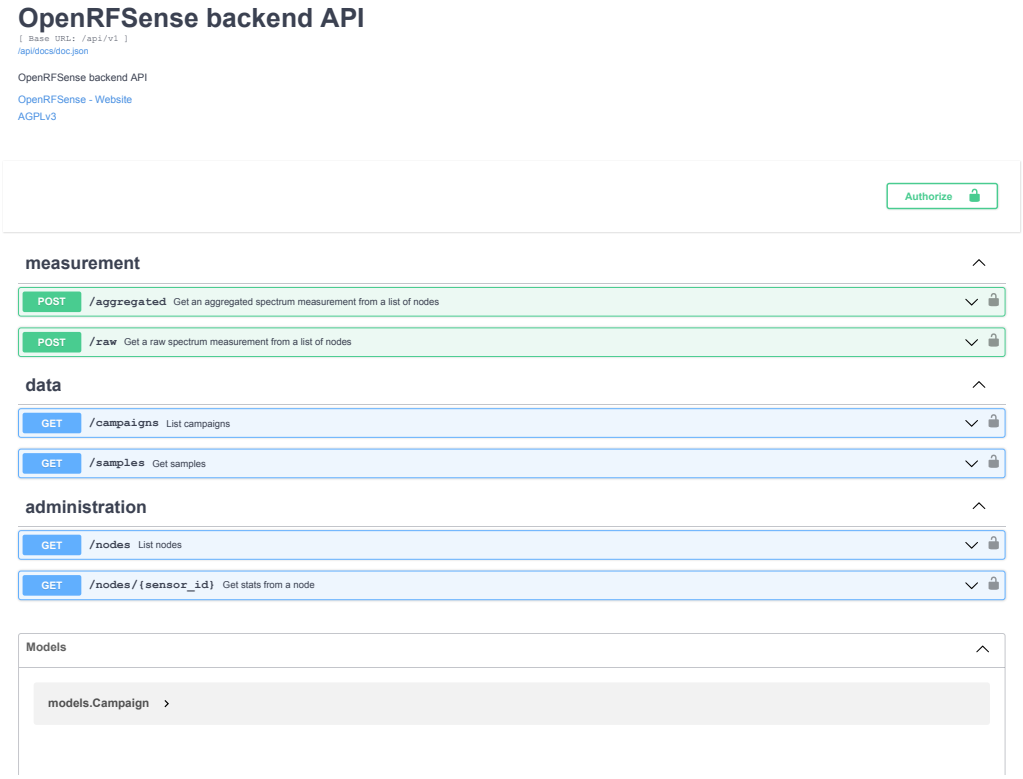

Figure 4.4: API documentation

4.2.4 Node system interface

Each node's local management UI provides a high-level management over the host system running the node. As such, it must present important information in logical order, starting with connection status (both WiFi and physical). On remote devices, this facilitates troubleshooting and provides an alternative method of restoring connection to a node.

