FISSURE Technical Details and Architecture

Christopher Poore
Assured Information Security, Inc.
153 Brooks Road
Rome, NY 13441
315-336-3306 x1569
poorec@ainfosec.com

1. INTRODUCTION

1.1 Purpose and Audience

This document provides a technical overview of the FISSURE framework. Its purpose is to explain what the system is made of, how the main components interact, and what the framework can do today and in the future. It is not a step-by-step manual, but rather a foundation for technical discussion and evaluation.

The intended audience is broad. DoD evaluators and acquisition staff can use this paper to understand how FISSURE is structured, how it scales, and what advantages it offers compared to traditional systems. Special Operations and other operational users will see how the framework translates into practical workflows for spectrum monitoring, analysis, and effects. Engineers and researchers will find details on software design, modularity, data flow, and integration points. Students, hobbyists, and the open-source community will find an accessible entry point, since FISSURE favors COTS hardware, Python, and a plugin approach that makes it possible to experiment and contribute without specialized equipment.

The goal is to keep this paper dual-use in nature. It is written so that new users can follow the main ideas while technical experts can still evaluate the design choices and architecture in depth. It should answer the central question for every reader: what is FISSURE, and how does it actually work?

1.2 Why FISSURE Matters

Modern RF, EW, and cyber tools are often fragmented, costly, or locked into proprietary ecosystems. Many require specialized hardware or vendor support, which limits accessibility and slows innovation. Open-source options exist, but they are usually standalone projects that demand significant expertise to assemble into a working environment. The result is a gap between what operators, researchers, and students need and what they can realistically use.

FISSURE addresses this gap by providing a single framework that combines RF reverse engineering, spectrum monitoring, protocol discovery, cyber experimentation, and attack execution. It is open-source, plugin-driven, and built around commercial off-the-shelf hardware. Users can begin with a laptop and a basic software-defined radio, then scale up to fleets of ruggedized tactical nodes managed from a central hub.

For operators, this means mission-ready capabilities that adapt to different radios, networks, and deployment scenarios. For researchers and educators, it provides an environment where experiments can be reproduced, lessons can be shared, and contributions can flow back into the community. FISSURE matters because it makes advanced spectrum and cyber operations both accessible and extensible, bridging the divide between hobbyists, classrooms, labs, and operational units.

1.3 Position in the Ecosystem

FISSURE sits at the intersection of radio frequency experimentation, electronic warfare, and cyber operations. It is not a single-purpose tool or proprietary box, but a framework that unifies signal detection, analysis, classification, protocol discovery, and effects in one environment.

Other FISSURE white papers explore how the framework applies to specific domains such as counter-UAS, maritime security, mobile operations, or ATAK integration. This paper is different: it provides the foundation by showing what the system is built from, how it looks and feels to use, and the architecture that supports those domain-specific applications.

FISSURE is also a living project. The details here reflect its current state, but capabilities continue to evolve as new radios, algorithms, and operational needs emerge. This document is intended to remain a reference point for newcomers and experts alike, offering a clear picture of how FISSURE fits into the broader ecosystem of spectrum and cyber tools.

1.4 Scope

This paper defines the technical structure and operation of the FISSURE framework. It describes the core components that make up the system, including the central hub, tactical nodes, plugins, processing engines, the database, and the user interface. It explains how these pieces interact, covering both manual workflows and automated pipelines for data and signal flow.

The scope also includes the user experience: what it feels like to operate FISSURE from the perspective of an analyst, researcher, or operator. It outlines what can be done today using the existing tabs, tools, and workflows, and it sets expectations for what is planned in the near future, such as automation, AI/ML integration, and advanced coordination across nodes and hubs.

This document is not a user guide. Instead, it provides the architectural context needed to understand how FISSURE is designed, why it works the way it does, and how it can grow. Readers will also find deployment scenarios and use cases that show how the framework can be applied across operational, educational, research, and hobbyist domains.

The paper is intended as a starting point for conversation and evaluation. It is comprehensive enough for decision-makers and technical experts to assess FISSURE's architecture, yet accessible for newcomers who want to see where the framework fits into the larger spectrum and cyber ecosystem.

1.5 How to Read This Paper

FISSURE is a flexible framework, and readers will approach it with different goals. This paper is organized so each audience can focus on the sections most relevant to their needs.

- **Engineers and developers** should focus on *System Architecture*, *Core Components*, and *Theory of Operation*, which explain the technical design, data flows, and extension points.
- **Operators and decision-makers** may want to begin with *Current Capabilities* and *Example Use Cases*, which show how FISSURE applies in real-world scenarios.
- **Students, hobbyists, and researchers** will find the most value in *Design Philosophy*, *Integration Paths*, and *Future Directions*, which highlight learning, experimentation, and extension.

The sections are written to stand on their own, so readers can jump directly to what matters most. This is not a manual but a reference and conversation starter, meant to show what FISSURE is, how it is structured, and how it can be applied or extended.

1.6 Terminology and Conventions

This paper uses several recurring terms and shorthand. They are defined here for clarity so that readers from different backgrounds can follow the technical discussion consistently.

- **Hub / Central Hub / HIPRFISR:** The orchestration point of the framework. It manages nodes, routes messages, coordinates tasks, and provides the anchor for plugins and processing engines.
- **Tactical Node / Sensor Node:** A deployed computer with radios or sensors attached that executes a subset of code. Nodes run plugins and flow graphs, capture and transmit signals, and can operate remotely or autonomously.
- **Processing Engine:** A specialized service for heavy compute tasks. Today, there are two: the Target Signal Identification (TSI) engine and the Protocol Discovery (PD) engine.
- **Dashboard:** The main desktop user interface built with PyQt. It provides tabs for detection, conditioning, classification, protocol discovery, IQ data handling, and attack execution.
- **Client App:** Any user-facing interface other than the main dashboard, such as lightweight mobile apps or TAK integrations.
- **TAK (Team Awareness Kit):** The TAK ecosystem used in field workflows; includes ATAK (Android), WinTAK (Windows), and WebTAK (web). In this paper, "TAK" refers to these clients and their plugins that exchange alerts, tasks, and status with the hub.
- **Plugin:** A modular package containing flow graphs, scripts, or tools. Plugins define tasks such as detection, analysis, attack execution, or visualization, and can be distributed from the hub to nodes.
- **Flow Graph:** A GNU Radio-based signal processing chain. Flow graphs are used inside plugins, detectors, demodulators, and fuzzers.
- **Playlist:** A sequence of single-stage or multi-stage attacks, IQ replays, or actions that can be run manually or set to autorun.
- **Detector:** A signal search component that identifies signals of interest based on thresholds, bands, or fixed frequencies.
- **SOI (Signal of Interest):** A signal identified by detectors or analysis steps as worth further investigation. SOIs can be tagged, saved, and added to the library.
- **Library:** The PostgreSQL database that stores metadata about signals, protocols, plugins, attacks, playlists, and models, along with references to IQ files and artifacts.
- **Meshtastic:** A low-throughput networking option using Meshtastic radios and the Meshtastic Python library. Provides long-range links where IP connectivity is limited.
- **ZeroMQ (ZMQ):** The primary networking layer used by FISSURE for hub-to-node communication on IP networks.
- **EW (Electronic Warfare):** Actions in the electromagnetic spectrum to sense, protect, and affect. In this paper, EW is an umbrella for detect/classify/direction finding (DF) /geolocate (Electronic Support), hardening and spectrum management (Electronic Protection), and authorized effects such as jamming/spoofing/replay (Electronic Attack). FISSURE today focuses primarily on ES, with modular paths toward EP/EA via plugins and coordinated nodes.

2. PHILOSOPHY AND CORE PRINCIPLES

FISSURE is built to adapt. Its architecture combines the accessibility of commercial off-the-shelf hardware with a plugin-driven software model, allowing new capabilities to be added without disrupting the core. By unifying RF and cyber operations in one environment, FISSURE supports operators in the field, researchers in the lab, and students in the classroom.

This section highlights the principles that guide FISSURE's design: openness and extensibility, a COTS-first hardware philosophy, modular software architecture, the convergence of RF and cyber domains, and a focus on accessibility and community.

2.1 Open and Extensible Framework

FISSURE is designed to grow and adapt. New capabilities can be added as plugins, lessons, or supporting tools without requiring changes to the core system. This allows users to move quickly from basic experiments to advanced workflows, building on the same foundation.

The framework supports a wide range of tasks, from signal detection and classification to protocol discovery and attack execution. Each task is encapsulated in a modular form so it can be run independently, combined with others, or automated through playlists. Analysts can step through workflows in the Dashboard, while developers extend the system with custom flow graphs, models, or scripts.

Openness also means that FISSURE invites contributions from many types of users. Researchers can add new algorithms, students can build lessons, and operators can package workflows for repeatable field use. This model allows the system to evolve quickly while remaining consistent and maintainable, making it useful to both the open-source community and operational environments.

2.2 Hardware Philosophy

FISSURE is built with a philosophy of using commercial off-the-shelf hardware wherever possible. The framework runs well on widely available software-defined radios, common computing platforms, and low-cost peripherals. This keeps the system accessible to students, researchers, and hobbyists, while reducing cost and deployment time for operational users.

A tactical node can be assembled from a laptop, desktop, single-board computer, or ruggedized mini-PC. Radios range from entry-level devices to high-performance systems, as long as drivers and operating system support are available. GPS receivers and other adjunct sensors can be added in the same way. This flexibility allows the same framework to scale from a classroom exercise to a field deployment.

Although commercial hardware is the preferred path, FISSURE is fully compatible with government off-the-shelf equipment and specialized systems. If a sensor kit or secure computer supports the required environment, it can be integrated seamlessly. This balance ensures FISSURE meets mission requirements while keeping its hardware model broadly accessible.

2.3 Software Modularity and Plugins

FISSURE's software architecture is built around modularity. The core is written in Python with PyQt providing the Dashboard interface, while the central hub (HIPRFISR) manages tasking, messaging, and coordination. Communication is typically handled through ZMQ for IP-based networking, with alternatives such as Meshtastic available for low-throughput or long-range links.

Capabilities are packaged into plugins, allowing the system to be extended without altering the core. A plugin can contain a Python script, a GNU Radio flow graph, or both, along with supporting utilities. Plugins span detection, classification, protocol discovery, attack execution, IQ analysis, fuzzing, and packet crafting. They can be simple single-stage actions or complex chained workflows, giving users flexibility in how they operate.

Because plugins can run locally or be distributed across the network, tasks are executed where they make the most sense. Lightweight detectors may run directly on a tactical node, while more demanding classifiers or protocol discovery routines may be routed to a processing engine. This makes the framework scalable and adaptable across varied environments.

Plugins can be managed according to sensitivity. Some may be published openly in the community, while others are maintained in private repositories for operational use. This separation allows the core framework to remain open-source while supporting restricted or mission-specific requirements.

This structure balances flexibility with consistency. Engineers can add algorithms, operators can package workflows, and students can design lessons without disrupting the core. The plugin system allows the framework to evolve continuously while keeping the foundation stable.

2.4 RF and Cyber Duality

FISSURE is built on the understanding that spectrum operations and cyber operations are no longer separate domains. Signals that begin in the air often end as network traffic, and networked systems are increasingly controlled through wireless links. By unifying these worlds, FISSURE allows users to move seamlessly from monitoring and analysis to direct effects.

On the RF side, the framework supports signal detection, conditioning, feature extraction, classification, and recursive demodulation. These functions identify signals of interest and reveal how protocols behave at the physical and link layers. On the cyber side, FISSURE provides fuzzing, packet crafting, injection, and vulnerability assessment. Together, these tools form a workflow that spans initial discovery through exploitation.

This duality serves multiple purposes. Operators gain a mission tool that covers both awareness and action. Researchers can explore how RF and cyber techniques interact. Students can learn about spectrum and networks in a single environment rather than juggling disconnected toolchains. This integration makes experimentation, education, and operations possible in one unified environment instead of a patchwork of systems.

2.5 Consolidation and Accessibility

The RF and cyber tool landscape is often fragmented. Analysts may need one program for capturing signals, another for analyzing IQ data, a separate toolkit for fuzzing or packet crafting, and still more utilities for visualization or reference. FISSURE consolidates these functions into a single environment, so users spend more time experimenting and less time managing tools.

Consolidation is paired with accessibility. The framework provides a graphical installer, built-in lessons, and a tabbed Dashboard that makes complex workflows easier to understand. Visualizations and reference materials are integrated so users do not have to leave the environment for common tasks. Legacy devices are supported alongside the latest software-defined radios, lowering the barrier for those who already own hardware.

For newcomers, this design makes the system approachable. For experienced users, it reduces friction by keeping workflows consistent across different devices and deployments. By combining many scattered functions into one framework, FISSURE lowers the learning curve while still supporting advanced operations.

2.6 Community and Education

FISSURE is more than a technical framework; it is a community project. The design encourages open-source collaboration so researchers, operators, educators, and hobbyists can all contribute and benefit. Contributions may take the form of new plugins, analysis techniques, protocol dissectors, datasets, or lessons. By keeping the framework open and extensible, users at every level have a clear path to add value.

Education is a central focus. FISSURE provides a hands-on environment for learning RF, electronic warfare, and cybersecurity. Students can progress from capturing IQ data to extracting features, training classifiers, and exploring protocol vulnerabilities within the same system. Educators can build guided lessons, classroom exercises, and demonstrations without needing to assemble a patchwork of tools.

