Real-Time RF Change Detection Pipeline for Wi-Fi Interference Identification

Team 15: Change Detection Team

November 2025

Contents

1. Detection Algorithms

We implemented a **Change Detection Pipeline** which is used to detect real-time anomalies in Wi-Fi RF telemetry. The pipeline was designed with three practical goals:

- **Low latency.** Detect relevant spectrum events within milliseconds–seconds so operators or automated controllers can react quickly.
- **Robustness to noise.** Wi-Fi measurements (especially CCA Busy%) are noisy and bursty; the pipeline suppresses transient spikes while remaining sensitive to sustained or structural changes.
- **Actionable, interpretable alerts.** Produce compact alerts (JSON) suitable for ingestion by orchestration/policy modules and for human review.

The pipeline applies a different sequential change detector to each RF metric (a *hybrid per-metric approach*). Individual detector outputs are fused using a lightweight voting + debounce + cooldown mechanism to reduce false positives and ensure alerts correspond to meaningful RF events. Baselines (pre-change statistics) are computed offline and hard-coded at runtime to avoid warmup delay in deployment.

This section explains each sequential change detection algorithm we use. Explanations are written for readers new to the area and include the math and a plain English intuition.

1.1 Exponentially Weighted Moving Average (EWMA)

Used for: CCA Busy%, Wi-Fi Airtime%.

Goal: Smooth noisy, bursty inputs while remaining responsive to sustained shifts.

Definition: the EWMA state Z_t updates recursively:

$$Z_t = \alpha X_t + (1 - \alpha) Z_{t-1}, \qquad 0 < \alpha \leq 1$$

where X_t is the new measurement and α controls responsiveness (large $\alpha \Rightarrow$ more responsive, smaller $\alpha \Rightarrow$ smoother).

Decision rule: Compare the smoothed value to a robust baseline μ_0 (computed offline) scaled by baseline spread σ_0 (MAD-based). Raise flag if

$$|Z_t - \mu_0| > L \cdot \sigma_0,$$

where L is a tuning constant (e.g., 3).

Intuition: EWMA acts like a low-pass filter: small, short spikes move Z_t little, but sustained increases/decreases push it steadily and trigger detection. This is ideal for metrics where short transmissions should not cause alerts, but sustained occupancy should.

1.2 Cumulative Sum (CUSUM)

Used for: SNR (dB).

Goal: Detect small but persistent shifts in the mean quickly (good for abrupt step changes).

Definition: Maintain two one-sided statistics for upward and downward shifts:

$$S_t^+ = \max\big(0,\, S_{t-1}^+ + (X_t - \mu_0 - k)\big),$$

$$S_t^- = \max\big(0,\, S_{t-1}^- + (\mu_0 - X_t - k)\big).$$

Here μ_0 is baseline center, k is a slack or reference value (commonly set to half the expected detectable shift). Trigger an alarm if

$$S_t^+ > h \quad \text{or} \quad S_t^- > h,$$

where h is the threshold.

Intuition: Each sample that supports a shift adds positive mass to the corresponding statistic; noise that oscillates tends to cancel and keep S near zero. After enough consistent evidence, S crosses h and signals a change. CUSUM is fast for step changes and has tunable false-alarm/detection tradeoffs via k, h.

1.3 Sequential Probability Ratio Test (SPRT)

Used for: Noise floor (dBm) in our pipeline.

Goal: Likelihood-ratio based test optimal (in the Neyman–Pearson sense) for distinguishing two known distributions f_0 (pre-change) and f_1 (post-change).

Definition: Maintain cumulative log-likelihood ratio:

$$\Lambda_t = \sum_{i=1}^{t} \log \frac{f_1(X_i)}{f_0(X_i)}.$$

Raise alarm if $\Lambda_t \geq \log B$ (upper threshold), or continue if below. Optionally a lower threshold can indicate no change.

Intuition: SPRT compares how probable the observations are under the post-change model vs the pre-change model. For noise floor, many interferer events are well modelled as a shift in mean or variance; SPRT can be made highly sensitive if f_0, f_1 are good fits.

Practical note: In practice we use robust parameter estimates for f_0 (baseline) and a pragmatic post-change model (e.g., shifted mean Gaussian) tuned from prototypes.