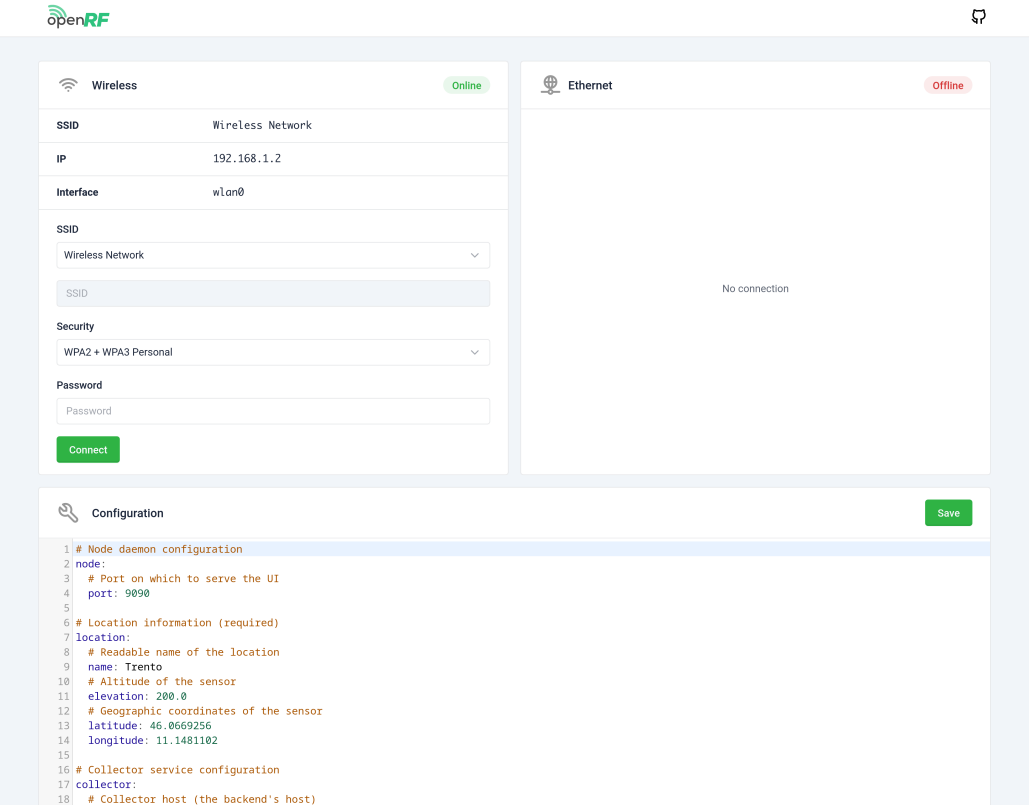

Figure 4.5: Node system management page

A simple text editor is also provided for the node management software configuration. Each change followed by a click on the "Save" button will write the changes to the configuration file and reboot the system.

5 Data collection and analysis

Data collection is at the core of the OpenRFSense project and, as such, requires some complex and specialized components. Overall, the data collection procedure can be better explained as an ordered pipeline, which starts from the remote node.

5.1 Data acquisition

The node software utilizes a set of low-level hardware APIs to interface with the SDR receiver(s) and collect the RF signals. The main data collection routine is handled by a customized version of a pre-existing Electrosense software, which interfaces with connected SDR devices and streams data towards an external destination.

The data is collected in one of two forms:

- raw spectral density ordered by frequency
- spectral density averaged over an arbitrary number of seconds using Fast Fourier Transform (FFT)

5.2 Signal processing

The signal processing component is responsible for cleaning up the collected RF signals and extracting the relevant features for analysis. The component utilizes a set of algorithms to filter out noise and interference from the collected signals. Once the signals are cleaned up, the component extracts relevant features such as the signal strength, modulation type, and frequency. At the time of writing, the following signal encodings can be preprocessed:

- Aircraft Communications Addressing and Reporting System (ACARS)
- LTE, both User Equipment-to-cell and cell-to-User Equipment
- Mode S [15] collision avoidance messages
- Aeronautical Information Service (AIS) air traffic information
- AM/FM radio

5.3 Data streaming

The processed data is serialized in a JSON-based binary format (Apache Avro [16]), together with extra metadata such as:

- time of recording
- node hardware identifier and campaign identifier code
- sensor hardware configuration (center frequency, antenna gain, estimated noise floor, etc.)

The encoded packets are then sent over the Internet to the backend service through several raw TCP connections per receiver to maximize flow. Surprisingly, even low-cost modern SDR receivers tend to output large quantities of data (usually in the order of several megabytes per second). This requires the usage of clever throughput maximization algorithms such as TCP Fast Open [17] and `SO_REUSEPORT`.

5.4 Data storage

The cleaned up and feature-extracted data is received by the backend and partially deserialized to extract the node hardware identifier and campaign identifier code, which are used to derive a unique key for that packet to be used in storage. The binary-encoded data is then stored in a database embedded in the backend for later analysis. The data is written to disk in a structured format that allows for fast querying and analysis. The storage software component is explained in more detail in Section 2.2.2.