This community-driven approach ensures FISSURE grows in both capability and usability. Professionals can advance the state of the art, while students and hobbyists gain access to tools that would otherwise be too fragmented or expensive. The result is a sustainable ecosystem that spans classrooms, labs, and operational environments.

3. SYSTEM ARCHITECTURE OVERVIEW

The FISSURE architecture is designed to be simple enough for new users to understand, yet flexible enough to scale from a single laptop to a distributed network of tactical nodes. At its core, the framework separates user-facing interfaces, orchestration and coordination, and execution of signal or cyber tasks. This structure allows analysts, operators, and researchers to interact with the system consistently, regardless of deployment size.

At the highest level, the system follows a client–hub–node model. Clients include the desktop Dashboard, lightweight mobile apps,

and TAK integrations, all of which allow users to initiate tasks and view results. The central hub (HIPRFISR) serves as the orchestration layer, routing messages, coordinating workflows, and managing system state. Tactical nodes form the execution layer, where radios, sensors, and plugins are deployed to perform detection, collection, and effects.

Supporting these layers are processing engines that handle compute-heavy tasks such as signal classification and protocol discovery, along with a data layer that preserves metadata, models, and artifacts. Together, these components create a flexible architecture that can support everything from classroom use to distributed operational deployments.

This section describes the major components, reference diagrams, control and data flows, and deployment patterns that define the FISSURE architecture.

3.1 Reference Architecture

At a high level, FISSURE follows a client–hub–node model that makes the framework both approachable for single-machine use and scalable for distributed deployments.

- **Clients** include the desktop Dashboard, lightweight mobile applications, and TAK integrations. These interfaces allow users to initiate tasks, configure settings, and visualize results. Each client type operates in its own task and status domain to prevent conflicts, with bridges enabled when integration is required.
- The **Central Hub (HIPRFISR)** is the orchestration layer. It registers nodes and processing engines, routes tasking messages, tracks system health, and manages job lifecycles. By centralizing coordination, it ensures consistency across multiple users and deployments.
- **Tactical Nodes** form the execution layer. Each node consists of a COTS or specialized computer with attached software-defined radios and sensors. Nodes run plugins, flow graphs, and scripts for detection, recording, classification, or attack execution. They can operate under hub control or autonomously when disconnected.
- **Processing Engines** provide dedicated services for compute-intensive pipelines such as target signal identification and protocol discovery. They can run alongside the hub or on separate systems for scalability.
- The **Data Layer** ensures persistence. PostgreSQL maintains metadata for signals, protocols, playlists, plugins, and models, while artifacts such as IQ captures, PCAPs, and trained models are stored and referenced in the library.

This architecture allows FISSURE to be deployed in many patterns: a single laptop for classroom use, a hub with local nodes for a lab environment, or a distributed set of tactical nodes with TAK integration for field operations. The structure provides a consistent framework that adapts to different users and scenarios.

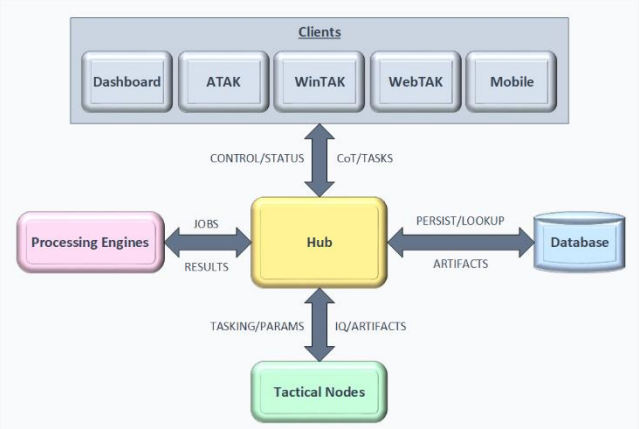

Figure 1. FISSURE system architecture showing the flow of control, tasks, and artifacts between Clients, the Hub, Tactical Nodes, Processing Engines, and the Database.

3.2 Control vs Data Planes and Message Flow

FISSURE routes tasking and large data results on separate paths so the interface stays responsive when large transfers occur.

- **Control Plane:** Clients send requests to the hub, the hub selects a node or engine, and responses return to the originating client. Start and stop actions follow a request and response pattern: the Dashboard packages a request with an identifier and parameters, the hub routes it to the selected target, and status and results return on the same path.
- **Data Plane:** Tasks may produce artifacts such as IQ recordings, decoded frames, PCAPs, features, or model outputs. Artifacts are stored on the node, transferred via the hub, or written into the library, depending on configuration and bandwidth. On low-throughput links, summaries and references are used rather than bulk transfers.
- **Status Flow:** Nodes and engines provide ongoing status and acknowledgements so progress is visible without polling. Details of channels and schemas are covered later.

This separation scales from a single laptop to distributed deployments while keeping tasking responsive. For networking and messaging specifics, see the dedicated sections in Chapter 8.

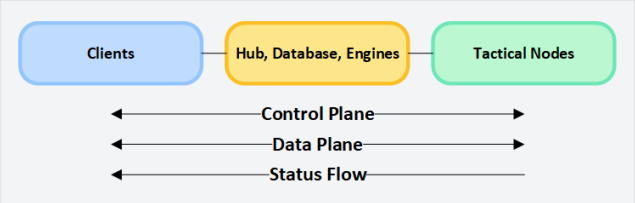

Figure 2. Control, data, and status message paths in FISSURE.

3.3 Deployment Topologies

FISSURE is flexible enough to support a range of deployment patterns, from small lab environments to distributed field operations. The same architecture scales up or down depending on the number of nodes, the role of the hub, and the networking environment. Common topologies include:

- **Single-Laptop Lab:** Everything runs on one machine, including the Dashboard, hub, database, and node software. This setup is common for classrooms, hobbyists, or early-stage development where simplicity matters more than distribution.
- **Hub with Local Nodes:** In a lab or research setting, the hub may run on one system while additional local machines act as tactical nodes. Each node has its own radios or sensors but remains on the same LAN. This enables multiple users to experiment with different roles or devices while sharing the same coordination layer.
- **Distributed Tactical Network:** In field operations, nodes may be mounted on vehicles, deployed as fixed stations, or carried on UAVs. These nodes connect back to the hub over IP or low-throughput links such as Meshtastic. A TAK server can bridge into the hub for integration with ATAK, WinTAK, or WebTAK clients. This topology supports monitoring, analysis, and effects across wide areas.
- **Cloud or Edge Hybrid:** For large-scale or multi-site use, the hub and database can run in a cloud or edge environment while nodes remain at the tactical edge. Processing engines can scale independently to handle classification, protocol discovery, or other heavy tasks. This model supports both remote operations and collaborative research.

These deployment options allow FISSURE to adapt to diverse use cases while keeping the same underlying structure. A setup that begins as a single laptop in a classroom can evolve into a distributed, multi-node network with minimal changes to workflows.

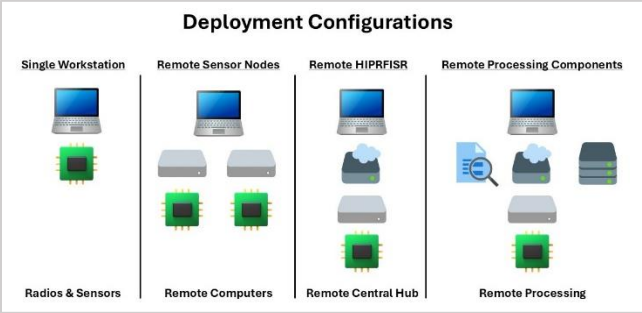

Figure 3. Modular deployment configurations illustrating how FISSURE adapts across single systems, remote sensor nodes, central hubs, and scalable processing components.

3.4 Extensibility and Interfaces

FISSURE is designed to expand as new radios, algorithms, and operational needs emerge. Extension points are well defined so new capabilities can be added without disrupting the core system.

- **Plugins and Scripts:** Most new functionality enters the system as plugins. A plugin can include Python scripts, GNU Radio flow graphs, or supporting utilities, covering detectors, decoders, attacks, analysis routines, or exporters. Because plugins are registered through the hub and tracked in the database, they can be deployed to tactical nodes or processing engines without modifying the rest of the framework.
- **Device Adapters:** FISSURE can interface with any software-defined radio or sensor that has Python, driver, and operating system support. Adapters provide the link between hardware and the framework, enabling users to expand supported devices as new technologies become available.
- **Third-Party Tools:** Integration with external programs expands functionality. Wireshark and Lua dissectors enable packet analysis, Scapy allows custom packet crafting, and visualization tools support quick reference and inspection. These can be launched directly from the Dashboard or used alongside FISSURE outputs.
- **Schemas and APIs:** Communication follows lightweight schemas that support request/response and publish/subscribe messaging. Callback functions are grouped by component, keeping the system organized and easier to extend. Planned work includes automatic API generation from Python functions to simplify integration with other systems.

Extensibility remains a core principle, allowing FISSURE to grow while keeping a consistent structure. Users can add plugins, integrate new sensors, or connect third-party tools without changing the foundation of the framework.

4. CORE COMPONENTS

The FISSURE framework is organized around a set of core components that define how tasks are orchestrated, executed, and recorded. Each has a distinct role, but together they form a cohesive system that scales from a single computer to a distributed network of tactical nodes.

At the center is the hub, which provides orchestration and coordination. Surrounding it are tactical nodes that execute radio-bound tasks, plugins that add new capabilities, and processing engines that handle compute-intensive workflows. A data layer preserves results, models, and metadata, while the Dashboard and client applications give users control and visibility into the system.

The following subsections describe each of these components in detail, focusing on their responsibilities, how they interact with one another, and how they contribute to the overall framework.

4.1 Central Hub (HIPRFISR)

The central hub, known as HIPRFISR, is the orchestration point of the framework. It manages connections between clients, tactical nodes, and processing engines, ensuring that tasks are routed to the right destination and results are returned to the right place. By centralizing coordination, the hub allows FISSURE to scale without losing consistency.

The hub maintains a registry of nodes, plugins, and engines, then uses that registry to select where work should run based on hardware, software, and available resources. It handles task routing and lifecycle management from creation through completion, tracking status, errors, and any returned artifacts as part of a consistent record.

Communication is organized into clear domains, so status and responses remain within the client namespace that originated them. Dashboard, mobile, and TAK applications do not share state unless an explicit bridge is enabled. When the TAK bridge is configured, the hub can publish selected alerts and status updates to a TAK server while keeping the Dashboard and mobile views separate.

Connectivity is monitored through health checks and heartbeats. If a node or engine goes offline, the hub preserves state, pauses coordination as needed, and resumes once the link is restored. It

also coordinates multi-node operations, keeps the plugin registry authoritative, and updates database metadata and references so results are discoverable without moving large files unnecessarily.

Over time, the hub will expand to include modes of operation orchestration and AI/ML decision support, bringing policy enforcement and higher-level coordination into the same place. In practice, the hub provides the backbone of the architecture through its orchestration, coordination, and tracking roles, keeping the framework organized, reliable, and ready to scale from small experiments to complex distributed operations.

4.2 Tactical Nodes

Tactical nodes are the execution layer of FISSURE. Each node is typically built from commercial off-the-shelf hardware, with one or more software-defined radios and optional sensors attached. Nodes can be lightweight portable devices or ruggedized systems with higher performance for demanding field deployments.

A node's primary role is to interact with radios and sensors. This includes tuning, capturing IQ data, storing and replaying signals, and transmitting when authorized. Nodes also execute plugins and scripts, enabling them to run detectors, classifiers, protocol discovery routines, or attacks. Time-sensitive and radio-bound actions occur locally on the node to minimize latency.

Each node is defined by its configuration, which records available hardware, supported radios, and networking setup. Configurations are stored locally so nodes can continue operating even when disconnected from the hub. This allows them to run autorun playlists at startup or collect data for later synchronization.

Nodes can operate in two modes. Under hub control, they act as remote execution points, carrying out tasks assigned by the central system and returning results. In autonomous mode, they continue executing playlists or plugins without hub connectivity, storing results locally until links are restored. This dual capability makes FISSURE adaptable to both connected and disconnected environments.

Distributing execution to tactical nodes gives FISSURE a scalable and flexible deployment model. A single user can experiment with one node on a desktop, or a network of nodes can be spread across vehicles, fixed sites, or UAVs for coordinated sensing and effects.

4.3 Plugin System

The plugin system is the primary mechanism for extending FISSURE. Instead of hard-coding functionality into the framework, capabilities are packaged as modular plugins that can be developed, shared, and deployed independently. This keeps the core lightweight while allowing users to adapt the system to new signals, protocols, or operational needs.

A plugin may include a Python script, a GNU Radio flow graph, or both, plus supporting files. Examples include detectors for new signals, classification routines, protocol decoders, attack scripts, IQ processing tools, and visualization utilities. When a plugin is imported, its metadata, such as name, version, capabilities, and dependencies, is registered in the database, making it available for tasking through the hub.

Plugins execute at the tactical node or at a processing engine that has the required hardware and environment. Results, logs, and artifacts return through the same messaging system used for core tasks, so behavior is consistent whether a plugin runs locally or remotely. Plugins can be shared openly or kept in private repositories for restricted use.

