Radio-Mapper Project Analysis

Objective and Overview

The **Radio-Mapper** project aims to map the locations of radio signal transmitters by using a distributed network of low-cost Software-Defined Radios (SDRs) (specifically RTL-SDR dongles attached to Raspberry Pi devices) [1]. In essence, multiple SDR-equipped nodes are deployed across a city to continuously capture raw IQ (in-phase/quadrature) signal samples on certain frequency bands of interest. These time-synchronized measurements are then aggregated and processed to **triangulate the source of radio emissions** via Time Difference of Arrival (TDoA) techniques, ultimately plotting the estimated transmitter positions (stationary or moving) as markers on a map [1]. A public-facing web interface (planned at **radio-mapper.org**) would display these results on a Google Maps overlay, allowing users to visualize where signals are coming from and apply filters by frequency or signal type [2]. This collaborative approach essentially crowdsources RF signal monitoring and direction-finding, with the long-term vision of a community-driven radio signal map.

The motivation for Radio-Mapper stems from the desire to create an inexpensive, automated system for radio direction finding and spectrum mapping that anyone can participate in. Traditional direction-finding often requires specialized equipment or manual "fox hunts" by hobbyists; Radio-Mapper proposes an automated network that could continuously locate transmitters across a wide area. By leveraging numerous distributed RTL-SDR sensors, the system can detect and **locate unauthorized transmitters, interference sources, or interesting signals in real time**, providing situational awareness of the RF environment in a manner that was previously the domain of government agencies or well-funded enterprises.

Data Collection and Processing Pipeline

To achieve its goals, Radio-Mapper must carefully design a **data collection and processing pipeline** that handles the synchronized gathering of IQ samples and the subsequent localization computations. The high-level pipeline involves several stages, from signal capture at the distributed nodes to data aggregation and final geolocation:

1. **Distributed Signal Capture:** Each node (Raspberry Pi + RTL-SDR) is deployed at a known fixed location (with latitude/longitude). The SDR hardware at each node is tuned to monitor specific frequency ranges of interest (e.g. public safety bands, amateur radio bands, etc.) [1]. The nodes digitize raw IQ samples of any signals present. To cover multiple frequencies, the nodes may periodically retune or scan through frequencies (the project references using a special branch of the RTL-SDR driver that allows rapid switching between a **synchronization signal** and the target signal of interest [3]). Each node effectively time-stamps its IQ samples, either via a shared clock or by capturing relative timing against a reference.

2. **Time Synchronization:** A crucial challenge is synchronizing the sampling clocks of all distributed SDRs. **TDoA positioning requires all receivers to share a common time base**, since we need to

compare arrival times of the same signal across different receivers [4] [5]. Radio-Mapper explores using an external reference radio signal as a timing beacon instead of relying solely on GPS or internet time sync [3] [6]. The idea (inspired by prior research) is that all nodes periodically listen to a strong, known transmitter (the "sync" signal) – for example, a TV tower, cellular broadcast, or a dedicated reference transmitter – to calibrate their clocks. The custom RTL-SDR code included in the project allows the SDR to quickly switch from the sync frequency to the frequency of the unknown **source of interest**, capturing both nearly back-to-back [3]. By doing this, each node can record the phase or timing of the reference signal and the target signal within the same timeframe. This provides a common reference point across all nodes.

Why not just use NTP or GPS? Standard Network Time Protocol (NTP) synchronization across the internet is only accurate to about 1 millisecond at best, which would introduce on the order of **±300 km of location error** in a TDoA calculation – clearly unacceptable [7]. Even Precision Time Protocol (PTP) over a local network can reach ~100 nanoseconds under ideal conditions (still ~30 m error) and is difficult to achieve over the public internet [8]. GPS-disciplined oscillators on each receiver can provide ~1 μs timing (≈300 m error) [6] and are used in some systems (e.g. KiwiSDR network uses GPS for TDoA sync [4]), but adding GPSDO hardware to every node increases cost and complexity. Radio-Mapper's approach leverages **ambient wideband signals** as timing references: if there is a broadcast signal that all nodes can receive (such as a strong TV station, cell tower, or DAB radio signal), the system can use it as a common clock tick [9]. All nodes measure their local time offset relative to this reference signal (e.g. by comparing the phase of the reference's known modulation or pilot tone). Immediately after, they switch to the target frequency and record the unknown signal, effectively time-stamping the unknown signal's arrival in terms of the reference signal's phase. This clever approach uses the radio environment itself to synchronize distributed receivers, achieving sub-microsecond synchronization without additional hardware [9].

Time Sync Method	Approx. Timing Accuracy	Equivalent Localization Error
Internet NTP	~1 ms	±300 km (too high for TDoA) [7]
Local PTP (wired network)	~100 ns	~30 m [8]
GPS-disciplined clock	~1 μs	~300 m [6]
RF Reference Signal (TDoA)	*sub-μs (phase alignment)*	*Target ~100 m or better* [10]

1. **Data Upload and Aggregation:** Once each node has collected a block of IQ samples (both reference and target signal snapshots), this data (or some distilled form of it) is sent to a central server or cloud service for processing. In an initial implementation, data might simply be stored locally and batch-uploaded, but ultimately the vision is for near real-time streaming of processed results. The **central aggregator** receives IQ data from at least three distinct nodes for any given signal event, along with metadata (timestamps or reference phase measurements, frequency, node GPS location, etc.). The project's plan is to only forward **"higher level" processed data** rather than raw IQ continuously [11]. For example, nodes might perform some pre-processing such as filtering or detecting the signal's arrival time locally, then send those time-stamps or trimmed IQ segments instead of overwhelming the network with full raw samples.