1.4 Page–Hinkley Test (PHT)

Used for: (not in the final mapping above but useful in experiments) — drift detection e.g., slow increases in busy fraction.

Definition: Track cumulative deviation from running mean:

$$S_t = S_{t-1} + (X_t - \mu_0 - \delta), \qquad m_t = \min(m_{t-1}, S_t).$$

Alarm if $S_t - m_t > \lambda$, where δ is small tolerance and λ a threshold.

Intuition: PHT is designed to detect gradual drifts: it remembers the minimum past cumulative deviation and signals when the current cumulative deviation exceeds that minimum by λ.

1.5 Robust Baseline Estimation (offline)

All detectors reference a precomputed baseline:

$$\mu_0 = \text{median}(X_{\text{clean}}), \qquad \sigma_0 = 1.4826 \cdot \text{MAD}(X_{\text{clean}}).$$

Median/MAD are robust to outliers. Baselines are computed offline on clean data and then hardcoded in production to avoid online warmup latency.

1.6 Mapping Algorithms to Metrics

We justify the choice of detector for each metric by combining domain knowledge of RF behavior with empirical noise characteristics:

- **CCA Busy% (EWMA)** — CCA Busy is bursty due to packet-level contention; EWMA smooths noise and detects sustained increases (congestion) while ignoring sub-frame jitter.
- **Wi-Fi Airtime% (EWMA)** — Airtime is directly tied to traffic; short bursts shouldn't trigger operator actions, while persistent high airtime should — EWMA is a natural match.
- **Noise Floor (SPRT)** — Noise floor shifts are often abrupt and can be well modelled as a change in distribution (mean increase). SPRT's likelihood ratio is well suited to quickly confirm such distributional changes with bounded false alarm rates when f_0, f_1 are reasonable.
- **SNR (CUSUM)** — SNR commonly exhibits step decreases (interferer onset, link degradation). CUSUM is designed for step detection with low delay and controllable false alarms.

This mapping is intentionally *logical* (based on algorithm properties) and not biased by specific experimental runs. Real deployments can adjust parameters (α, k, h, L) per radio class or SKU.

2. Fusion Engine

Individual detector outputs are binary signals (fire/not fire) per metric per frame. Fusion transforms these into stable, actionable system alerts using several modules described below.

2.1 K-of-N Voting

We use a K-of-N rule with $N = 4$ metrics and $K = 2$ (two metrics must fire in a short temporal window W to consider an alert).

Why? Single metric anomalies are often spurious. Requiring at least two independent metrics reduces false positives while still capturing multi-metric events (e.g., noise floor up and SNR down).

Implementation detail: maintain a sliding buffer of the last W seconds (e.g., $W = 3$ seconds) and count unique metrics that fired within this window.

2.2 Debounce

Once the K-of-N condition is true, require it to persist for D consecutive frames (e.g., $D = 2$) before emitting a confirmed alert.

Why? Debounce filters transient coincidences (two metrics firing in a single noisy frame) and only allows alerts when the anomalous state persists.

2.3 Cooldown

After emitting an alert, suppress new alerts for a cooldown period (e.g., 30 seconds). During cooldown, we may still collect detections and extend the current event window, but we do not emit duplicate alerts.

Why? Prevents alert storms for the same event and simplifies higher-layer state machines.

2.4 Event Stitching (Stitch small bursts into single events)

Detections within a short gap (e.g., 5–30 seconds depending on policy) are merged into a single event interval. This produces a cleaner timeline for operators and reduces alert duplication.

Why? In noisy RF, detectors may flicker leading to multiple separate alerts; stitching groups them into one event spanning from first fire to last fire plus small margins.

3. Hybrid Algorithm Integration

3.1 Rolling window smoothing

Before feeding a metric to its detector, a small rolling median (window 3–7 samples depending on metric frequency) is applied to remove outliers. This is cheap and reduces false firing of detectors that are sensitive to single extreme samples.

3.2 Handling different frame sizes / irregular arrivals

In real pipelines metrics can arrive at different cadences. We recommend:

- Treat each metric as an independent stream and maintain per-metric timestamps.
- When applying K-of-N voting use a time window W (seconds) instead of a frame count so metrics with different sampling rates still vote fairly.
- For missing samples, the detector continues with last state; avoid imputing unless reasonable (linear interpolation over small gaps).