In practice, the plugin lifecycle and management follow these steps:

- **Authoring:** Plugins can be created inside the Dashboard today. A dedicated plugin builder application is in development to let authors design and package plugins without a full FISSURE installation.
- **Import and Registration:** New plugins are imported at the hub and recorded in the database with name, version, capabilities, and dependencies so availability and consistency can be tracked across nodes.
- **Distribution to Nodes:** The hub can push plugins to tactical nodes. Nodes confirm versions they already have installed, helping operators avoid mismatched or missing components. Planned enhancements include dependency checks before execution.
- **Execution Policy:** Core utilities remain part of the base framework. Specialized attacks, analysis routines, and experimental features run as plugins. Policy controls will continue to evolve, with items such as plugin signing and stricter controls for sensitive environments on the roadmap.
- **Version Reporting and Consistency:** Nodes report which plugins and versions are installed, and the hub reconciles against its registry to prevent mismatches or missing functionality across a deployment.
- **Unified Referencing:** The system is moving toward referencing non-core actions by plugin identifier rather than ad hoc file paths. This simplifies tasking, improves traceability, and makes plugin use more consistent across the framework.

By encapsulating detection, analysis, and effects into modular units, the plugin system allows FISSURE to evolve continuously. New ideas can be tested without altering the foundation, and specialized capabilities can be integrated without exposing the entire system. Together, these controls let plugins be shared, updated, and executed reliably across classrooms, research labs, and operational environments.

4.4 Processing Engines

Processing engines provide dedicated services for compute-heavy tasks that benefit from running outside the tactical node. By separating these functions, FISSURE avoids overloading systems directly connected to radios and can scale more effectively. Engines may run alongside the hub in smaller deployments or on separate machines when more processing capacity is required.

At present, two engines are supported:

- **Target Signal Identification (TSI) Engine:** Detects, conditions, and classifies signals. It takes IQ data as input, applies detectors, extracts features, and uses machine learning models to identify signals of interest. Outputs include detections with time, frequency, bandwidth, and power, along with feature vectors and classification results with confidence levels.
- **Protocol Discovery (PD) Engine:** Analyzes unknown or partially known protocols. It performs recursive demodulation, bit slicing, and pattern analysis to uncover message fields and structures. Outputs can include decoded frames, candidate field maps, identified cyclic redundancy check (CRC) parameters, and Lua dissectors for Wireshark.

Both engines interface with the hub for tasking and reporting. Results and artifacts are published back into the system, making them accessible to clients and the library. This division allows radio-bound actions to remain on nodes while heavier analysis is handled by components optimized for sustained processing.

4.5 Data Layer and Library

The data layer provides persistence for everything FISSURE collects, processes, and generates. It ensures that signals, metadata, and analysis results are preserved, reusable, and searchable across workflows.

At the center is a PostgreSQL database that maintains structured records for protocols, attacks, playlists, features, models, and plugin metadata. Linking results to their originating task, node, and configuration is expanding as the library matures so the system can support both manual analysis and automated workflows.

Artifacts sit alongside the database. These include IQ captures, decoded frames, PCAPs, logs, generated plots, and trained models. Artifacts may remain on the node for local use, be transferred to the hub, or be synchronized based on bandwidth and operational needs.

- **Library Entries:** Signals of interest from TSI or Protocol Discovery can be added with metadata such as time, frequency, bandwidth, and classification results. Protocol definitions, message fields, and CRC parameters are recorded as they are discovered so knowledge accumulates over time. Entries can be created automatically from workflows or added manually when analysts identify important details.
- **Artifact Referencing:** Large files, for example IQ captures or packet traces, may stay on the node or hub, while references and metadata are recorded in the database where enabled. This avoids duplication, links results to the conditions and tools that produced them, and keeps data discoverable. Richer cross-linking and search are in progress as part of the evolving data layer.

Caching supports operations in constrained environments. Nodes can store results locally and forward summaries or artifacts when connections are restored. Over time, store and forward synchronization and selective replication will make large datasets easier to manage in distributed deployments.

Provenance is recorded at a basic level today, including timestamps, node identifiers, and configurations used for each task. Future development will expand this into richer audit trails that combine policy, provenance, and evidence so results remain reproducible and trustworthy.

The combination of structured records and stored artifacts turns one-time experiments into reusable knowledge that benefits operators, researchers, and students.

4.6 Dashboard and Client Apps

The Dashboard is the primary user interface for FISSURE. Built with PyQt, it provides a tabbed environment for detection, classification, protocol discovery, IQ data management, and attack tools. Each tab maps directly to backend functions, with results returned through callbacks so operators can view logs, progress, and artifacts in context.

Client applications extend this model to other environments. A lightweight mobile app offers simplified tasking and status updates when operators need quick control without the full Dashboard. TAK integrations bridge FISSURE into ATAK, WinTAK, and WebTAK, allowing alerts and detections to appear in the shared operational picture.

Each client type maintains its own task and status space. This separation prevents conflicts between Dashboard, mobile, and TAK sessions while still allowing bridges when explicit integration is desired.

Multiple entry points make FISSURE's capabilities accessible across diverse use cases. Analysts in a lab can rely on the full Dashboard, operators in the field may use mobile apps, and teams invested in TAK can receive alerts and reports in their existing systems. These clients extend the reach of the framework while preserving a consistent structure.

5. THEORY OF OPERATION

The theory of operation describes how FISSURE functions in practice. While the architecture defines the components and their roles, this section explains how users interact with those components, how tasks are executed, and how results move through the framework.

Operation falls into two broad categories: manual workflows, where the user drives actions directly through the Dashboard, and automated workflows, where playlists, triggers, or future modes of operation guide activity without constant input. In both cases, the hub coordinates, tactical nodes handle radio-bound tasks, and processing engines perform heavier analysis.

Messaging, logging, and connectivity remain consistent across these workflows. Requests flow from clients to the hub, results return to the originating client, and logs provide transparency at every step. Nodes can continue working when disconnected, while the hub resumes synchronization once links are restored.

Together, these patterns allow FISSURE to support exploratory analysis, training exercises, and coordinated multi-node operations within a single system.

5.1 Interaction Model Overview

FISSURE turns an operator's action into coordinated work across nodes and engines. A client submits a request, the hub selects an appropriate target based on capability and load, the target executes while streaming status and logs, and results flow back to the originating client in context.

Operation supports two modes. In manual mode, the operator launches tasks from the Dashboard or another client and can adjust parameters as execution proceeds. In automated mode, playlists, schedules, or triggers start tasks without supervision. When bandwidth is limited or connectivity is intermittent, nodes keep artifacts locally and send compact summaries first. Operators can transfer artifacts when connectivity permits, with store and forward planned for a future release.

Each task follows a predictable lifecycle: created, dispatched, running, producing results, then completed, failed, or canceled. The system records an identifier, timestamps, the originating client, the selected node or engine, and the configuration in use. Status updates stay lightweight so interfaces remain responsive, and logs are preserved for review.

Execution is placed where it fits best. Radio-bound actions stay on nodes to reduce latency and jitter, while compute-heavy routines run on engines for throughput and scale. Retries, timeouts, and idempotent task identifiers guard against duplication after reconnects.

Results can return directly to the client, be written to the library, or both. Summaries capture key fields such as time, frequency, bandwidth, power, and confidence, with links to artifacts like IQ files or PCAPs.

5.2 Manual Workflow

Manual operation is the most direct way to use FISSURE. The operator selects a tactical node and issues tasks through the Dashboard. By default, actions target the currently active node, unless a workflow explicitly supports coordination across multiple nodes.

The active node is chosen in the top bar of the Dashboard with a right-click. Once selected, any action in the Dashboard is routed through the hub to that node. The operator can start or stop tasks on demand, with results and status shown in the corresponding tab.

The Dashboard supports a wide range of manual actions, including:

- Running single-stage attacks such as Python scripts or GNU Radio flow graphs.
- Executing multi-stage attacks that chain several actions with timing controls.
- Performing fuzzing at the protocol field level or by modifying flow graph variables during runtime.
- Launching Scapy-based packet injections.
- Recording or replaying IQ data.
- Running signal archive replay playlists.
- Starting and stopping autorun playlists.
- Using Target Signal Identification (TSI) functions such as detectors and signal conditioners.
- Running protocol discovery flow graphs and related actions.

Each action uses callbacks. The GUI backend packages the request, sends it to the hub, and the hub routes it to the active node. Responses return to the same tab as logs, progress updates, or artifacts. This consistent pattern keeps workflows predictable and easy to follow.

Manual operation gives users complete control and visibility. It is well suited to exploratory analysis, hands-on training, and situations where conditions change too quickly for automation to be effective.

5.3 Automated Workflow

Automation in FISSURE is centered on autorun playlists. These let users define a sequence of actions that a node executes without manual intervention. Playlists can start automatically at boot or be launched on demand, supporting both continuous monitoring and one-time scripted runs.

Each playlist item runs in its own thread, which allows multiple radios and actions to operate at the same time. Playlists are typically composed of single-stage or multi-stage attacks, and they can also include IQ recording, replay, or analysis routines. As long as the hardware can support it, a single node can run concurrent workflows.

Timing is configurable. Operators can set global or per-item delays, repetition intervals, and stop conditions. Triggers can be attached so execution begins only when specific conditions are met, for example RF events, file system changes, environmental inputs, or time-based triggers.

The hub handles task routing, then the node executes the playlist locally. This reduces dependency on constant connectivity and lets nodes keep working when disconnected. Results and status return through the hub when links are available, keeping the operator informed.

Although automation today is limited to playlists, it already scales operations. Nodes can be staged in advance, started remotely, and left to run predefined behaviors. Future development will expand automation into coordinated modes of operation, but autorun playlists are the current foundation.

5.4 Results, Artifacts, and Logging

Tasks may produce artifacts, or they may only emit status updates and logs. Handling both consistently makes workflows reproducible and results reusable when they exist.

- **Artifact Types:** IQ snippets and recordings, decoded frames and PCAPs, feature vectors and classifications, trained models, Lua dissectors, plots, and run logs.
- **Return Paths:** Status and concise summaries return immediately to the originating client and appear in the relevant tab. Artifacts remain on the node by default, and the client records references with producing node, timestamps, sizes, and other basic details. Operators export or transfer artifacts when connectivity permits. For constrained links, use summaries and references rather than bulk transfers. Store-and-forward capabilities are on the roadmap.
- **Planned Database Integration:** Future releases will record artifact references and richer metadata in the database, linking results to the node, configuration, and task that produced them. The goal is a searchable library of signals, protocols, and workflows for follow-on analysis or training.
- **Logging Model:** Components support standard levels, info, warning, error, and debug. Logs are written locally as text files, capped and rotated to prevent uncontrolled growth on small systems. In the UI, a Log tab provides filtering by source, destination, type, and level so operators can focus on the entries that matter. The terminal console shows real-time log output from active tasks and components for quick diagnosis while work is running.

Treating outputs as structured artifacts, paired with consistent logging, keeps results organized, traceable, and easy to reuse across education, research, and operations.

5.5 Connectivity and Resilience in the Field

FISSURE coordinates distributed nodes under a range of network conditions, from stable IP links to intermittent or bandwidth-limited paths. The framework maintains useful behavior when links drop, and it keeps overhead low when capacity is scarce.

- **IP Connections and Heartbeats:** On standard IP links, nodes and the hub exchange periodic heartbeats to confirm connectivity and report status. If a link drops, both ends

retain state and coordination resumes when the connection returns, avoiding unnecessary restarts.

- **Low-Throughput Links:** On constrained networks such as Meshtastic, heartbeats are suppressed to conserve airtime. Messages use compact function codes with a shared lookup table to keep payloads small. The UI surfaces summaries and references first, and bulk artifact transfers are typically deferred until connectivity improves.
- **Retries and Timeouts:** The hub manages task lifecycles with sensible retries and timeouts. Failed requests are retried where appropriate, and partial results are reported clearly so operators know what completed.
- **Link-Loss Behavior:** During outages, nodes can continue local work when configured to do so. When links return, status and new results resume streaming to the client. Artifacts created while disconnected remain on the node until an operator exports or transfers them.
- **Isolation Mode:** Nodes can run autonomously, executing playlists or local tasks without hub supervision. In this mode they do not accept live connections and prioritize completing assigned work. Outputs are stored locally for later retrieval.
- **Store and Forward (Planned):** Future releases will queue results and artifacts while offline, then forward them automatically once connectivity is restored.

These behaviors keep tasking responsive on good networks, conserve scarce bandwidth on limited links, and preserve work during outages so operations remain productive in the field.

5.6 Execution on Nodes vs Engines

FISSURE divides tasks between tactical nodes and processing engines to balance responsiveness with computational power. This separation ensures that time-sensitive actions tied to radios happen close to the hardware, while heavier analysis is offloaded to components designed for sustained processing.