2. **Time Difference of Arrival Computation:** The server uses the data from multiple receivers to compute the TDoA for the signal of interest. If a common reference signal was used, the task is to

compare the phase/timing of the unknown signal as received at each node, after accounting for each node's measured offset relative to the reference. In practice, this involves **cross-correlating** the IQ recordings from different receivers. By sliding one signal's sample stream relative to another's, the cross-correlation will peak when the streams are aligned – the amount of offset at the peak directly gives the **time difference of arrival** between those two receivers. Using a reference transmitter simplifies this, as the reference provides a phase zero-point for all receivers (each node knows how offset its clock was when the unknown signal hit, because it recorded where the reference's phase was at that moment [12]). With or without a separate reference, the algorithm needs to correct for any known fixed delays (e.g. processing or cable delays) and possibly Doppler shifts if the source or receivers are moving. The project references academic work that **implements TDoA using known-location transmitters as timing references** and applies several corrections for timing and frequency errors on inexpensive RTL-SDRs[5] [13] – these techniques would be applicable in Radio-Mapper to improve accuracy.

3. **Geolocation (Multilateration):** Once the time differences are calculated for a signal across multiple pairs of receivers, the system can solve for the transmitter's location. The classic result of a TDoA measurement from two receivers is a hyperbolic curve of possible locations (where the difference in distances to the two receivers is constant). With three or more receivers, multiple hyperbola curves are obtained, and their intersection pinpoint the source location [9]. In practice, the server can use the time differences to form equations and then apply multilateration algorithms (for example, a linearization and least-squares solution, or more complex iterative techniques) to compute the latitude/longitude of the source. The expected accuracy depends on receiver geometry and timing precision – with good geometry and ~100 ns timing, an accuracy on the order of tens of meters is feasible [8]. Empirically, an early experiment that inspired this project demonstrated ~100 m accuracy using three synchronized RTL-SDRs in Europe [10], which is a target benchmark for Radio-Mapper. It's important to note that in dense urban environments, multipath propagation can introduce errors (signals bouncing off buildings can arrive later than direct paths). The system may need to mitigate this by ignoring obviously delayed echoes or using algorithms that can identify the direct-path signal among multipath arrivals (e.g. by looking at the cross-correlation peak sharpness or consistency across multiple runs).

4. **Visualization and User Interface:** The final stage is presenting the results. The Radio-Mapper plan is to aggregate results on a web interface with a map overlay [11]. Each identified transmitter can be shown as a pin or heatmap on the map at its estimated location. The interface would likely allow selecting a frequency or signal type filter to view specific transmissions (for example, only show active FM broadcast stations, or highlight a particular emission frequency). Because the system can handle moving signals as well, the backend might track those over time and update their coordinates (this could enable showing a track or trajectory for a moving transmitter like a vehicle-based signal). The **public website** not only visualizes the data but also serves as a community resource – for instance, one could click on a mapped signal to see additional info or even listen to an excerpt of the signal if available. The use of a Google Maps API or similar ensures familiarity and the ability to overlay on real geography [2]. Privacy and security are considerations here: presumably only non-sensitive bands or authorized spectrum monitoring would be done, since locating transmitters can raise legal issues if misused. However, many applications (as discussed later) are benign or beneficial.

Throughout this pipeline, a lot of processing happens behind the scenes. The **IQ data** might be pre-processed with digital signal processing techniques – for example, filtering to isolate the particular signal of interest from noise, decimating sample rate to just what's needed (to reduce data size), and possibly detecting the signal's onset to mark an exact arrival time. The project's focus on IQ recordings suggests they keep as much raw information as possible for flexibility [1]. The heavy lifting of cross-correlation and multilateration is done in software at the server. These algorithms need to be efficient to handle potentially many signals and nodes; cross-correlation can be accelerated by using FFT convolution methods for longer signals, and the multilateration solution can use closed-form approaches for speed if needed. Given the distributed nature, the system could even offload some computations to the edge (each Pi could, for instance, cross-correlate its own data with the reference signal and just report a time offset). Such design decisions would evolve as the project matures.

Signal Processing and Algorithms

At the core of Radio-Mapper is the **Time Difference of Arrival (TDoA)** algorithm for radio direction finding. TDoA is a well-established technique in radio navigation and signal triangulation: if a signal emanates from an unknown transmitter, and you have multiple receivers at different locations that all capture the signal, any differences in the reception time can be attributed to the different path lengths from the transmitter to each receiver. Mathematically, the locus of possible transmitter positions for a given time difference (between two receivers) is a hyperbola on the map. With three or more receivers, the true location is where all the corresponding hyperbolic curves intersect. Radio-Mapper embraces this concept to compute latitude/longitude of emitters [9].