3.3 Hardcoded baselines

To avoid startup/warmup cost and keep latency minimal, baselines (μ_0, σ_0 or f_0 params) are computed offline and provided to the runtime as constants. This allows immediate, correct detection on cold start.

3.4 Latency budget and where time is spent

Typical pipeline stages and their rough relative cost:

1. Ingestion parsing of a telemetry row: microseconds–ms (depends on transport).
2. Rolling median / lightweight transforms: microseconds.
3. Per-metric detector update (EWMA/CUSUM/SPRT): microseconds.
4. Voting debounce logic: microseconds.
5. Emission persistence of alert (write to file/DB/queue): ms (depends on storage).

To minimize latency: keep detectors O(1) per sample (they are), avoid expensive I/O on the critical path (buffer writes / async persistence), and precompute baselines.

4. Input and Output formats

4.1 Input Metrics

We monitor four per-interval (e.g., per second) RF metrics from each radio/channel:

CCA Busy% Fraction of the interval during which Clear Channel Assessment reports the medium busy (captures channel contention and overall occupancy).

Wi-Fi Airtime% Fraction of interval occupied specifically by Wi-Fi frames (gives insight into actual Wi-Fi traffic load).

Noise Floor (dBm) Measured baseline RF noise power at the radio front end (sensitive to external/interferer sources).

SNR (dB) Signal-to-noise ratio experienced by clients — drops indicate worsening link quality (interference, obstruction).

Important characteristics:

- *CCA Busy%* and *Wi-Fi Airtime%* are bounded fractions [0,1] and can be highly bursty (short transmissions, contention spikes).
- *Noise floor* often changes more smoothly but exhibits abrupt step shifts when a new continuous interferer appears.
- *SNR* can experience abrupt downward steps (e.g., strong interferer or client movement) and occasional jitter.

Because these metrics have different statistical and temporal behaviors, applying a single detector to all metrics produced suboptimal results in earlier experiments. The hybrid design maps detectors to metrics by those behaviors (explained next).

4.2 Input row (single telemetry event)

Each telemetry row is a JSON object like:

```
{
  "timestamp": "2025-11-09T03:39:59.123Z",
  "channel": 36,
  "cca_busy_pct": 0.528,
  "wifi_airtime_pct": 0.412,
  "noise_floor_dbm": -51.3,
  "snr_db": 82.1
}
```

Notes: timestamps should be ISO8601 in UTC. Metric units consistent across producers. Frame intervals may vary — the pipeline uses time windows for voting/debounce.

4.3 Alert output (JSON)

If no alert:

```
{"alert": false, "timestamp": "2025-11-09T03:40:00Z"}
```

If alert:

```
{
  "alert": true,
  "timestamp": "2025-11-09T03:40:12.345Z",
  "channel": 36,
  "metrics_fired": ["cca_busy_pct","snr_db"],
  "reason": "2-of-4 vote + debounce satisfied",
  "event_id": "evt-20251109-0001",
  "window_start": "2025-11-09T03:40:10Z",
  "window_end": "2025-11-09T03:40:15Z"
}
```

4.4 When no alert is produced

Output is a minimal JSON indicating no alert. Internally detectors still update their states.

5. Conclusions

- **Per-metric detectors** match mathematical sensitivity to real signal behavior (EWMA for bursty fractions, CUSUM for abrupt SNR dips, SPRT for distribution shifts in noise floor).
- **Voting + debounce + cooldown** reduces operator noise: voting avoids single-metric false positives, debounce avoids transient collisions, cooldown prevents repeated alerts for the same event.
- **Hardcoded baselines and rolling median** keep runtime simple and low latency while robust to transient spikes and sampling irregularities.
- **Event stitching** produces human-friendly events rather than many tiny alerts.

The hybrid pipeline described here blends simple, fast online detectors with a robust fusion layer that suppresses noise and produces actionable alerts. It respects low-latency constraints, handles irregular sampling, and provides clear I/O for integration. With modest parameter tuning per deployment (baselines + thresholds), this pipeline is ready for real-data evaluation and integration into real-time orchestration layers (Arista / Cisco telemetry processors).