5.5 Analysis of spectrum data

To analyze the collected spectrum data, several techniques can be employed. One approach is to use statistical methods to identify patterns or anomalies in the data. For example, signal strength and frequency distribution can be analyzed to detect unusual patterns, which may indicate the presence of interference sources. Another approach is to use machine learning algorithms to automatically classify different types of signals based on their frequency, bandwidth, and modulation. This can help to identify different types of signals, such as WiFi, Bluetooth, or cellular signals, and to detect any new or unknown signals.

Overall, the collected spectrum data is a valuable resource for a wide range of applications. The different techniques for analyzing the data can provide useful insights into the electromagnetic environment, giving users easy access to dense and reasonably accurate research samples.

6 Conclusion and future work

OpenRFSense was developed as an experimental software suite powered by modern software-defined radio technology. The following sections contain some closing remarks and reflections about the project and the status of the SDR-based data analysis as a whole.

6.1 Evaluation of system performance and limitations

As with any project, OpenRFSense has its limitations that must be acknowledged. Below are some of the limitations of the project.

1. Coverage: OpenRFSense relies on users who can install and maintain their own sensors. As a result, there may be regions where the sensor density is low, leading to gaps in the collected data.
2. Sensor accuracy: The accuracy of the collected data is dependent on the quality of the sensors installed. Low-quality sensors may produce unreliable data that could affect the accuracy of the analysis and interpretation of the collected data.
3. Signal interference: Interference from other radio sources can tamper with the accuracy of the collected data. For example, electromagnetic radiation from other devices or sources could create a noisy signal that can be challenging to analyze.
4. Cost: The cost of installing and maintaining the nodes could be a restraining factor, especially in regions where resources are limited.
5. Security: The sensors collect data from the environment, and ensuring the security and privacy of the collected data is crucial. The project must ensure that the data collected is not misused or accessed by unauthorized individuals.
6. Data quality: The data collected may not always be of high quality due to a variety of reasons such as environmental conditions, sensor errors, and data transmission errors.

Therefore, it is essential to have mechanisms in place to ensure data quality control and filtering.

6.2 Applications of the project

The collected spectrum data can be used for a wide range of applications, including radio frequency interference (RFI) identification, spectrum occupancy measurements, and wireless network planning.

RFI identification is a major application of spectrum data. OpenRFSense provides a real-time view of the electromagnetic environment, allowing users to detect and locate interference sources. The data can also be used for spectrum monitoring, such as measuring the spectrum occupancy of various frequencies. This information can be used to identify underutilized spectrum bands, which can then be allocated for new services or applications.

Another important application of the data is wireless network planning. OpenRFSense provides accurate and up-to-date information on the electromagnetic environment, allowing users to identify optimal locations for wireless access points or base stations. The data can also be used to optimize the configuration of said networks, such as adjusting the transmission power or selecting the most appropriate frequency band.

Hobbyists and researchers have also been able to carry out more complex tasks thanks to SDR receivers, such as satellite tracking and image reception [18]. This proves SDR-sourced data to be useful even for longer range and higher gain spectrum analysis.

6.3 Future directions for research and development

Being released as open-source software, OpenRFSense is not intended to be a finished, industrial-grade product. Some future areas in which the project could be improved or may need additional research are the following.