Cross-correlation is the primary signal processing tool used to measure time differences. After nodes record IQ samples of the target signal (ensuring they are time-aligned via the reference as described), the system compares the signal from receiver A vs receiver B, A vs C, and so on. If the signals are identical except for a time delay, their cross-correlation will show a clear peak offset from zero by that delay. For example, if receiver A's signal is correlated against receiver B's signal, a peak at +5 samples means B's copy lags A's by 5 samples (i.e. B received it 5 samples later than A did). With sample rate known, that translates to a specific time difference (e.g. 5 samples at 2 MHz sample rate is 2.5 µs). Cross-correlation inherently also handles any slight frequency differences by finding the best alignment; however, significant frequency drift (from local oscillator offsets) can smear the correlation peak. Thus, part of the algorithmic effort goes into **frequency error correction** – techniques like resampling one signal or using an FFT-based approach can correct small frequency offset between dongles. The referenced thesis by Krüger (2017) implemented multilateration with RTL-SDRs and discusses methods to correct timing and frequency errors to achieve accurate results on inexpensive hardware [13], likely including applying calibration using known signals and perhaps compensating for each dongle's oscillator drift.

Once pairwise time differences (Δt for each pair of receivers) are obtained, the system formulates a set of hyperbolic equations. For receivers at known coordinates (x1, y1), (x2, y2), etc., and an unknown transmitter at (x, y), the difference in distance (and thus time) between receiver1 and receiver2 is c·Δt (where c is speed of light). This gives an equation: distance((x,y),(x1,y1)) – distance((x,y),(x2,y2)) = c · Δt_12. Similar equations exist for other pairs. The solver can use one receiver as a reference (say, the one that got the signal first) and subtract its range – this yields two hyperbolas for a three-receiver case. Solving these equations can be done analytically or via numerical methods. In practice, a **least-squares multilateration** approach might be used to find the best-fit intersection point that minimizes error across all pairs. If more than three

receivers are available, the system becomes overdetermined and thus can average out noise for a more robust solution (e.g. using the Chan or Taylor series estimation methods known in TDoA literature).

One important algorithmic aspect is the **synchronization via reference signal**. Radio-Mapper's use of an ambient sync signal effectively turns the problem into one of measuring **phase differences** relative to that reference. The project lead illustrated the concept with a water ripple analogy: imagine all receivers "feel" ripples from a known source (the sync) and then a ripple from an unknown source; by noting the phase of the known ripple when the unknown ripple arrives at each receiver, one can back-calculate the unknown source's location [12]. In signal processing terms, this could mean that each node measures the phase of a continuous wave (from the reference transmitter) at the exact moment it detects the unknown signal's arrival. The **phase difference** between nodes for the reference at that moment is essentially the timing offset we seek for the unknown signal. This approach requires that the reference signal is synchronous (or at least its phase is predictable and common at all nodes) and that the switching from reference to target happens faster than the drift of local oscillators. The custom code in Radio-Mapper is likely responsible for **coordinating this fast tuning and sample alignment** [3]. Notably, an example of this approach is found in an earlier experiment that used a modified RTL-SDR driver to rapidly hop between a GPS-derived reference signal and the signal of interest, achieving successful localization with three receivers [3].

Visualization algorithms: After computing a transmitter's coordinates, the system may also compute an *uncertainty region* (for instance, an error ellipse or circle) based on the timing uncertainty. Visualizing these on the map (perhaps as translucent circles or heatmap gradients) would be useful to indicate confidence. Another possible visualization is drawing the TDoA hyperbolas themselves; for example, the AEDA platform (a similar crowd-sourced SDR project in 2025) displays hyperbolic lines from TDoA calculations to illustrate how the intersection identifies the sourc[14]. An example from AEDA is shown below, where three hyperbola curves (each corresponding to a pair of receivers) intersect near the transmitter's true location:

Example of TDoA hyperbolas from three distributed receivers converging at the transmitter's location. Each curve represents points of equal time-difference between a pair of SDR nodes (image courtesy of AEDA platform).

Beyond TDoA, Radio-Mapper also considers general **spectrum analysis and signal classification** as part of its feature set. The distributed nodes can act as a city-wide spectrum sensor network. This means they could perform tasks like scanning various frequencies and measuring signal power or identifying modulation types at each location. In fact, a modern implementation (AEDA) allows users to initiate spectrum scans on selected nodes and even automatically classifies modulation (FM, DMR, P25 e[15].) Radio-Mapper's inclusion of a "Signal Identification Guide" (sigidwiki) in its resources [16] hints that identifying what a signal is (not just where it is) is an intended feature. The IQ recordings collected could be post-processed with known algorithms or machine learning models to classify signals by type. Over time, the system could build a **database of signals** observed in the area – cataloging emitters by frequency, type, and location. This would transform raw data into actionable information (for example, flagging an unknown digital transmission in a band where only analog FM should be, which could indicate unauthorized usage).

In summary, the algorithms powering Radio-Mapper involve a synergy of **classical signal processing** (filtering, cross-correlation, multilateration) and potentially modern techniques (machine learning for signal recognition, statistical methods for error correction). The project is rooted in well-known theory but is pushing the envelope by applying it with commodity hardware on a large scale. It stands on the shoulders of prior work – from academic theses demonstrating RTL-SDR multilateration [5] to open-source platforms

like KiwiSDR which use GPS-synchronized receivers for HF direction finding [4] – adapting these ideas to a more accessible and scalable form.