- **On Tactical Nodes:** Nodes handle actions that require direct access to radios and sensors. This includes tuning, capture, transmission, IQ recording and playback, packet injection, and live demodulation flow graphs for known protocols. Nodes also run single-stage and multi-stage attacks, Scapy-based injections, and quick-look transforms, along with plugin code and flow graphs tied directly to radios. Executing these tasks locally minimizes latency and keeps workflows responsive.
- **On Processing Engines:** Engines run tasks that are computationally intensive but not tied to real-time radio interaction. The Target Signal Identification (TSI) engine handles detection, conditioning, feature extraction, and classification pipelines. The Protocol Discovery (PD) engine focuses on recursive demodulation, bit slicing, CRC analysis, candidate field mapping, and optional Lua dissector generation. Engines also handle batch processing of IQ files offloaded from nodes, and they scale independently of node workloads.
- **Split Strategies:** Tasks can be divided between nodes and engines. A node might capture a limited IQ window and run lightweight models locally, returning classification results to the operator, while deeper protocol discovery proceeds on an engine. In other cases, larger IQ files are offloaded from the node and batch processed on an engine, adapting to bandwidth and resource conditions.
- **Selection and Control:** The hub selects targets based on capability and current load, including radio type and drivers, installed plugins and models, and available compute and storage. The operator can direct placement by choosing a specific node in the Dashboard or by assigning an engine where supported. If a required radio is in use or a dependency is missing, the system surfaces a clear notification so the operator can switch devices, resolve the issue, or retry.

5.7 Operator Experience Cues

The framework gives operators clear, continuous feedback. Because workflows can involve multiple nodes, engines, and clients, users should always know which system is active, what task is running, and how results are returned.

- **Active Node Indicator:** The Dashboard shows which node is currently active so actions go to the intended target and context switches are obvious.
- **Task State and Progress:** Each action displays its current state with progress updates and recent logs. Operators can start, pause, stop, or restart tasks as needed. A Logs tab provides filtering by source, destination, type, and level, and the terminal console shows live output for quick diagnosis.
- **Popup Status Bar:** A pop-up status bar provides more detail when tasks start, progress, or complete, with quick access to related tabs, logs, and artifacts.
- **Notifications and Conflicts:** The system surfaces clear notifications for conflicts or missing components, for example radios already in use, missing plugins, or version mismatches between nodes and the hub, so operators can act without guesswork.
- **Namespaced Status:** Dashboard, mobile, and TAK clients maintain separate status namespaces. Each user sees only the activity relevant to their session, which avoids confusion across roles and devices.

These cues keep operators oriented and confident as they manage complex or distributed workflows.

6. SUPPORTING TOOLS AND ECOSYSTEM

In addition to its core components, FISSURE includes supporting tools and ecosystem features that enhance usability and expand what users can accomplish. These tools are optional, but they make the framework easier to learn, extend, and integrate into broader workflows.

Supporting tools include standalone flow graphs for quick experiments and teaching, third-party programs that launch from the Dashboard, and lessons designed for classrooms or hobbyist use. Optional containers provide visualization and analysis services, and the installer simplifies deployment across different environments.

Together, these elements strengthen the ecosystem around the framework. They make FISSURE approachable for newcomers, flexible for researchers, and effective for operators who want everything in one place.

6.1 Standalone Flow Graphs

FISSURE includes a collection of standalone flow graphs that launch directly from the Dashboard menus. These are separate from the plugin lifecycle and provide ready-made examples for experimentation, teaching, or quick testing.

Standalone flow graphs act as starting points for prototyping new ideas, let students explore GNU Radio concepts without building everything from scratch, and give operators simple tools for rapid awareness. Because they are independent of the plugin system, they can be modified freely without affecting the rest of the framework.

By including these examples, FISSURE lowers the barrier to entry for new users while giving advanced users a foundation for custom development. They demonstrate how the framework interacts with software-defined radios and data streams in a way that is easy to understand and extend.

6.2 Third-Party Tools

FISSURE integrates a variety of third-party tools that expand its functionality without forcing users to leave the environment. These tools are accessible from the Dashboard menus, making it easy to launch external applications or reference utilities as part of a workflow.

Examples include Wireshark for packet inspection, CyberChef for data transformation, and protocol references or calculators for quick lookups. Visualization tools such as Gqrx, Inspectrum, and IQEngine can be launched to analyze IQ data outside the Dashboard. By embedding these options into the interface, FISSURE streamlines workflows that would otherwise require switching among separate tools.

The goal is not to replace these programs but to make them more accessible. Researchers and operators can keep using familiar tools while staying within FISSURE, and students can see how external utilities complement internal workflows in practice.

6.3 Lessons and Educational Modules

FISSURE includes lessons and educational modules designed to teach RF, spectrum operations, and cybersecurity concepts through direct interaction with the framework. These modules are accessible through the Dashboard and range from introductory exercises to advanced workflows.

Lessons may include step-by-step guides for using flow graphs, tutorials on protocol analysis, or demonstrations of how to capture and replay signals. Some focus on specific technologies such as OpenBTS, Lua dissectors, ESP-based boards, or radiosonde tracking. Others cover broader topics like modulation schemes, IQ data handling, or attack workflows.

These modules are valuable in classrooms, research labs, and hobbyist groups. They provide hands-on experience with software-defined radios and analysis techniques, while giving instructors ready-made material that can be adapted for different skill levels.

6.4 Optional Containers and Extra Services

Beyond the core framework, FISSURE can be paired with optional containers and services that enhance analysis and visualization. These are not required, but they provide added value for users who want expanded capability.

For example, IQEngine runs in a Docker container and provides advanced visualization of IQ captures. By connecting IQEngine to FISSURE, users can browse and inspect large datasets with an interface tailored for signal analysis. Other containers may include web-based tools or specialized services that complement workflows already built into the Dashboard.

These extra services run alongside FISSURE without complicating the core installation. Users enable them when needed, keeping the system lightweight while allowing richer environments for research, teaching, or operations.

6.5 Installer and Deployment Options

To make the framework accessible, FISSURE provides an installer that simplifies deployment across supported systems. Users can select which components to include, installing a full standalone system or a lightweight tactical node tailored to the mission.

The installer supports several deployment modes. A full installation includes the Dashboard, hub, database, and supporting tools for standalone use or as a central controller. A tactical node installation includes only the components required to connect radios and execute playlists, keeping the footprint small. Presets allow quick switching between defaults for controller systems and remote nodes.

FISSURE and its components can also run inside containers for reproducibility and portability. The current database deployment uses Docker, which loads a database dump file at startup. Future work will explore container systems such as Apptainer to better support high-performance computing and edge environments.

By packaging the system in a clear installer and providing container options, FISSURE lowers the barrier to entry for new users while giving advanced users flexibility in how they deploy and scale.

6.6 Import/Export and Presets

FISSURE supports configuration portability so workflows are easy to repeat, share, and restore across systems.

- **Import and Export:** Playlists, CSVs, and other configuration files can be imported and exported from the Dashboard. This makes it straightforward to share workflows, replicate experiments, or back up operational setups. For example, an autorun playlist built on one system can be exported and loaded on another node with minimal changes.
- **Presets and Options:** The options menu lets users save presets that capture preferred configurations. On startup, FISSURE can restore the last-used setup or load system defaults, depending on user preference. Presets reduce setup time when moving between projects or switching roles.
- **Configuration Files:** The Dashboard stores its settings in FISSURE Dashboard config YAML files, which the Options menu reads and writes so changes persist across sessions. Tactical nodes read a sensor node config YAML at boot to load radio, path, and workflow settings appropriate for that node. These files can be versioned and distributed like other artifacts to keep controller and node configurations aligned.

7. DATA AND SIGNAL FLOW

This section shows how a signal moves through FISSURE from first detection to useful results. It focuses on what changes hands at each step, where work happens, and how outcomes become artifacts that can be reviewed or reused.

Readers will see two ways the same pipeline is used. Today, analysts drive the steps directly in the Dashboard. Over time, more of that path will run as coordinated automation, with the hub supervising progress and the operator staying in control of decisions.

Tactical nodes handle radio work close to hardware, engines take on heavier analysis, and the hub keeps the pieces aligned so results return to the right place. The diagram outlines the flow, and the subsections explain each stage.

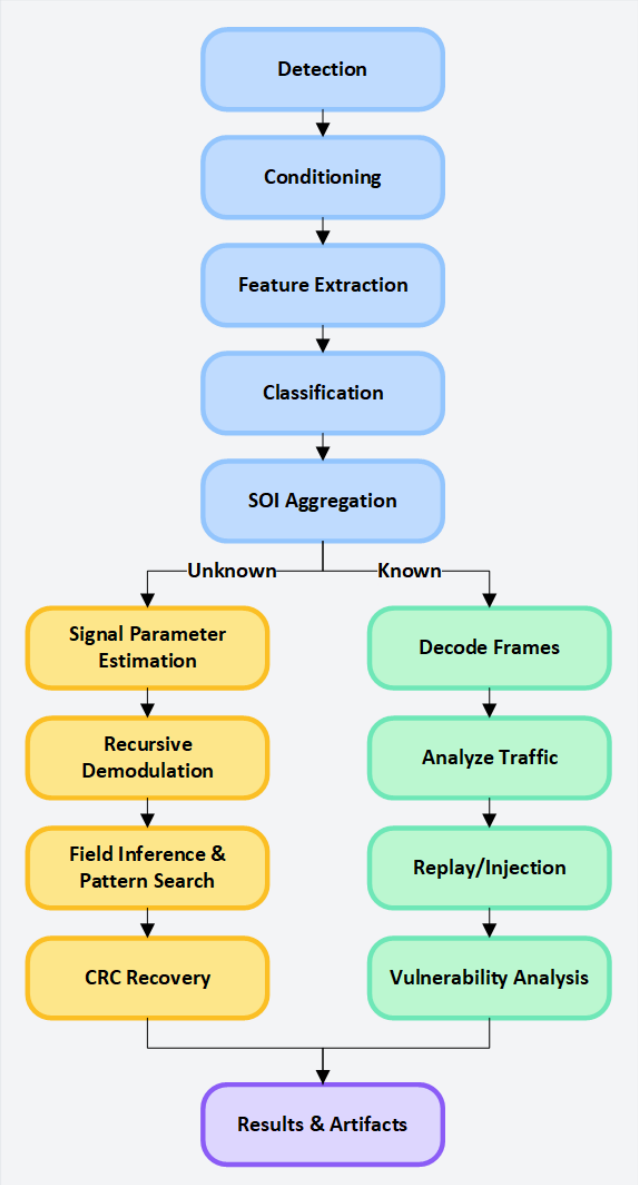

Figure 4. End-to-end signal pipeline in FISSURE. Detection and conditioning occur at tactical nodes, features can be extracted and classified by engines, and known vs unknown protocols follow distinct branches. Known protocols focus on decoding, analysis, and replay or injection, while unknown protocols undergo recursive demodulation, field inference, and protocol discovery. Results flow back to clients, with artifacts recorded in the library.

7.1 Pipeline Overview

This overview orients the reader to the end-to-end path a signal follows in FISSURE. A single pipeline runs from detection to outcomes, with two ways to traverse it: analyst driven today and coordinated automation in the target state. The known-protocol branch focuses on recognition, decoding, analysis, and, where authorized, replay and injection. The unknown-protocol branch focuses on discovery, recursive demodulation, field inference, fuzzing, and tool handoff. Both paths cover the full cycle of RF reverse engineering: monitor, collect, replay, analyze, research, demodulate, inject, and assess vulnerabilities. Results return to the originating client and, where enabled, are saved as artifacts and references for later work.

7.2 Manual Analyst Flow (Today)

Most workflows in FISSURE are manual and analyst-driven. The operator selects an active node in the Dashboard's top bar, issues actions from the relevant tab, and the request is routed through the hub to that node. Results and status return to the same tab for review, with detailed filtering in the Logs tab and live output in the terminal console when needed.

Common manual tasks include:

- Running single-stage or multi-stage attacks.
- Launching fuzzing at the protocol-field level or by modifying flow graph variables.
- Executing Scapy-style injections or archive replay playlists.
- Recording and replaying IQ data for inspection.
- Starting detectors and signal conditioners from the TSI tab.
- Running demodulation flow graphs and protocol discovery actions.

Each action follows a consistent pattern. The Dashboard backend packages the request, sends it to the hub, and the hub routes it to the targeted node. Results, logs, and artifacts flow back along the same path and are displayed through tab-specific callbacks. Artifacts remain on the node by default; the client records references and summaries, and operators export or transfer files when connectivity and workflow allow.

Manual workflows are especially valuable for research, training, and exploratory analysis. They keep the operator in full control, allow parameter changes on the fly, and provide immediate feedback at each step.

7.3 Automated Pipeline (Roadmap)

In the automated pipeline, FISSURE runs the same stages end to end under hub supervision. Nodes and engines advance through detection, classification, and protocol discovery, and the operator stays in the loop to review results and authorize actions. This is a roadmap capability that builds on today's manual workflows.

Typical automated sequence:

1. Deploy tactical nodes in the field and register them with the hub.
2. Activate the Target Signal Identification (TSI) engine to begin scanning.
3. Run detectors to identify signals of interest.
4. Tune to selected frequencies, isolate or filter the signal, and capture IQ.

5. Extract features from the capture.
6. Classify using stored models and record confidence scores.
7. Check against the protocol library.
    a. If known, capture traffic and analyze with decoding tools.
    b. If unknown, send data to the Protocol Discovery (PD) engine.
8. For unknown signals, perform recursive demodulation to obtain a stable bitstream, then apply field inference, pattern search, and CRC testing.
9. Produce preliminary protocol definitions, confidence summaries, and optional Lua dissectors for Wireshark.
10. Recommend relevant attacks or fuzzing workflows, with operator approval required before execution.

Many of these steps exist today in manual form. The automated pipeline connects them under hub coordination, enabling larger-scale operation and multi-node workflows while keeping the operator in charge of decisions. Results and summaries return to the initiating client, artifacts remain on the node by default with references recorded in the UI, and planned features will streamline storage and reuse.

