

iwchaos: Architecture Blueprint

Project: Memory-Safe, Formally Verified, and Chaos-Driven Network Initialization

Lead Architect: Kenny Glowner

Target Hardware: Lenovo ThinkPad P53 (Intel Wi-Fi Chipset, tuned-rs Power Management)

Host Environment: CIQ RLC Pro (Standard Kernel Build)

Init Target: RustD / rustd-resolved integration

This document outlines the architectural blueprint for `iwchaos`, a custom rewrite of the Linux Intel Wi-Fi driver combining memory safety, formal theorem proving, and non-linear chaotic dynamics to manage hardware interaction strictly in Ring 0, replacing the legacy `iwlmwifi` module.

1. The Multi-Language Stack

Deploying this stack into the Linux kernel requires stripping all runtimes and standard libraries. Each language serves a strictly defined, isolated purpose within the driver architecture.

- **Rust (RFL):** The core memory-safe manager. Handles PCIe bus initialization, DMA ring buffer allocation, and memory mapping. Acts as the central anchor for Kbuild.
- **C (Shims):** Minimalist translation layer. Provides the necessary ABI hooks into the entrenched `mac80211` and `cfg80211` kernel network structures.
- **Idris 2:** Leverages Quantitative Type Theory (QTT). Models the Intel firmware state machine and strictly enforces linear types (multiplicity of 1) to guarantee DMA buffers are consumed exactly once, eliminating use-after-free panics. Compiles down to freestanding C using the Idris C-backend with a stripped RTS.
- **Agda:** Utilized offline to formally prove wire and bound invariants for the networking protocols before generating verified algorithms.
- **Fortran:** The Chaos Engine. Compiles to freestanding object files (no `libgfortran`) to compute mathematically dense, deterministic chaotic functions for RF adaptation.

2. The Chaos Engine (Non-Linear RF Adaptation)

Standard Wi-Fi drivers rely on basic linear algorithms for power scaling and collision avoidance. **iwchaos** introduces deterministic chaos theory to navigate congested and unpredictable RF environments.

A. Lorenz Attractor MAC Backoff (CSMA/CA)

Instead of relying on standard binary exponential backoff when a packet collision occurs, the CSMA/CA state machine defers to a Lorenz system calculated by the Fortran module.

$$\begin{aligned}dx/dt &= \sigma(y - x) \\ dy/dt &= x(\rho - z) - y \\ dz/dt &= xy - \beta z\end{aligned}$$

The x coordinate is mapped directly to the microsecond delay. Highly sensitive to initial conditions, this generates a deterministic but wildly unpredictable backoff sequence, theoretically reducing repeated collisions in densely populated Wi-Fi channels.

B. Mandelbrot Power & SNR Scaling

Signal-to-Noise Ratio (SNR) and interface interference are modeled on the complex plane. The driver maps real-time signal degradation to the Mandelbrot equation:

$$z_{n+1} = z_n^2 + c$$

The escape time of the sequence dictates the hardware power state. If the sequence rapidly escapes (indicating high, chaotic interference), the driver seamlessly triggers tuned-rs aware power scaling or forces an aggressive channel hop.

3. The Ring 0 Build & Integration Pipeline

Linking these components into the standard kernel tree without standard libraries requires a highly tailored `Makefile` sequence within the Kbuild infrastructure.

- **Phase 1: Idris 2 Compilation** - The firmware interaction protocols and state machines are checked. Idris 2 generates pure, zero-allocation C files representing the verified logic.
- **Phase 2: Fortran Object Generation** - The Chaos Engine is compiled using `gfortran -ffreestanding -c`, bypassing `libgfortran` to produce raw `.o` files containing the pure mathematical subroutines.
- **Phase 3: The Rust/C Binding** - Rust's `bindgen` generates safe interfaces for the raw C shims bridging `mac80211`.
- **Phase 4: Kbuild Final Link** - The Linux Kbuild system ingests the Rust core, the Idris-generated C, the Fortran objects, and the C shims, linking them into the final `iwchaos.ko` loadable kernel module.

4. RustD / CIQ Architecture Alignment

As the ultimate replacement for legacy `iwlfwif`, this module naturally integrates with the `rustd` ecosystem. By utilizing native Netlink bindings passed through `rustd-udevd`, the network initialization process bypasses legacy `systemd-networkd` logic entirely, reporting interface readiness directly to `rustd-resolved`.