Technical Implementation and System Architecture

The Radio-Mapper repository reveals the **technical design choices** in terms of programming languages, libraries, and system architecture. The implementation heavily leverages existing SDR driver code and utilities, combined with custom coordination logic:

- **RTL-SDR Driver Integration (C code):** A significant portion of the codebase is a direct import of the *rtl-sdr* library (from the Osmocom project) into the Radio-Mapper repository [17]. In fact, one of the first commits added ~109,000 lines of C code and build scripts from librtlsdr, described as "code for real time sync/signal" support [17]. This includes source files like `tuner_r82xx.c`, `rtl_test.c`, and many others[18] – confirming that the standard RTL-SDR driver and tools are present. By incorporating the driver's source, the project can modify it to suit its needs (for example, to implement the fast frequency hopping or precise timestamping required for synchronization). The included CMake build files explicitly identify the project as "rtl-sdr" [19] and carry Osmocom's GPL license headers, indicating Radio-Mapper is built atop this open-source foundation. Using C at the low level is essential for performance: it allows direct interaction with the SDR hardware via libusb, precise control of threading and timing, and efficient handling of large IQ buffers in real-time.

- **Build System (CMake, Autotools):** The repository includes both CMake configuration and Autotools files (e.g. `CMakeLists.txt`, `Makefile.am`, `configure.ac`), mirroring the structure of the upstream RTL-SDR project [20] [21]. This suggests that the project can be compiled on Linux systems using the typical `cmake` or `./configure && make` workflow. The presence of `Doxyfile.in` and Roff (`.1` manual) files means documentation for command-line utilities (like `rtl_sdr.1` man page) is also included. The heavy use of CMake/Makefile and some M4 (macro) scripts accounts for about ~13% of the codebase [22] and is primarily for building the software and generating docs.

- **Shell Scripts and Orchestration:** About 18.7% of the code is Shell script [22], which likely includes installation or running scripts. These could be things like `rtl_power` plotting scripts, or custom shell scripts to launch the capture process on each Raspberry Pi at boot. For example, a script might start the SDR capture referencing a central server or schedule periodic sync/calibration operations. Shell scripting makes it easy to glue together system tasks: configuring the Pi's network, launching the SDR sampling program, and possibly calling OS utilities (like `ntpdate` or GPS device reads if those were used as a fallback). The exact scripts aren't listed in the README, but given the percentage, the repository may contain a `Code` subfolder with scripts to manage the cluster.

- **Programming Languages Used:** According to GitHub's analysis, the majority of the repository's content is **Roff (51.6%)**, which represents documentation (likely the included man pages and possibly some troff/nroff formatted text) [22]. This is followed by **Shell (18.7%)**, **C (16.7%)**, **Makefile (9.8%)**, **CMake (3.1%)**, and a tiny bit of **M**[22]. Notably absent is Python or other high-level languages – at least in this initial codebase. It appears the strategy was to do as much as possible in C for the SDR handling, and rely on simple scripts for coordination. That said, future development could introduce Python or Node.js for server-side processing or the web interface. The current repo, however, is focused on the data acquisition side.

- **System Architecture:** The envisioned architecture is a classic **distributed sensor network** with a centralized brain:

- **Capture Nodes:** Each Raspberry Pi runs the Radio-Mapper capture program (likely a modified version of the `rtl_sdr` tool or a custom C application using librtlsdr). These nodes might run a lightweight Linux (Raspbian) possibly stripped down for real-time performance as one commenter suggested (e.g. using a real-time kernel for more consistent timing [7]). They are outfitted with an SDR dongle and optionally a stable clock source (some might use a GPS module for 1 PPS timing, though the project tries to avoid reliance on GPS [6]). The nodes perform the low-level tasks of tuning, sampling IQ data, and possibly detecting when a signal of interest is present. Each node knows its own location (likely pre-configured coordinates). Networking-wise, the nodes need to communicate with the server – this could be via Wi-Fi or wired Ethernet. The project might use simple protocols (even something as straightforward as uploading files over SCP, or sending UDP packets of timestamps). Given the experimental state, a pragmatic approach could be each node writing IQ captures to a file, then an rsync or HTTP post to the server when done.
- **Central Server:** The server side is not fully fleshed out in code yet, but conceptually it's a cloud or central PC that aggregates data from all nodes. It would run the correlation and multilateration algorithms described earlier. It might be implemented as a Python service or a set of C++ programs using scientific libraries – since the repo doesn't show those yet, this part was likely under development or intended future work. The server would maintain a database of known transmitter results, handle new requests (e.g. if a user on the website asks "locate signal at 433.920 MHz"), instruct nodes to perform measurements, then process the incoming data. There is an analogy here to the KiwiSDR TDoA service: KiwiSDR receivers upload snippets of raw IQ to a central server which then runs cross-correlation and returns coordinates. Radio-Mapper could follow a similar architecture, perhaps leveraging existing code from Kiwi's open-source TDoA extension (which is written in Go for the server, by Kiwi's author). However, nothing in the repository explicitly mentions Kiwi integration, so Radio-Mapper's server might have been planned from scratch.