7.4 Processing Paths: TSI and Protocol Discovery

FISSURE turns raw detections into actionable results along two complementary paths. TSI refines signals into structured events and classifications. Protocol Discovery investigates unknowns to produce bitstreams, fields, and dissectors.

- **TSI Event Path (Detections → Conditioning → Features → Classifications):** Detectors run on tactical nodes, scanning ranges or fixed channels. When a threshold is crossed, the node opens a temporary ZMQ subscription and forwards concise alerts to the hub for display in the Dashboard. Conditioning then segments larger IQ captures into workable windows and applies basic transforms where needed. Today these steps run on the node, with future options to choose whether results are streamed to the hub or retained locally. Feature extraction and classification follow, producing vectors and predictions with confidence scores. Results can be aggregated into Signals of Interest (SOIs) with time, frequency, bandwidth, and classification summaries. Where configured, metadata is recorded for later lookup, and expanded database linking remains on the roadmap. The authoritative location of large signal files is a design choice that can favor node storage or hub transfer based on bandwidth and workflow.
- **Protocol Discovery Path (Unknown Protocols):** For known protocols in the library, live demodulation flow graphs can run on a node in real time or operate offline on stored IQ files. When a signal does not match an existing definition, the PD engine performs recursive demodulation to obtain a stable bitstream, then applies field inference, pattern search, and CRC testing. Outputs include decoded frames, candidate field maps, and identified CRC parameters. When possible, Lua dissectors are generated for Wireshark so analysts can inspect traffic with named fields. Artifacts and summaries can be saved with metadata for reuse, with richer database integration planned as part of the evolving data layer.

7.5 IQ Handling: Recording, Transfer, and Playback

IQ handling is central to FISSURE. The framework supports recording at the node, moving files when needed, and replaying them for analysis or effects.

- **Recording:** When recording is started from the Dashboard, the tactical node captures IQ and saves the file locally on that node. If the Dashboard is running on the same machine as the recording node, plots can open from the local path. If the Dashboard is remote, the file must be transferred before review elsewhere.
- **Transfer:** Files move between nodes and the hub using the same messaging and file-transfer utilities that support other actions. Transfers can occur automatically when artifacts are logged; for large files, the system asks for confirmation to avoid saturating links. On constrained links, the UI surfaces summaries and references first, and the system defers large transfers by default until connectivity improves. Operators can also initiate transfers manually from the Dashboard's navigation views. Store-and-forward is on the roadmap.
- **Playback:** To replay a capture, the hub sends an IQ file to a selected node. The node transmits through radios or processes the file with a flow graph. Archive replay playlists schedule sequences of files in order. During playback, the Dashboard indicates which file is active so the operator can track progress.
- **Provenance and Reuse:** The UI records basic details such as producing node, timestamps, and sizes, with richer database linkage planned. Captures can be reused for training, demonstrations, and repeatable tests without reacquiring signals in the field.

7.6 Outputs and Integration: Alerts and Reports

FISSURE surfaces outcomes as lightweight alerts and structured summaries so results are visible in real time and easy to act on. Deeper reporting and export are roadmap items.

- **Alerts:** Actions such as attacks, playlists, detectors, and discovery steps can generate alerts to the Dashboard. Alerts indicate outcomes like successful execution, detection of a condition, or discovery events, and include context such as producing node, time, signal details, and quick links to logs or artifacts on the node. Alerts appear immediately and can prompt follow-up actions that the operator initiates. When configured, alerts can also be forwarded to external systems.
- **TAK Integration:** When the TAK bridge is enabled, selected alerts and status updates publish to a TAK server so they appear in ATAK, WinTAK, or WebTAK, often with map location, short text, and references back to the originating task. Dashboard, mobile, and TAK clients use separate namespaces, so only the intended audience sees each stream.
- **Reports and Recommendations:** Tasks can generate structured summaries in the UI describing the target signal, parameters used, results obtained, and suggested next steps, for example replay, injection, or fuzzing where appropriate. Persisting reports alongside run outputs and exporting them for sharing are planned features.

- **Planned Approvals and Audit:** Future releases will add optional approval gates for effects that change the environment, along with an audit trail recording who approved actions, when, and on which node.

8. TECHNICAL STACK AND INTEGRATION

FISSURE's stack is built to be approachable, modular, and dependable in the field. The core is written in Python with a PyQt5 desktop Dashboard. Radio work is carried primarily by GNU Radio flow graphs, while Python scripts handle cases that do not involve GNU Radio. Metadata is kept in PostgreSQL and large artifacts stay on the disks where they are produced, moving only when it makes sense.

Integration is part of the design. Control travels over ZMQ on IP networks, with Meshtastic available when links are low throughput. Plugins add capability without changing the core, and familiar tools such as Wireshark and CyberChef remain close at hand. When enabled, a TAK bridge publishes selected alerts and status into ATAK, WinTAK, or WebTAK so results can appear in the same environment operators already use.

These choices keep the framework easy to learn, flexible to extend, and steady under operational constraints. The sections that follow describe how each layer fits with the others, with earlier chapters covering behavior and workflows so this chapter can stay focused on implementation and integration.

8.1 Language, UI, and Config

FISSURE is written in Python, which keeps the code readable, integrates cleanly with drivers and analysis libraries, and lets teams prototype quickly. The desktop Dashboard is built with PyQt5, so interfaces are native and cross-platform without special runtime requirements.

Interface elements are laid out with Qt Designer, then wired to Python logic. This keeps view and control code separate, makes tab layouts easy to maintain, and shortens the path from a sketch to a working tool.

Configuration is plain YAML so settings are human-readable and easy to version. The Dashboard reads and writes FISSURE Dashboard config YAML files to persist options and presets, and each tactical node loads a sensor node config YAML at boot to set radios, paths, and defaults. Playlists and other artifacts follow the same pattern. For details on importing, exporting, and presets, see Section 6.6.

8.2 Database and Persistence

PostgreSQL serves as the framework's registry and index. Plugins own the actual data files; the database stores references to those file locations and records parsed fields from their contents so items can be searched and retrieved quickly. It also maintains the plugin registry (name, version, capabilities, dependencies) and central catalogs such as device profiles and protocol templates that benefit from a single source of truth.

Deployments initialize PostgreSQL from a baseline dump to keep schemas and starter content consistent. From there, the registry grows as plugins are imported and catalogs are updated, with migrations handled during installation, see Section 8.3.

The hub and nodes ship with the same source tree and plugin assets, so both ends import identical helpers for message schemas, parsers, and serialization; for example, function-code lookup tables for low-throughput links are packaged as config or plugin files on both sides, keeping codes in sync without relying on the database.

8.3 Build, Packaging, and Deployment

FISSURE favors a straightforward build and install so teams can stand up a controller or a tactical node with minimal ceremony. Source is managed in GitHub, core code lives in the fissure/ directory, and GNU Radio out-of-tree modules are tracked as git submodules so the installer can fetch the right branches during setup.

- **Installer Roles:** The installer supports a full controller or a lightweight tactical node. It pulls submodules, installs dependencies, writes initial config YAMLs, and registers services appropriate to the selected role. Plugin packaging and distribution are covered in Section 4.3.
- **Database Packaging:** PostgreSQL runs in a Docker container and is initialized from a baseline dump at startup. This provides a consistent schema and starter content across systems, with data persisted on a host volume.
- **Migrations and Seeds:** During installation or update, schema migrations and baseline content loads are applied so deployments stay aligned. Routine updates to catalog data are handled the same way.
- **Updates and Releases:** The framework updates through git. Operators pull the latest commits and update submodules when needed. Stable points are marked with snapshot tags that teams can pin to or roll back to if they prefer a fixed baseline for exercises or operations.
- **Portability Roadmap:** Apptainer is under evaluation for environments that prioritize reproducibility and isolation, for example high-performance computing (HPC) clusters or edge systems. Dependency management improvements, such as adopting Poetry, are also being considered to make packaging more predictable.

This approach keeps day-to-day installs native and fast, uses containers where they add consistency, and leaves room to adopt more formal packaging where mission needs require it.

8.4 Signal Processing and Flow Graphs

GNU Radio provides the foundation for FISSURE's signal processing. Flow graphs handle detection, demodulation, protocol discovery, fuzzing, and replay. Python scripts complement GNU Radio for workflows that bypass digital signal processing (DSP) chains, for example Wi-Fi utilities, Scapy-based injection, PCAP parsing, device provisioning, and targeted protocol tests.

Flow graphs can be launched from the Dashboard or run headless through plugins. Parameters are set through presets or per-run inputs, then passed to the flow graph at start. Where supported, operators can adjust variables at runtime, for example frequency, gain, thresholds, or symbol timing, without restarting the job.

Control is consistent with the rest of the framework. Start and stop are issued from the UI, status and logs stream back to the same tab, and artifacts are handled as described in Section 7.5. Conflicts such as a radio already in use or a missing out-of-tree block are surfaced with clear notifications.

Flow graphs are stored with the source and packaged in plugins so versions can be tracked along with any required out-of-tree modules. Authors are encouraged to keep flow graphs small and

composable, favor shared utility blocks over one-off customizations, and write with reuse in mind so others can read, extend, and share quick-look transforms without friction.

For standalone flow graphs that serve as teaching examples or quick experiments, see Section 6.1. For the plugin packaging and distribution model, see Section 4.3.

8.5 Machine Learning and AI Tools

FISSURE uses machine learning to turn IQ data into useful labels that guide what to do next. Models are built with familiar libraries such as scikit-learn and TensorFlow, packaged as plugin assets, and loaded by the Target Signal Identification pipeline when classification is requested.

Feature extraction produces vectors from short IQ windows. Classifiers return a predicted class and confidence, and the Dashboard shows both so an analyst can decide whether to proceed, retune, or collect more data. The classification step is part of the TSI path, not a separate system.

Training follows a simple pattern. Analysts prepare labeled CSVs with the IQ Dataset Builder in the IQ Data Tools, which can apply controlled randomizations such as amplitude scaling, phase shift, and noise before export. Training scripts fit models from those datasets, and the resulting models are versioned and shipped with plugins so nodes and engines use the same files, see Section 4.3.

Current use focuses on classification and feature analysis. Roadmap work will expand ML's role in detection, prioritization, and recommendations while keeping operator approval in the loop.

8.6 Networking Transports and Messaging Formats

FISSURE connects the Dashboard, hub, nodes, and engines across fast IP links and low-throughput meshes. This section focuses on how bytes are carried and how messages are shaped.

- **ZMQ on IP:** Control travels over request–reply sockets, with logs and status on streaming sockets. The hub can handle many concurrent actions without blocking, and addressing uses stable node or engine identifiers. Each message carries a small envelope with fields like *action_id*, *source*, *target*, *namespace*, *timestamp*, and *params*.
- **Meshtastic on Constrained Links:** Messages are reduced to compact function codes plus minimal parameters. A shared lookup table is packaged with both hub and nodes so codes match without a database. Links stay quiet to conserve airtime, with short acknowledgements where needed. Heartbeats are suppressed; see 5.5 for link behavior.
- **Schemas & Payloads:** Control messages use lightweight JSON or YAML. Field names are stable, with version tags for compatibility. Artifacts are referenced, not inlined, so payloads remain small; bulk files move via the IQ handling paths in 7.5.
- **Namespaces & Routing:** A namespace field keeps Dashboard, mobile, and TAK streams distinct. Bridges can be enabled explicitly when cross-system visibility is desired, see 4.1 and 7.6.
- **Extensible Transports:** Additional links can be added by reusing the same envelope and, on low-throughput paths, the function-code pattern. No changes to higher-level workflows are required.

Security details for certificates and roles are covered in 8.9.

8.7 TAK Integration

FISSURE can publish detections, alerts, and status into the TAK ecosystem so results appear where operators already work.

- **Enablement:** Configure the TAK server endpoint, load the required certificates into FISSURE, and select which events to publish. Certificates are used to authenticate clients and encrypt traffic.
- **What is Published:** Alerts and status summaries, typically with time, location, and brief context. Entries can include references back to the originating task for follow up.
- **Namespaces:** Dashboard, mobile, and TAK streams remain separate. Only the items explicitly selected for TAK are forwarded.
- **Plugins & Extensibility:** Lightweight TAK-side plugins are planned to improve tasking and visualization. The bridge is designed to accommodate additional TAK data products as they are needed.

8.8 API and Interfaces

FISSURE does not expose a formal external API today. Integration happens through Python calls inside the framework and through the messaging layer used by the hub, nodes, and engines.

- **Message Schemas:** Control messages use compact JSON or YAML with stable field names and version tags for compatibility. Payloads reference artifacts rather than embedding large files.
- **Plugin Entry Points:** Plugins register callable actions with the hub, making new capabilities available without changes to the core.
- **External Tool Hooks:** Wireshark can consume Lua dissectors and PCAP outputs, while CSV and YAML files support exchange with other analysis pipelines.
- **API Roadmap:** A generated API is under consideration, for example FastAPI or gRPC backing selected Python functions, to support external automation and system-to-system integration.

8.9 Security and Access Control

FISSURE balances openness for research with safeguards for operational use. Security focuses on authenticated links, predictable execution, and a roadmap for tighter controls.