1. Integration with other sensor networks: OpenRFSense can be integrated with other sensor networks, such as those monitoring air quality, water quality, and weather. Integration with these networks can provide a more comprehensive picture of the environment and the impact of electromagnetic radiation on it.
2. Machine learning and AI: OpenRFSense can be enhanced with machine learning and artificial intelligence (AI) algorithms to improve the accuracy and speed of data analysis. These algorithms can help identify patterns and anomalies in the data and assist in making predictions and recommendations based on the data.
3. Real-time monitoring: The current version of OpenRFSense only provides playback of recorded spectrum data. However, further development can be done to improve the latency and accuracy of the system, at the point of providing users with real-time signal visualization. This can require the use of more advanced sensors and improved data transmission technologies.
4. Visualization and data analysis tools: OpenRFSense generates a large amount of data, and tools are needed to help users visualize and analyze the data. This can involve the development of new visualization techniques, such as heat maps and trend analysis, as well as ad-hoc data mining tools and other task-specific analysis tools.
5. Standardization and data sharing: OpenRFSense can benefit from standardization of data collection and sharing protocols, such as a universal format for spectrum data storage. This can help ensure that data collected by different sensors and networks can be easily shared and combined to provide a more comprehensive picture of the environment.

Appendix A Code structure

All the code for the OpenRFSense project is hosted on Github, a free Git repository hosting service with additional code and community management tools. At the time of writing, the various components are placed in the following repositories under the OpenRFSense organization:

- `backend`: source code and tooling for the backend service
- `node`: source code and tooling for the node management software
- `orfs-sensor`: source code for the signal recording software
- `image`: various scripts and tooling to generate the pre-configured system image for Raspberry Pi boards
- `openrfsense.github.io`: technical documentation for the project (see Appendix C)
- `common`: shared code which would otherwise be duplicated in both node and backend

Appendix B Project branding

Special care was put behind giving an identity to the OpenRFSense project through graphic design. A simple but meaningful logo contains the abbreviated project name and a round symbology reminiscent of WiFi-associated commercial products.

Figure B.1: Official OpenRFSense logo

The color palette has enough contrast to be accessible but maintains at least one saturated color.

Figure B.2: Color palette

Appendix C Technical documentation

All software components for the OpenRFSense project contain self-documenting code with numerous comments and additional language-specific documentation tooling (`godoc`). Moreover, additional user-oriented technical documentation has been written and is deployed automatically on Github's static file hosting service (Github Pages).

The documentation is written in Markdown format and is rendered to a static website using the Jekyll program. It includes code blocks with ready-to-use examples to prepare deployments and configure the services before starting the system, together with overviews of the various components. All pages also feature a search widget which facilitates navigation through the text contents.