- **Web Interface:** The public website (radio-mapper.org) would be the user-facing component. While no front-end code is in the repository yet, we know from the README that the idea is to present a Google Maps (or similar) view of the city with radio sources plotted [2]. This implies using web technologies (HTML/JavaScript) and possibly a maps API. The site would likely query the server for the latest detected transmitters and their coordinates. It could also allow users to input frequencies or select from a list of active signals. The backend for the site could be a simple REST API that queries the database of results. Security and performance would be concerns if it became widely used (to avoid too frequent scanning or misuse), but at the scale of a community project, it might simply show whatever data the system has collected in the last day or so.

- **Libraries and Tools:** In addition to the core RTL-SDR C library, Radio-Mapper might incorporate other libraries as development continues. For instance, GNU Radio is a powerful DSP environment that could be used for more advanced processing or visualization (one of the references is about GNU Radio dataset generation [23]). However, integrating GNU Radio on every node (especially on Raspberry Pis) could be heavy, so the project may stick to custom lightweight code. For geolocation math, they might use a library like Boost for linear algebra or even a GIS library for coordinate transforms (since working with lat/long directly in algorithms can be tricky; converting to a local Cartesian coordinate grid is often done for calculation then converted back to GPS coordinates). The website might use standard JavaScript libraries and possibly WebGL or leaflet for the maps. Again, these parts are speculative as the repo doesn't show them yet.

To summarize the technical stack, below is a brief breakdown of key components and their roles:

Component	Role in System	Implementation Details
RTL-SDR Library	Low-level SDR control and sampling	Imported Osmocom *rtl-sdr* code in C for tuning, IQ reading, and device handling [24]. Modified for fast frequency switching and timing. Compiled with CMake/Make.
Capture Node Software	Runs on each Raspberry Pi to orchestrate captures	Likely a combination of C programs (for SDR) and shell scripts (for scheduling and network transfer). Possibly uses a reference-tuning routine as per Stefan Scholl's method [3].
Synchronization Mechanism	Ensures all nodes share time base	Utilizes an external RF signal (e.g. broadcast TV/cell) for sync. The custom code toggles between reference and target frequency in real-time. Could use GPIO or PPS from GPS as backup.
Central Processing Server	Aggregates data and computes locations	Not fully implemented in codebase yet; planned to run correlation and multilateration algorithms (could be Python, C++ or Go). Would interface with database or in-memory data stores for results.
Web Front-End (radio-mapper.org)	User interface for visualization and interaction	Planned use of Google Maps or similar to display transmitter locations [2]. Would involve a web server (possibly Python Flask or Node.js) serving an interactive map, querying the latest results from the server.

This architecture is inherently scalable: more nodes can be added to improve coverage and accuracy, and the server can be scaled up to handle more concurrent processing if needed. It is also flexible in that it can be repurposed for different frequency bands or adapted to various scenarios (the software could be updated to target, say, an ISM band for tracking devices, or HF bands for ionospheric research, with minimal changes to the core logic).

One should note that working at the limits of cheap hardware requires a lot of tuning and possibly platform-specific adjustments. For example, the Raspberry Pi's USB bus and scheduling might introduce unpredictable latencies; part of the technical implementation would be measuring and compensating those (perhaps why an RTOS or real-time patches were considered by the community [7]). The **Documents** folder in the repository contains several PDF research papers [25] which likely guided the implementation details (including one on detecting unintended emitters, and a thesis by Krüger on RTL-SDR positioning). By embedding these references, the developer ensured that the implementation aligns with proven techniques and addresses known pitfalls as documented in academic and hacker literature.

In conclusion, the technical implementation of Radio-Mapper is a blend of existing SDR tooling and custom glue logic. It prefers efficiency and low-level control (C and shell) to handle real-time signal capture, while leaving higher-level processing and user interaction to be built on top. This foundation provides a robust starting point for experimentation in distributed radio monitoring.

Research Motivation and Background

Radio-Mapper is driven by a strong research motivation: it tackles an open problem in the field of wireless communications and sensing – **how to accurately locate radio transmitters using inexpensive, off-the-shelf hardware and distributed computing**. There are several theoretical and historical underpinnings for this work:

- **Time Difference of Arrival Theory:** The concept of using TDoA for localization is well-known in engineering (also used in GPS, where the receiver's position is found by time differences from satellites, albeit inverted roles). In terrestrial radio, multilateration has been used by military and regulatory bodies for decades to find signal sources. The theory provides that with at least three receivers and sufficient timing precision, one can solve for a transmitter location [5]. This project leans on that theory but within a very constrained budget and hardware capability, which makes it a research challenge to achieve good accuracy. An academic precedent is the 2017 thesis by Krüger, which successfully demonstrated locating transmitters with unmodified RTL-SDRs by using **transmitters of known location as timing references** [13]. That work presented the theoretical background and practical measurement process, including methods to correct errors, effectively proving that *yes, it is possible to do TDoA with cheap dongles*. Radio-Mapper builds directly on such findings, taking them from an isolated experiment to a networked, scalable system.

- **Previous Experiments (Stefan Scholl's TDoA demo):**The project was initially inspired by a presentation from Stefan Scholl titled "Experiments on Transmitter Localization with TDOA" [26]. In that demo, Scholl used a modified RTL-SDR driver (the `async_rearrangements` branch of librtlsdr) to switch between a synchronization source and the target signal, and reportedly managed to geolocate a transmitter with just three Raspberry Pi + RTL-SDR units [3]. The success of that experiment (achieving around 100 m accuracy) validated the approach and motivated Radio-Mapper's developer to pursue a larger-scale version [10]. It essentially provided a blueprint: use one frequency as a sync, another as the signal, rapidly alternate, and apply math – a blueprint Radio-Mapper adopts and aims to improve upon (e.g., increasing the number of receivers beyond three to potentially get better accuracy or handle signal dropouts).