- **Authentication & Transport Security:** Client–hub connections on IP use certificates for mutual authentication and encrypted transport. Low-throughput links inherit encryption and key management from Meshtastic.
- **Namespaces & Publication Controls:** Status and alerts are namespaced so each client sees only its own stream. Forwarding to external systems, such as TAK, is explicit and scoped to selected items.
- **Secrets & Configuration:** Database credentials and certificates are managed per deployment. Installers initialize sensible defaults, and teams control certificate issuance, rotation, and revocation.
- **Role-Based Access Control (Roadmap):** Planned roles will govern GUI actions, hub operations, and plugin

execution, enabling least-privilege setups for operators, analysts, and developers.

- Plugin Integrity & Policy (Roadmap): Planned measures include plugin signing, dependency verification, and execution policies that distinguish trusted mission packages from experimental code.
- Audit & Approvals (Roadmap): Planned audit trails will record who initiated actions and when, with optional approval gates for effects that change the environment.

8.10 Documentation and Modeling

Clear references and shared models make the framework easier to learn, teach, and extend.

- **User Documentation:** Maintained on ReadTheDocs, synchronized with GitHub, and viewable online or offline. Content can be exported to PDF for static use in restricted environments.
- **System Modeling:** Systems Modeling Language (SysML) and Unified Modeling Language (UML) diagrams (created in Modelio) capture architecture, interfaces, and key use cases. Selected diagrams are folded into the user docs to support onboarding and integration.
- **Roadmap & Visuals:** Public, non-sensitive dashboards built with Plotly show technology hierarchy and development progress, giving readers a quick view of direction without exposing operational details.

9. CURRENT CAPABILITIES

FISSURE is an operational framework for RF reverse engineering, signal analysis, protocol exploration, and effects testing. It is usable today in classrooms, labs, and field environments while broader automation and integration continue.

FISSURE can run as a standalone system or be split into a controlling setup and one or more tactical nodes. With the Dashboard and plugins, users can detect and condition signals, manipulate and replay IQ, extract features and classify, explore protocols, and execute single- or multi-stage attacks. The subsections that follow summarize the capabilities available now: installation and deployment; signal detection and conditioning; feature extraction and classification; protocol discovery; attacks and fuzzing; IQ Data Tools; sensor-node features; library and admin tools; menus and extras; and current limitations with roadmap connections.

9.1 Installation and Deployment

FISSURE scales from a single laptop to a distributed set of tactical nodes. It can be installed as a standalone system or split into a controlling setup and one or more nodes.

- **Installer:** A graphical installer enables component selection and writes starter configuration files, supporting full standalone, central controller, or lightweight tactical node builds.
- **Deployment Modes:** Standalone includes the Dashboard, hub (HIPRFISR), database, and supporting tools for labs or single-operator use. A controlling setup runs the hub and the Dashboard to manage remote nodes. A tactical-node install includes only the components needed to connect radios and execute playlists, keeping the footprint small while remaining hub-compatible.
- **Configuration:** Remote nodes load a configuration file that specifies networking transport (IP or Meshtastic) and details such as IP addresses or serial ports. GPS may be provided by a receiver or by a Meshtastic radio. Nodes may optionally beacon position.
- **Behavior:** IP-connected nodes behave similarly to local instances, with additional latency. Meshtastic-connected nodes currently support autorun playlists at boot or by configuration; Dashboard-driven actions over low-throughput links are planned.
- **Probing:** The Dashboard can query nodes for hardware, radio, and system information to provide quick status without direct access.

These options allow deployments to be tailored for classroom training, research, and field operations with minimal friction.

9.2 Signal Detection and Conditioning

FISSURE provides tools to detect, isolate, and prepare signals for downstream analysis, combining automated scanning with interactive controls.

- **Detectors:** Slow-scanning detectors sweep bands with configurable dwell and thresholds, reporting frequency, power, and time to the Dashboard. Fixed-frequency detectors open a GNU Radio plot where center frequency, gain, and thresholds are adjustable; results stream in real time for visual and numeric confirmation.
- **Signal Conditioning:** Captured IQ can be segmented into per-transmission files and prepared via cropping, splitting (e.g., preamble/energy based), normalization, and resampling. Outputs include per-file statistics (e.g., duration, bandwidth, SNR proxies) to guide feature extraction, classification, and protocol discovery. Live SDR-side conditioning is planned.

9.3 Feature Extraction and Classification

After detection and conditioning, FISSURE converts captures into structured representations and categories. The Feature Extractor organizes calculators that turn IQ into numeric descriptors suitable for learning and search. Batches can be queued, and run parameters are recorded so results are comparable and reproducible. The Classifier evaluates features with saved models and reports predictions with confidence, supporting quick checks and deeper studies.

- **Feature Sets:** Time-domain statistics, spectral descriptors, cyclostationary or higher-order summaries (where applicable), and modulation/constellation descriptors for common RF classes.
- **Batch Processing & Reproducibility:** Queued jobs process multiple IQ files with recorded parameters (window length, step, FFT size, detectors used) to enable consistent reruns and comparisons.
- **Presets & Profiles:** Quick-select presets align calculators to saved models; named profiles capture choices and parameters for repeatable workflows.
- **Outputs & Preview:** Per-file feature tables with optional lightweight previews (for example, PSD snapshots or histograms) to sanity-check distributions before classification.

- **Classification:** Evaluation against saved scikit-learn or TensorFlow models with per-sample prediction, confidence scores, top-k candidates, and adjustable decision thresholds.
- **Model Comparison & Voting:** Side-by-side model evaluation with optional voting across models or feature sets to improve robustness on ambiguous signals.
- **Training:** In-tab training from labeled datasets with common options (train/validation splits, cross-validation) and reporting of accuracy and confusion matrices.
- **Model Registry & Versioning:** Models saved with metadata (name, version, required feature-set signature, training metrics, creation time) and compatibility checks before inference or retraining.
- **SOI Aggregation:** Results from conditioning, feature extraction, and classification can be merged into Signals of Interest with time, frequency, and classification summaries; deeper database linkage is in progress.

9.4 Protocol Discovery

Protocol Discovery supports moving from raw signal captures to structured message understanding. Current tools focus on known protocols while unknown-signal workflows continue to mature.

- **Demodulation Flow Graphs:** For known protocols, demodulation flow graphs stream captured data into a circular buffer so raw bitstreams can be examined and exported to downstream utilities.
- **Analysis Tools:** Preamble-based message splitting for IQ or bitstreams; entropy plotting to highlight structured versus random regions; and a bit viewer to shift, encode/decode, transform sequences, convert to hex or ASCII, and compare against known entries or other captures.
- **Wireshark Integration:** Sniffer flow graphs can stream live decoded traffic into Wireshark. Custom Lua dissectors enable visualization and inspection of new message types within familiar packet-analysis views.
- **CRC Tools:** Built-in calculators support common algorithms, seeds, and polynomials. A reverse-lookup mode derives polynomial parameters from known message/CRC pairs to aid unfamiliar protocol decoding.

Planned enhancements include recursive demodulation, improved logging, and additional workflows for identifying and documenting previously unknown protocols.

9.5 Attacks and Fuzzing

FISSURE provides a structured environment for creating and running wireless attacks, from simple transmissions to parameter sweeps and protocol field variations. Tools support experimentation, vulnerability research, and demonstration of effects.

- **Single-Stage Attacks:** A Python script or GNU Radio flow graph performs one action (for example, transmit a crafted packet or adjust a signal parameter). Defaults can be edited before launch, and selected variables may be changed at runtime while the attack is active.
- **Multi-Stage Attacks:** Chains of single-stage attacks executed in sequence. Each stage runs for a defined duration before handing off to the next, enabling compound effects to be orchestrated and saved for reuse.
- **Fuzzing:** Two styles are supported. Protocol field-level fuzzing varies selected fields sequentially or randomly and updates CRC fields automatically. Flow-graph variable fuzzing drives internal parameters (sequentially or randomly) to stress physical-layer behaviors.
- **Triggers:** Attacks can be tied to trigger conditions. Available triggers span acoustic, environmental, filesystem, networking, RF, time, and visual categories. Multiple triggers may be assigned; execution begins when any one condition succeeds and remaining triggers are canceled.
- **Reporting:** Runs can produce alerts, recommendations, and reports. Results appear in the Dashboard and can be forwarded to TAK for operational workflows, supporting immediate action or later analysis.

9.6 IQ Data Tools

IQ Data Tools provide a focused workspace to capture, manipulate, analyze, and replay IQ, bridging raw signal work to classification, protocol discovery, and effects.

- **Recording and Playback:** Signals can be recorded at a tactical node and saved as IQ files. Files are transferred to the hub for plotting in the Dashboard. Playback pushes the file back to a node for on-air transmission. Archive replay playlists queue multiple files with repeat and duration options.
- **Inspection Flow Graphs:** Quick-look GNU Radio flow graphs visualize captured data to provide awareness without full processing or protocol discovery.
- **Manipulation Tools:** Crop, append, split, normalize, resample, convert data types and formats, and swap endianness. Datasets can be reorganized for different applications.
- **Visualization:** An IQ viewer supports spectrograms, FFTs, AM/FM demodulation, phase analysis, and Morse code detection to examine portions of files.
- **Third-Party Tools:** One-click launch to external programs such as Gqrx, Inspectrum, and IQEngine for deeper analysis.
- **Online IQ Archive:** A curated archive of IQ collections recorded with various devices and sample rates. Each collection includes SigMF metadata at collection and file levels. Files can be previewed, favorited, downloaded individually or as collections, and added to playlists to simulate devices and environments. Additional online archives can be selected when available and integrated into the GUI. The archive operates independently of the library.
- **Dataset Builder:** Prepares training and evaluation sets from local IQ files. Applies controlled randomizations such as amplitude scaling, phase shift, and noise. Truth can be assigned per file or table row. Exports CSVs that pair file references with labels and parameters for model training. The Dataset Builder does not write into the library.

9.7 Sensor Node Features

Tactical nodes extend FISSURE to the edge. They host radios and sensors, execute playlists locally, and coordinate with the hub over IP or Meshtastic. Nodes support unattended operation from boot, react to triggers in the field, and report status, alerts, and position to the Dashboard.

- **Autorun Playlists:** Playlists composed of single-stage and multi-stage attacks. Triggers may be applied to items or to the playlist. Global and per-item delays, timeouts, and repetition intervals are supported. Playlists can be exported, imported, or saved over the default on a node.
- **Autonomous Operation:** Nodes can run an autorun playlist automatically at boot, enabling field use without continuous operator interaction, including scenarios with intermittent connectivity.
- **File Navigation:** The Dashboard transfers files between the hub and a node for workflows such as retrieving IQ captures or pushing datasets and scripts.
- **Alerts and GPS Beacons:** Nodes generate alerts during runs and send them to the Dashboard or forward them to a TAK server. Optional GPS beacons report node location using a receiver or Meshtastic-provided GPS.

9.8 Library and Database Tools

FISSURE uses a PostgreSQL database to manage signals, protocols, attacks, playlists, models, and other supporting information. Several tools are provided in the Dashboard to help users interact with this library and maintain their datasets.

- **Library Viewer:** Browse tables from within the Dashboard and remove entries when needed.
- **Search:** Find signals of interest or known protocols across stored records; optionally consult a companion spreadsheet of popular signals for quick lookups.
- **Plugin Management:** Build and import plugins from the Dashboard with metadata recorded in the database. Authoring features are being transitioned to a standalone plugin builder to simplify creation without a full install.
- **Image Gallery:** View saved images generated during analysis or operations.
- **Logs:** Inspect and filter rotating text logs to monitor local activity from the GUI.
- **Advanced Access:** Connect with external tools such as pgAdmin4 for direct queries and schema inspection when deeper research or debugging is required.

9.9 Menus and Extras

Menus and supplemental utilities support everyday use of the Dashboard. They provide quick starts, references, lessons, and experimental features alongside the core tabs. These items are optional, but they reduce setup time and help teams learn, demonstrate, and troubleshoot.

- **Options Menu:** Save presets for startup behavior. Configurations can be remembered between sessions or reset to defaults.
- **Standalone Flow Graphs:** Launch frequently used GNU Radio flow graphs directly from the Dashboard for quick demonstrations and common tasks.
- **Lessons:** Built-in modules covering RF, cybersecurity, and related topics connect FISSURE functions to instructional objectives.
- **Reference Tools:** Calculators, protocol references, and curated links provide quick access to supporting information.
- **Demo Menus:** Experimental automation features can be exercised for preview and feedback.
- **Help Menu:** Links are provided to the user manual and other documentation.

9.10 Current Limitations and Roadmap Connections

While FISSURE already provides a wide range of functionality, many workflows are still evolving. Current capabilities are mostly manual and tab-specific, and ongoing development is focused on integrating these into automated pipelines and scaling deployments.

- **Manual Workflows:** Most operations are initiated within individual Dashboard tabs. End-to-end orchestration across tabs is not yet integrated.
- **Low-Throughput Links:** Meshtastic-connected nodes currently support autorun playlists. Interactive Dashboard control over low-bandwidth links is limited; queued actions and store-and-forward are planned.
- **Results Integration:** Signals of Interest are not yet fully linked into the PostgreSQL database. Richer provenance, queries, and aggregation are in progress.
- **Protocol Discovery Coverage:** Tooling favors known protocols today. Recursive demodulation, improved logging, and unknown-signal workflows are under development.
- **Access Control and Security:** Once installed, access is broadly open. Role-based access control, plugin signing, and approvals and audit workflows are roadmap items.
- **Multi-User and Concurrency:** A single Dashboard controls the hub. Concurrent Dashboards and shared sessions will be introduced incrementally, with TAK integration as an early coordination path.