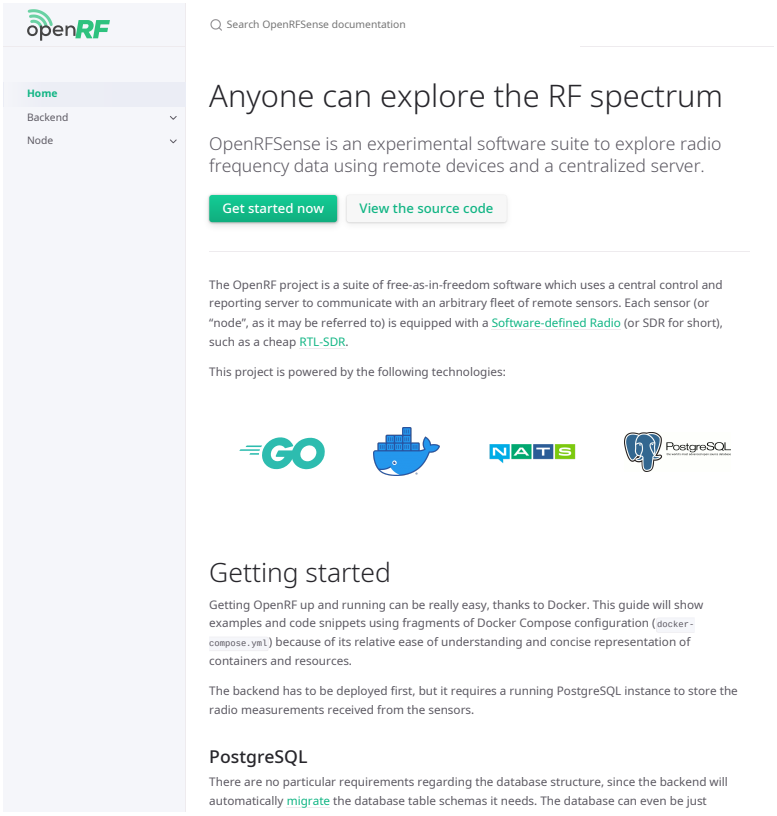

Figure C.1: OpenRFSense documentation

Bibliography

[1] J. Mitola, "Software radios-survey, critical evaluation and future directions," in *[Proceedings] NTC-92: National Telesystems Conference*, 1992, pp. 13/15–13/23, doi: 10.1109/NTC.1992.267870 [Online]. Available: http://ieeexplore.ieee.org/document/267870/

[2] R. T. Fielding and R. N. Taylor, "Architectural styles and the design of network-based software architectures," University of California, Irvine, 2000.

[3] S. Rajendran *et al.*, "Electrosense: Open and Big Spectrum Data," *IEEE Commun. Mag.*, vol. 56, no. 1, pp. 210–217, Jan. 2018, doi: 10.1109/MCOM.2017.1700200. [Online]. Available: http://ieeexplore.ieee.org/document/8121869/

[4] A. Retzler and J. Ketterl, "OpenWebRX web-based software defined radio," 2018. [Online]. Available: https://www.openwebrx.de

[5] "The Go Programming Language." [Online]. Available: https://go.dev/

[6] M. Heideman, D. Johnson, and C. Burrus, "Gauss and the history of the fast fourier transform," *IEEE ASSP Mag.*, vol. 1, no. 4, pp. 14–21, Oct. 1984, doi: 10.1109/MASSP.1984.1162257. [Online]. Available: http://ieeexplore.ieee.org/document/1162257/. [Accessed: Feb. 27, 2023]

[7] R. Fielding and J. Reschke, "Hypertext transfer protocol (HTTP/1.1): Authentication," RFC Editor / RFC Editor; Internet Requests for Comments, RFC 7235, Jun. 2014 [Online]. Available: http://www.rfc-editor.org/rfc/rfc7235.txt

[8] "BadgerDB." [Online]. Available: https://dgraph.io/docs/badger

[9] P. G. D. Group, "PostgreSQL," Feb. 25, 2023. [Online]. Available: https://www.postgresql.org/

[10] "Systemd." [Online]. Available: https://www.freedesktop.org/wiki/Software/systemd/

[11] "NATS." [Online]. Available: https://nats.io/about

[12] ECMA, *ECMA-404: The JSON data interchange syntax*. Geneva, Switzerland: ECMA (European Association for Standardizing Information and Communication Systems), 2017 [Online]. Available: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/

[13] H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, and T. Schaub, "The GeoJSON format," RFC Editor / RFC Editor; Internet Requests for Comments, RFC 7946, Aug. 2016.

[14] "OpenAPI Specification - Version 2.0 | Swagger." [Online]. Available: https://swagger.io/specification/v2/

[15] I. C. A. Organization, Ed., *Manual on mode S specific services*, 2nd. ed. Montreal: ICAO, 2004.

[16] "Apache Avro Documentation." [Online]. Available: https://avro.apache.org/docs/

[17] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan, "TCP fast open," in *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, Dec. 2011, pp. 1–12, doi: 10.1145/2079296.2079317 [Online]. Available: https://dl.acm.org/doi/10.1145/2079296.2079317

[18] D. J. M. Peralta, D. S. D. Santos, A. Tikami, W. A. D. Santos, and E. W. R. Pereira, "Satellite Telemetry and Image Reception with Software Defined Radio Applied to Space Outreach Projects in Brazil," *An. Acad. Bras. Ciênc.*, vol. 90, no. 3, pp. 3175–3184, Sep. 2018, doi: 10.1590/0001-3765201820170955. [Online]. Available: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0001-37652018000603175&tlng=en