- **KiwiSDR and Open Source TDoA:** In the late 2010s, the KiwiSDR (a popular HF networked SDR) community implemented a global TDoA service. KiwiSDRs are equipped with GPS-disciplined oscillators, so each unit has a very accurate timestamp. The Kiwi system allows anyone to select a few receivers on a map and run a TDoA calculation for an HF signal, resulting in a location overlay. Notably, the algorithm for KiwiSDR's TDoA was developed by Christoph Mayer (DL1CH) and presented in conferences [27]. Radio-Mapper's approach differs in timing method (they avoid requiring GPS at each node), but the existence of KiwiSDR's system proves that a network of receivers can be harnessed by the public for crowdsourced geolocation [4]. It provided inspiration and possibly code ideas (for example, strategies to pick receivers with good geometry, or how to integrate results into a map UI [28]). KiwiSDR primarily focuses on 0–30 MHz (HF), whereas Radio-Mapper is targeting VHF/UHF (since RTL-SDRs work from ~24 MHz up to ~1.7 GHz). This means Radio-Mapper addresses a different segment of the spectrum – one filled with local communications (police, fire, airband, broadcast FM/TV, etc.) where geolocation can have immediate practical uses.

- **Academic and Scientific Impetus:** Beyond finding "rogue transmitters," the data collected by Radio-Mapper has scientific value. For example, mapping signal strengths across a city can feed studies on

propagation modeling (urban RF propagation is complex due to buildings, and having real data points aids model tuning). There is also interest in using **machine learning in radio** – one of the references is a GNU Radio Conference paper on dataset generation for Radio Machine Learning [23]. A distributed network like Radio-Mapper could generate rich datasets: recordings of various signals along with their known locations, which could be used to train AI models to, say, infer location from signal characteristics, or to classify signals. This ties into the emerging field of **RF machine learning**, where having labeled data (location is a label too) is often a bottleneck. The project explicitly references a "Radio Machine Learning Dataset Generation" document, indicating an intent to contribute to or leverage that area.

- **Spectrum Awareness and Security:** Another motivation is improving spectrum awareness. Regulatory agencies (like the FCC in the US) have to monitor spectrum use to catch illegal transmissions or resolve interference. Doing this with a handful of expensive monitoring stations is hard – but a dense network of cheap monitors could make it far easier. The reference "Detecting/ Locating Unintended Radio Transmissions" [29] suggests applications in security, where electronic devices might emit unintentional RF signals that could be detected and located (for example, a hidden camera or a keystroke logger might emit RF; being able to locate such devices is valuable). Radio-Mapper's approach could aid in **electromagnetic security sweeps** by quickly mapping out all transmissions and identifying anomalous ones. This is indeed an open challenge in the security community: finding unwanted emitters in sensitive locations. The project's inclusion of that reference shows a recognition of this use case.

- **Collaboration and Crowdsourcing:** The scientific community benefits greatly from open, shared data. Radio-Mapper is inherently a collaborative project – by making the data public on a website, it allows researchers, hobbyists, and students to use real-world RF data. This democratization of signal intelligence (SIGINT) is quite novel. It parallels how projects like ADS-B Exchange crowdsource aircraft tracking data from many ADS-B receivers. Here, it would be crowd-sourced spectrum data. The *AEDA* platform mentioned earlier is a recent development (2025) that strongly echoes Radio-Mapper's concept: a crowd-sourced RF detection and TDoA platform using RTL-SDRs [30]. AEDA is even looking to monetize and pay contributors, showing that there is economic and research interest in this idea [31]. Radio-Mapper, being open-source and community-driven, is more in the spirit of open research – it could enable experiments in distributed computing, sensor fusion, and even social aspects of who contributes nodes.

- **Open Problems Addressed:** A key research question Radio-Mapper addresses is **how to achieve reliable synchronization without expensive hardware**. By experimenting with reference signals and software timing alignment, it contributes knowledge on what's feasible with purely software approaches. Another problem is **scalability** – as the number of nodes increases, how to efficiently coordinate them? This might lead into research on distributed scheduling (which nodes should monitor which frequencies at what times to cover most ground?) and data reduction (sending raw IQ from dozens of nodes isn't bandwidth-friendly, so what intelligent preprocessing can be done at the edge?). These are current problems in IoT and sensor networks, and Radio-Mapper can be seen as an IoT of RF sensors. The insights gained could apply to other domains as well (like acoustic arrays or seismic sensor networks, which have analogous localization problems).

In summary, Radio-Mapper stands at the intersection of **theory and practice**: it takes theoretical concepts of multilateration and signal processing, combines them with practical engineering in SDRs, and aims to

produce a system that not only demonstrates these ideas but also provides useful services. The project is clearly informed by prior academic work (as evidenced by the documents and references in its repository) and seeks to advance the state of the art by removing cost barriers and encouraging wide participation. It addresses open problems of synchronization, distributed detection, and real-time data sharing, contributing to both the scientific community and the radio enthusiast community. By doing so, it can validate theories in a real-world environment and potentially inspire follow-up research (for instance, papers analyzing the data it collects, or improvements in algorithms derived from observing its performance).