10. ROADMAP AND FUTURE DIRECTIONS

FISSURE is an active project with a clear path forward. The current framework supports a wide range of manual workflows and experiments, and the roadmap focuses on scaling, automation, and integration so those capabilities extend to coordinated, policy-aware operations.

Future development connects individual features into end-to-end pipelines, improves coordination across nodes, and deepens integration with TAK and mobile applications. Usability, documentation, and community growth remain priorities so the framework stays approachable for students and researchers while meeting operational needs.

This section presents the guiding principles, areas of planned development, and the long-term vision for how FISSURE will evolve across RF reverse engineering, cyber experimentation, and automated spectrum operations.

10.1 Guiding Principles

The roadmap is anchored by principles that keep development useful across audiences, sustainable to maintain, and consistent with the framework's mission.

- **Customer-Driven Development:** Priorities reflect the needs of operators, researchers, and adopting organizations.

Funded work and operational requirements lead, with improvements shared across the broader community.

- **Dual-Use Accessibility:** Every addition serves both professional and educational users. Capabilities valuable to Special Operations also translate to classrooms and labs, keeping the framework open, accessible, and adaptable.
- **Scalability:** Growth from single-laptop setups to distributed multi-node deployments remains a baseline requirement. Planned improvements emphasize coordination, resource sharing, and flexible deployment models.
- **Sustainability:** Modularity and maintainability guide design choices. Reliance on open-source tools, plugins, and containerization avoids vendor lock-in and supports long-term viability.
- **Community Growth:** Documentation, lessons, and outreach cultivate contributors from hobbyists and students to researchers and operators, strengthening the ecosystem over time.

Together, these principles keep FISSURE focused on unifying RF reverse engineering, spectrum operations, and cyber experimentation while remaining approachable and durable.

10.2 Automation and Modes of Operation

FISSURE is moving from manual, tab-specific workflows toward coordinated automation. The goal is for nodes and the hub to execute end-to-end tasks with minimal oversight while keeping operators in the loop for control, review, and intervention.

- **Automation Pipelines:** Connect detection, conditioning, feature extraction, classification, and protocol discovery into end-to-end workflows. Pipelines progress through stages automatically and record results for immediate action and later analysis.
- **Modes of Operation:** Introduce named modes for tactical nodes such as survey, monitor-and-alert, and pursue-and-collect. Each mode defines predefined behaviors that can run autonomously at the edge.
- **Triggering & Switching:** Switch modes or launch actions based on triggers from the hub, operator input, environmental events, scheduled conditions, or detected signals.
- **Store-and-Forward:** Queue results, logs, and artifacts locally during outages, then forward automatically when connectivity to the hub is restored.
- **Central Coordination at the Hub:** Keep orchestration, policies, and future decision logic at the hub so multi-node operations scale cleanly. The hub assigns tasks, resolves conflicts, and balances workloads across nodes and processing engines.
- **Client Roles:** The Dashboard remains the primary client for initiating pipelines and reviewing results. TAK plugins and mobile apps surface alerts, status, and basic tasking within hub policy. If links drop, nodes continue locally and synchronize state and artifacts upon reconnection.

10.3 Scaling and Coordination

FISSURE expands from single-machine experiments to coordinated deployments. The hub orchestrates tasks across nodes and processing engines while the Dashboard, TAK, and mobile apps provide tasking and visibility. Nodes continue operating locally when links are unreliable and synchronize state when connectivity returns.

- **Multi-Node Coordination:** The hub assigns tasks across nodes, manages spectrum sharing, balances workloads, and directs nodes to survey, monitor, or exploit different parts of the spectrum in parallel.
- **Deployment Scaling:** Architected to grow from small labs to field fleets. One hub may manage a few nodes or dozens across vehicles, aircraft, and fixed sites. Cloud and edge options allow hubs and databases to run centrally while nodes collect and act at the edge.
- **Smarter Coordination:** Roadmap items include AI-assisted tasking, automatic reassignment when nodes are unavailable, and cross-node collaboration for direction finding and distributed geolocation.
- **Multi-User Support:** Multiple Dashboards and remote users are planned, governed by role-based access control. TAK and mobile clients provide lightweight access for alerting, status, and basic tasking within hub policy.
- **Separation of Concerns:** Coordination and decision logic reside at the hub. The Dashboard focuses on tasking and visualization. Nodes execute locally and remain autonomous when disconnected, then reconcile results and logs upon reconnection.

10.4 Integration with Existing Solutions

FISSURE is designed to plug into tools and systems already in use so teams can extend existing workflows rather than replace them. Integration work focuses on operational interoperability, lightweight field access, multi-sensor inputs, and portable deployment.

- **TAK Integration:** The framework bridges to TAK servers so alerts and detections appear in ATAK, WinTAK, and WebTAK. Planned expansions include publishing protocol discoveries, Signals of Interest updates, and structured reports, with dedicated TAK plugins for tasking and review within hub policy.
- **Mobile Applications:** Lightweight apps provide basic tasking, status monitoring, and alert visualization on field devices. They operate within the same network domain as the hub to extend access when a full Dashboard is not available.
- **Additional Sensors:** Support is planned for adjunct sensors beyond software-defined radios where Python, drivers, and operating system support exist, enabling richer multi-domain fusion without changing the core architecture.
- **Containerized Deployment:** Packaging is expanding beyond the database to include additional components with Docker or Apptainer, enabling consistent deployments across cloud, edge, and tactical environments.

10.5 Community and Ecosystem Growth

FISSURE's long-term value depends on a healthy ecosystem as much as new features. The roadmap emphasizes openness, education, and steady contributor growth across students, researchers, hobbyists, and operators.

- **Open-Source Presence:** Source code, issues, and contributions are managed in public repositories with visible milestones and release notes to keep progress transparent.

- **Educational Outreach:** Lessons, labs, workshops, and recorded demos help classrooms and training programs adopt the framework; an ongoing CTF invites community-contributed challenges.
- **Diverse User Base:** Onboarding paths are shaped for beginners while maintaining depth for research and operations, with clear avenues for plugin authors and data contributors.
- **Documentation and Tutorials:** Ongoing investment in user guides, quickstarts, examples, and architecture notes shortens time to proficiency and supports advanced usage.
- **Public Roadmap and White Papers:** Progress and priorities remain visible through living roadmaps, with periodic white papers highlighting use cases, capabilities, and lessons learned.

10.6 Framework Evolution

As capabilities expand, the architecture will evolve to support more advanced workflows, stronger resilience, and tighter integration with adjacent technologies. The focus remains modular components, clear separation of concerns, and consistent behavior from single-machine setups to fleet deployments.

- **AI and Machine Learning Integration:** Tighter placement of AI and ML in the hub and processing engines to enable automated classification, smarter coordination across nodes, and adaptive responses to emerging signals.
- **Protocol Discovery Overhaul:** Recursive demodulation and improved unknown-signal workflows, with greater automation from raw IQ to structured protocol definitions and clearer reporting.
- **Direction Finding and Tracking:** New features for direction finding, emitter geolocation, and multi-node triangulation to support both research and field operations.
- **Expanded Fuzzing and Vulnerability Analysis:** Broader coverage across signal types and deeper protocol layers, with methods that better stress wireless systems and surface weaknesses.
- **Policy-Aware Messaging and Provenance:** Controls to enforce operational policies, track data lineage, and maintain audit trails for research and operational use.
- **Operational Telemetry:** Aggregation at the hub to summarize fleet health, node utilization, and mode effectiveness, with telemetry views in the Dashboard for quick assessment.

10.7 Long-Term Vision

FISSURE aims to be the open, community-governed framework for spectrum operations, protocol exploration, and cyber experimentation. The focus is a collaborative, policy-aware system that scales from single machines to coordinated fleets while remaining approachable for education and research.

- **Collaborative Hub:** A shared space where signals, protocols, workflows, and lessons are documented, exchanged, and extended. Community-built plugins and curated libraries allow knowledge to accumulate over time.
- **Dual-Use Growth:** Capabilities are shaped for professional operations and education alike, balancing scalable, policy-aware features with approachable learning paths for classrooms and labs.
- **Sensor Fusion and Automation:** Expansion beyond RF to include acoustic, visual, cyber, and other sensors. Automation across inputs enables adaptive behavior and coordinated, cross-domain operations.
- **Sustainable Development:** A hybrid model of open-source contributions, professional services, and integration into funded projects keeps progress steady without losing openness or extensibility.

Taken together, these directions shift FISSURE from capable manual tools to an orchestrated, policy-aware framework. It remains open and teachable for classrooms and research while adding the automation, security, and resilience required for mission use. The outcome is a collaborative system that fuses RF, cyber, and other sensors into repeatable workflows that organizations can adopt and extend with confidence.

11. EXAMPLE USE CASES

FISSURE's modular design and COTS-first approach allow the same framework to support classrooms, research labs, and field deployments. This section presents representative scenarios across fixed sites, vehicles, aerial platforms, maritime environments, counter-UAS missions, TAK and mobile integrations, education, cyber-spectrum convergence, automation with AI/ML, and community extensions. The list is illustrative, not exhaustive, and will evolve as adoption grows.

11.1 Baseline Workflow

FISSURE is a general-purpose framework for RF reverse engineering, cyber experimentation, and distributed sensing. It unifies detection, conditioning, feature extraction, classification, protocol discovery, and attack execution under a shared library and the hub for coordination. The same workflow scales from laptops to multi-node deployments without changing tools.

- **Operational Example:** An operator detects an unfamiliar signal, runs feature extraction and classification, and matches it to a known protocol in the library. Results are sent through the hub, with alerts and reports forwarded to TAK for situational awareness.
- **Research and Training Example:** The workflow is replicated in a classroom. Students capture signals with low-cost software-defined radios, run classifiers, and compare findings against the shared library, reinforcing theory with hands-on practice.

This common path keeps research, education, and operations aligned on methods and artifacts.

11.2 Perimeter and Infrastructure Defense

Fixed sites such as bases, airports, utilities, campuses, and data centers benefit from persistent spectrum monitoring. FISSURE combines detection, classification, and protocol discovery, records Signals of Interest, and forwards alerts through the hub to TAK when enabled. Baselines can be established per site so anomalies stand out during patrols and audits.

- **Operational Example:** A node at an airport perimeter flags a rogue or look-alike Wi-Fi access point on an unusual channel. The signal is classified, logged as a Signal of Interest with time and location, and an alert is published to TAK for shared awareness and response by security teams.
- **Research and Training Example:** A university lab installs a fixed node to observe campus protocols. Students

configure detectors, extract features, and classify transmissions while simulating unauthorized radios or spoofed beacons to practice triage and reporting.

Applied at fixed facilities, outputs drive perimeter RF baselines, watch lists, and SOI timelines, and support artifact-backed reports for security, safety, and spectrum management teams.

11.3 Vehicle and Mobility Systems

Vehicles extend coverage beyond fixed sites and let teams sense, record, and act while in motion. A tactical node mounted in a vehicle collects geotagged IQ, classifies activity along a route, and exchanges alerts and results with the hub over IP or Meshtastic when links are available.

- **Operational Example:** A convoy support vehicle runs a tactical node during transit. Detectors scan priority bands, classifications mark Signals of Interest with time and GPS, and alerts are forwarded to the hub and TAK when connectivity is present. Route heat maps and SOI timelines provide a record of activity for post-mission review and for planning return passes.
- **Research and Training Example:** A lab team outfits a vehicle with a low-cost software-defined radio and a tactical node to conduct wardriving-style surveys. Captures of Wi-Fi, Bluetooth, and other common protocols are organized by location, then analyzed in the Dashboard to compare neighborhoods, assess spectrum usage near facilities, and build teaching datasets.

Applied on vehicles, outputs support convoy protection, RF leakage audits near sensitive sites, and pop-up monitoring during short stops. Over time this can lead to coordinated multi-vehicle surveys with direction finding and geolocation, pursue-and-collect modes that react to detections, and automated route overlays in TAK for patrol planning and after-action analysis.

11.4 Drone Payloads and Aerial Operations

Aerial nodes extend coverage, improve line of sight, and access terrain that is hard or unsafe to reach from the ground. A lightweight payload with a tactical node, software-defined radio, GPS, and a link to the hub records IQ in flight, classifies activity, and synchronizes detections and artifacts when connectivity is available.

- Operational Example: An sUAS flies over dense urban blocks where ground sensors struggle with blockage. From altitude it collects cleaner RF, identifies active transmitters, logs Signals of Interest with time and position, and returns detections to the hub after landing or via a relay during flight. The hub tasks a follow-up pass to refine location or capture longer samples.
- Research and Training Example: A test team mounts a low-power node on a small drone to study altitude, antenna placement, and polarization effects. Flights record IQ over campus and industrial areas, then the Dashboard is used to compare coverage maps, spectrum occupancy, and classification accuracy across altitudes.

Applied in the air domain, outputs support rooftop surveys, disaster communications mapping, perimeter overwatch, and approach sweeps to ports or facilities. Future work can add coordinated multi-drone surveys with direction finding and geolocation, pursue-and-collect modes that loiter on detections, airborne store-and-forward relays for low-throughput links, and hub-planned flight tasks tied to TAK map overlays.