Applications and Potential Use Cases

If successful, Radio-Mapper unlocks numerous applications across spectrum monitoring, communications, and security domains. Some prominent use cases include:

- **Spectrum Sensing and Frequency Allocation:** A distributed map of radio signals helps regulators and researchers understand how spectrum is being used in a region. For example, **band occupancy** measurements could be made by having nodes scan and report which frequencies are active and where. This can reveal unused spectrum (white spaces) or detect spectrum hoarding. It is valuable for spectrum management and planning – e.g., deciding where there is room for new services or whether certain frequency bands are underutilized. Researchers conducting **band studies** (frequency surveys) could utilize the network to get data without deploying their own sensors [31].

- **Interference Detection and Mitigation:** One of the most immediate uses is finding the source of interference. Whether it's an illegal jammer, a malfunctioning transmitter, or even an incidental radiator (like a leaky cable TV amplifier causing noise in the airwaves), Radio-Mapper can localize it. For instance, if an airport's control tower reports intermittent interference on their radios, a set of Radio-Mapper nodes around the area could quickly triangulate the source – possibly faster and cheaper than sending out a van with a directional antenna. This is directly mentioned as a goal in similar platforms; AEDA specifically lists **radiolocation and interference detection** as target applications for their rented node network [31]. In an urban environment, there are many sources of interference (from pirate FM stations to wireless microphone users on illegal frequencies), and an automated system to pinpoint them is extremely useful for communications authorities.

- **Collaborative Signal Intelligence (SIGINT):** In the intelligence and security arena, multiple sensors working together can capture different aspects of a signal. For example, one node might get a clearer view of a signal's content (if closer), while another provides location. By combining these, one can both **decode and locate** a transmission. Applications include monitoring public safety or military exercises (by hobbyists or researchers), tracking illegal broadcasts, or even academic projects like mapping out all the cell tower signals in an area and identifying their operators. "Collaborative signal intelligence" also implies community involvement – hobbyist radio listeners could use the network to, say, track down the position of a mysterious transmission they heard. Instead of each person doing a manual hunt, they all benefit from the network's result. This democratizes capabilities that were once limited to large organizations.

- **Emergency Services and Search & Rescue:** Many emergency beacons (ELTs from downed aircraft, personal locator beacons, etc.) operate via radio. Locating them is crucial in rescue operations. Typically, special receivers (direction finders) are used by search teams. With a permanent network

like Radio-Mapper, an active distress beacon could be automatically detected and located if within range of the city-wide network – potentially saving critical time. Even without dedicated distress signals, if someone in trouble used a two-way radio (e.g., calling for help on a common channel), the system could flag and locate it. On the flip side, unauthorized or hoax signals on emergency channels could be identified and dealt with.

- **Tracking Moving Transmitters:** Beyond static maps, the system can track movement. For example, if someone is driving around with a transmitter (intentional or unintentional), the network could follow their path. This has law enforcement applications (tracking a suspect with a bugged radio or a stolen police radio) as well as wildlife or asset tracking in research. Notably, an article on RTL-SDR.com described using TDoA to track wild bats fitted with tiny radio tags [32] – a wildlife telemetry system called ATLAS uses TDoA to locate animals. A Radio-Mapper network could contribute to such scientific efforts in a city or region by listening for tag frequencies and providing continuous localization updates. The ability to do this with low-cost nodes makes it feasible to cover large areas (for environmental studies or even tracking low-power IoT devices moving through a city).

- **Heatmaps and Coverage Mapping:** By measuring signal strength (RSSI) at multiple nodes, one can create coverage maps for transmitters. For example, for each FM broadcast station, the system could log the received power level at all nodes and interpolate a coverage map, highlighting weak and strong areas. This is akin to the reference example of creating a VHF signal heatmap with RTL-SDRs [33]. Such maps are useful for broadcasters (to see if their coverage claims match reality) and for identifying RF dead zones. It can also help in network planning – telecom companies could use a crowd-sourced system to verify their cell tower coverage. Although Radio-Mapper primarily focuses on pinpointing transmitters, collecting power data is a complementary function easily done by the same hardware.

- **Public Transparency and Education:** An often overlooked application is simply making the invisible radio world visible to the public. By providing a live map of RF activity, Radio-Mapper can educate people about the devices and signals around them. It can spark interest in radio science and engineering. For the tech community, it's a sandbox for experimenting: students could use the network to test their own localization algorithms or to study propagation by time of day (e.g., see how far a 2 m band signal gets at night vs day by checking which nodes hear it). The data could be made open-access for academic research, enabling papers on topics from urban propagation to dynamic spectrum access (where networks decide frequencies on the fly based on usage maps). Additionally, transparency can deter misuse of the spectrum – if everyone knows there's a public map showing who's transmitting where, rogue operators might think twice.

- **Commercial Services:** While Radio-Mapper itself is an open project, it hints at potential commercial services. As mentioned, the AEDA platform is looking to monetize a similar concept by renting out the sensors for tasks like **band monitoring or locating interference for companies** [31]. One can imagine a telecommunications company paying for detailed spectrum surveys before deploying a new network, or a city government using it to detect illegal broadcasts (e.g., unauthorized taxi dispatch radios). The hardware cost is so low that a city could blanket itself in these sensors (much like CCTV cameras) for continuous RF monitoring. Privacy concerns aside, it could become part of smart city infrastructure (detecting drone control signals, for instance, as part of airspace security). Radio-Mapper demonstrates the feasibility of this concept, potentially leading to startups or municipal projects adopting the idea.

In light of the above applications, Radio-Mapper can be seen as an enabling technology – enabling better use of spectrum, faster response to interference, and community engagement in the airwaves. It aligns with trends in both citizen science and the increasing need for robust spectrum management tools. The fact that a near-identical idea (AEDA) is emerging commercially in 2025 underscores the timeliness and relevance of Radio-Mapper's goals [30] [31]. Each use case reinforces the value of having a **distributed ear to the ground** in the RF domain.

Conclusion

In conclusion, the **physiii/radio-mapper** project is an ambitious blend of research and engineering that targets a cutting-edge problem in radio technology: mapping the RF environment in real time using a distributed network of cheap sensors. Its objectives are clear – to localize radio emitters and present that information in an intuitive map form – and it approaches them with a thoughtful combination of known techniques (TDoA multilateration, reference signal synchronization) and practical engineering (leveraging Raspberry Pis and RTL-SDR dongles). The data pipeline involves innovative synchronization via ambient signals, meticulous signal processing to extract time differences, and scalable architecture for data aggregation. The technical implementation shows a reliance on proven open-source components (like the RTL-SDR driver in C) while also carving out space for custom development tailored to this application.

From a research perspective, Radio-Mapper stands on solid theoretical ground and pushes into new territory by removing the cost barrier, essentially asking, "What if anyone could deploy a radio direction-finding network?" The potential applications are wide-ranging, from mundane (but important) tasks like interference hunting and spectrum surveys, to high-impact uses in security, rescue, and scientific research. By structuring the project as open and community-focused, the developers invite collaboration and ensure that its benefits (and lessons learned) propagate to others in the field.

While still in early stages (with some components like the web interface and real-time server processing in development), Radio-Mapper lays the foundation for a **crowdsourced radio intelligence network**. It exemplifies how advances in software-defined radio and time-sync algorithms can transform the way we understand and manage the invisible spectrum around us. In doing so, it not only addresses current needs (like finding that noisy neighbor's illicit transmitter) but also opens the door to future innovations in wireless networking, smart cities, and beyond. The project is a testament to the power of combining academic insight with open-source development: it takes theory out of textbooks and proves it out on the streets (literally, via nodes on street corners), creating a new kind of atlas – a radio map – for the modern world.

Sources:

- physiii, **Radio-Mapper README** – Project description and objectives [34] [35].
- physiii (commenter), *Hackaday*: "Where's That Radio? A Brief History Of Direction Finding" – Author's explanation of Radio-Mapper goals and TDoA approach [1] [11], discussion of synchronization challenges [7] [6] and expected accuracy [10].
- **GitHub Repository Commit History** – Commit message "added librtlsdr code for real time sync/signal" indicating integration of RTL-SDR C code [17]. Repository language breakdown showing C, Shell, Roff content [22]. Excerpts from imported `CMakeLists.txt` confirming Osmocom RTL-SDR usage [24].

- admin, *RTL-SDR.com*: “AEDA: Crowd Sourced RTL-SDR Spectrum Analysis and TDoA Platform” – Describes a similar distributed SDR system, using external signals for sync and listing uses (spectrum analysis, TDoA direction finding, interference detection) [15] [31] .
- admin, *RTL-SDR.com*: “Thesis on Locating Transmitters with TDoA and RTL-SDRs” – Highlights use of known transmitters as timing references and correction of errors on RTL-SDRs [5] [13] .
- admin, *RTL-SDR.com*: “KiwiSDR TDoA Direction Finding Now Freely Available” – Notes the requirement of GPS-synchronized clocks for TDoA and the ability to use public KiwiSDRs for geolocation [4] .
- **Radio-Mapper References** (Documents folder) – included research papers: *“Detecting and locating electronic devices using their unintended emissions”*, *Krüger (2017) Thesis on RTL-SDR positioning*, etc., which informed the project’s design [25] .

---

[1] [3] [6] [7] [8] [10] [11] [12] Where’s That Radio? A Brief History Of Direction Finding | Hackaday
https://hackaday.com/2021/08/19/wheres-that-radio-a-brief-history-of-direction-finding/

[2] [16] [23] [26] [29] [33] [34] [35] raw.githubusercontent.com
https://raw.githubusercontent.com/physiii/radio-mapper/master/README.md

[4] [5] [9] [13] [14] [15] [27] [28] [30] [31] [32] time difference of arrival
https://www.rtl-sdr.com/tag/time-difference-of-arrival/

[17] Commits · physiii/radio-mapper · GitHub
https://github.com/physiii/radio-mapper/commits/master/

[18] [19] [20] [21] [24] [25] added librtlsdr code for real time sync/signal and Documents folder · physiii/radio-mapper@d20cb6b · GitHub
https://github.com/physiii/radio-mapper/commit/d20cb6b1c674c5255fdeb451eaeceacfd51439c9

[22] GitHub - physiii/radio-mapper: Map stationary and moving radio signals to longitude, latitude points with SDR
https://github.com/physiii/radio-mapper