11.5 Maritime and Port Protection

Ports, harbors, and vessels benefit from persistent spectrum awareness to spot unauthorized transmitters, spoofed identifiers, and interference that can disrupt safety and operations. FISSURE nodes positioned waterside and afloat monitor AIS, marine VHF, radar-adjacent bands, and local control links, recording IQ, classifying activity, and logging Signals of Interest. Alerts and summaries flow to the hub and, when enabled, to TAK for shared awareness with port security.

- **Operational Example:** Shore nodes at a container terminal and a node aboard a harbor patrol boat detect an AIS transmission with a mismatched MMSI pattern and unusual timing. The signal is classified, logged with time and position, and an alert is sent through the hub to TAK. Follow-on captures include short IQ snippets and decoded fields for review during incident response.
- **Research and Training Example:** A university team installs a pier-side node to study maritime channels. Students record AIS and marine VHF traffic, compare occupancy during shift changes and storms, and use protocol discovery tools to examine AIS message fields. Simulated spoofed beacons and unauthorized handheld transmissions provide practice in triage and reporting.

Applied on the water, outputs support berth-clearance checks, restricted-area monitoring, and radio interference audits tied to vessel movements and port operations. Future work can add coordinated shore-and-afloat direction finding and geolocation, anomaly detection on AIS and port-control links, and fusion with radar tracks and cameras for faster investigation and enforcement.

11.6 Counter-UAS Applications

Small drones create spectrum signatures that can be detected, classified, and trended for defense of bases, events, and critical sites. FISSURE nodes monitor likely control and video links, record IQ, label Signals of Interest, and publish alerts through the hub to TAK for shared response.

- **Operational Example:** A perimeter node observes activity in known small-UAS control and telemetry bands during a VIP movement. Detectors flag bursts consistent with a common C2 waveform, classification confirms a likely match, and the hub issues an alert to TAK with time, frequency, and location. Follow-on tasks capture short IQ snippets and decoded fields for rapid triage.
- **Research and Training Example:** A test range team stages flights with several hobby and commercial sUAS. Nodes record IQ and metadata, students compare link behaviors at different ranges and antenna geometries, and protocol tools examine framing and identifiers. Results build a labeled dataset for future classifiers and drills.

Applied to C-UAS, outputs support early warning, trend lines for recurring activity, and artifact-backed incident reports. Future work can add coordinated direction finding and geolocation across nodes, pursue-and-collect modes that react to detections, correlation with Remote ID where available, and policy-aware effect playbooks that align detection, tracking, and approved countermeasures.

11.7 TAK and Mobile Integration

TAK and mobile clients extend FISSURE into field workflows. Alerts, status, and basic tasking flow through the hub so teams on ATAK, WinTAK, or WebTAK can act without switching tools. Mobile access focuses on quick checks and simple actions rather than full Dashboard control.

- **Operational Example:** A patrol team on ATAK receives an RF alert with time, frequency, location, and a link to the related Signal of Interest. With a TAK plugin, the team tasks a nearby node to capture a short IQ snippet and post results back to the hub. The alert thread in TAK updates with the capture summary and a thumbnail for review.
- **Research and Training Example:** During a field lab, students use a lightweight mobile client to view node status, acknowledge alerts, and start a detector on a preset band. Instructors monitor progress from the Dashboard while students operate from phones and tablets.

Applied on mobile devices, outputs support patrol alerts, quick-look captures, and on-site triage. Future work can add queued tasks for offline operations, richer map overlays and modes, and policy-aware push notifications that coordinate with the hub.

11.8 Education and Training

Classrooms, training units, and labs can apply FISSURE to teach spectrum operations with real signals and reproducible workflows. A modular Dashboard, low-cost software-defined radios, and IQ Data Tools turn theory into practice while preserving artifacts for assessment and reuse.

- **Operational Example:** Training units run detector, conditioning, and classification labs that mirror field workflows. Results are logged as Signals of Interest with time, frequency, and location, and summaries can be forwarded through the hub to TAK for capstone exercises.
- **Research and Training Example:** A university course records IQ with inexpensive radios, extracts features, trains baseline models, and compares predictions to a shared library. Students submit CSVs, plots, and PCAPs as graded artifacts, building a reusable dataset across semesters.

Applied to education, outputs support standardized lab packs, curriculum-aligned assessments, and a growing repository of labeled IQ for future classes. Future work can add instructor dashboards for class progress, remote lab nodes for distance learning, and shared lesson bundles that integrate the Online IQ Archive with step-by-step activities.

11.9 Cyber and Spectrum Convergence

Wireless links often anchor larger systems. FISSURE highlights this connection by letting teams move from RF captures to protocol understanding and into cyber workflows without leaving the framework.

- **Operational Example:** A node records traffic from a target device. After demodulation and classification, decoded fields are exported to packet analysis tools and matched against known weaknesses. The hub packages an alert with the Signal of Interest, decoded summaries, and recommended follow-on actions for a controlled exploitation test.
- **Research and Training Example:** A lab exercise collects traffic from an IoT sensor, inspects framing with bit-level tools, and then pivots to PCAP-based analysis to explore authentication and replay behavior. Students compare outcomes across devices and document conditions that lead to failures.

Applied at this boundary, outputs support protocol hardening, red-blue validation of wireless entry points, and artifact-backed reports that trace findings from IQ to packets and application impact. Future work can add automated pivots from detections to replay or fuzzing tasks, plus curated playbooks that tie spectrum observations to cyber test procedures.

11.10 Automation and AI/ML in RF

Automation centers on using trained models to triage detections, route unknowns, and recommend actions across nodes and the hub. Current workflows support training and applying classifiers; the roadmap deepens placement of models in coordination paths so results arrive with confidence, provenance, and next-step suggestions.

- **Operational Example:** The hub runs a coordinated mission with two ground nodes and three aerial nodes in a survey mode. Aerial Node 1 flags bursts consistent with a priority emitter. The hub raises confidence with a saved model, switches Aerial Node 1 to pursue-and-collect, tasks Aerial Node 2 to cross-cut the area for geometry, and shifts a ground node to monitor-and-alert on adjacent bands. Short IQ clips and decoded fields flow back; if a link drops, nodes queue artifacts and sync on reconnect. TAK receives an alert thread that updates as tasks complete with time, frequency, location, and confidence.
- **Research and Training Example:** A lab cohort records IQ with low-cost software-defined radios, extracts features, and trains baseline models in the Dashboard. The team then replays held-out captures to validate accuracy, studies failure cases, and exports labeled CSVs and PCAPs to refine future models.

Applied to RF automation, outputs accelerate triage, prioritize scarce link time, and create artifacts suitable for audit and retraining. Future work adds fleet-level pipelines at the hub, model-drift monitoring, anomaly routing into protocol discovery and fuzzing tasks, and active-learning loops that request targeted labels when confidence is low.

11.11 Contributing and Extending FISSURE

FISSURE grows through contributions that add plugins, dissectors, lessons, and datasets. The framework's catalogs and registries track plugin metadata and versions so extensions can be deployed in labs or operational environments without changing core code.

- **Operational Example:** A defense contractor develops a private plugin to analyze a proprietary radio protocol used in field equipment. The package is imported with version and metadata recorded in the catalog. Operators detect and interpret the protocol alongside existing workflows, and updates roll out as new plugin versions are released.
- **Research and Training Example:** Senior capstone teams adopt FISSURE to deliver a complete plugin and dataset. Each team selects a protocol or RF task, collects IQ with low-cost software-defined radios or from the Online IQ Archive, annotates files with SigMF, and uses IQ Data Tools and the Dataset Builder to produce labeled CSVs. Deliverables include a versioned plugin with catalog

metadata, a labeled dataset, a short lesson or README, and a demo-day runbook. Faculty can retain the plugin and dataset in the library for future cohorts.

Applied across teams, contributions build a practical catalog of reusable capabilities, lesson packs, and datasets that shorten onboarding and enable repeatable studies. Future work includes plugin signing and execution policy, stronger version guarantees for interfaces, and curated catalogs that highlight well-maintained extensions.

12. CLOSING AND NEXT STEPS

This paper outlined FISSURE's purpose, architecture, and path forward. The framework brings spectrum analysis, protocol exploration, cyber experimentation, and distributed sensing into one adaptable system that serves operators, researchers, educators, and hobbyists.

This section closes with a concise summary of value, ways to engage, collaboration pathways, roadmap continuity, and concrete next steps. The aim is to turn interest into action while keeping the framework open, teachable, and ready for operational use.

12.1 Summary of FISSURE's Value

FISSURE is an open-source, plugin-driven framework that unifies RF reverse engineering, cyber experimentation, and distributed sensing in one adaptable environment. A COTS-first design, graphical installer, and modular components lower barriers to entry while supporting workflows that scale from a single laptop to fleets of tactical nodes. Results, artifacts, and alerts flow through the hub and can integrate with TAK for shared awareness.

- **Operators:** Detect, classify, and analyze signals; explore protocols; execute single- and multi-stage attacks; publish alerts and summaries for team awareness through TAK.
- **Researchers and Educators:** Run transparent, reproducible studies; build and compare models; use IQ Data Tools and the Dataset Builder to create labeled datasets and lesson materials.
- **Hobbyists:** Learn spectrum operations with affordable software-defined radios; extend capabilities through plugins, presets, and shared examples.

FISSURE adapts to teaching, research, and operations without changing tools, providing a common foundation that grows with contributions from its community.

12.2 Invitation to Engage

FISSURE grows through participation. Engagement turns interest into feedback, extensions, and shared artifacts that improve the framework for education, research, and operations.

- **Repository Access:** Source code, issues, milestones, and documentation are available on GitHub.
- **Local Evaluation:** Install the framework and run with local software-defined radios or sample datasets.
- **Contributions:** Add plugins, lessons, or datasets and improve documentation through pull requests or shared packages.
- **Community Channels:** Join discussions on GitHub Discussions, Discord, or social media to share ideas and results.
- **Feedback and Roadmap Input:** File issues, propose features, and report usage to inform prioritization.

Active engagement strengthens testing coverage, expands datasets, and keeps development aligned with real scenarios.

12.3 Collaboration Pathways

FISSURE provides multiple ways to collaborate, reflecting its role as both an open-source framework and a system that supports specialized projects.

- **Open-Source Contributions:** Build plugins, improve documentation, create tutorials, and share datasets. Contributions follow versioning and licensing guidance, with metadata recorded in catalogs for reuse.
- **Academic Collaboration:** Adopt FISSURE for courses, labs, and capstone projects. Typical outputs include lesson packs, lab rubrics, plugins, and labeled datasets that faculty can retain in the library for future cohorts.
- **Government and DoD Engagement:** Pursue integration pilots, evaluations, and exercises that align with operational needs and data-handling requirements. Deliverables often include demo nodes, plugin catalogs, test reports, and TAK-integrated workflows.
- **Industry Partnerships:** Integrate sensors, radios, and adjunct tools, or use FISSURE as a testbed for new capabilities. Collaboration can provide interface specifications, plugin SDKs, container images, and CI-tested builds for consistent deployment.

These pathways convert research outputs, field lessons, and product integrations into versioned plugins, datasets, and documentation that remain useful across releases.

12.4 Roadmap Continuity

FISSURE is a living framework with a public roadmap that tracks near-term deliverables and long-term goals. Priorities include automation, multi-node coordination, protocol discovery improvements, and integrations with TAK and mobile clients. Progress is visualized in Plotly dashboards aligned to the technology hierarchy; a sanitized view is linked from the GitHub README, while customer-specific details remain internal.

Future white papers will expand focused applications such as counter-UAS, maritime operations, mobile integration, and cyber-spectrum convergence. This architecture paper serves as the baseline and will be revised as features mature and operational lessons accumulate.

12.5 Next Steps for Readers

FISSURE offers clear paths from interest to action. The items below outline concrete first steps for different audiences.

- **Operators:** Schedule a demo with AIS; identify a pilot use case; deploy a controlling setup with one tactical node; integrate alerts with an existing TAK server; capture a week of activity and review Signals of Interest with stakeholders.
- **Researchers and Educators:** Review documentation and quickstarts; stand up a lab instance; adapt detector → conditioning → classification exercises to course objectives; package lesson materials and small labeled datasets for reuse; share outcomes and feedback to inform roadmap priorities.

- **Hobbyists and Students:** Install locally with a single radio; replicate quick-start workflows; experiment with example plugins, third-party tools and reference material (for example, Gqrx, Inspectrum, IQEngine); contribute a small dataset with SigMF metadata; submit a challenge to the ongoing CTF.
- **All Readers:** Track the public roadmap; file issues and feature requests; share findings and artifacts in community channels; follow release tags and upgrade notes to stay aligned with interfaces and capabilities.

12.6 Closing Note

FISSURE is a framework, not a single-purpose tool. It unifies spectrum analysis, protocol exploration, cyber experimentation, and distributed sensing in an open, adaptable system. Strength comes from an active community and from deployments that inform design choices.

This paper captures the architecture and technical foundations as they stand today and outlines directions that guide near-term work. As features mature and lessons accumulate, updated editions will keep the baseline current.

The project remains open to collaboration. Operators, researchers, educators, and hobbyists contribute artifacts, datasets, and plugins that move the framework forward. With continued contributions and disciplined engineering, FISSURE will remain a durable foundation for classroom learning, research, and mission use.