Client Centric WiFi-RRM

Arista Networks

Team 15

November 2025

Contents

1. Introduction

This report presents Team 15's client-centric Wi-Fi RRM system, designed to address the Inter-IIT Arista challenge. Our solution focuses on building a practical and intelligent RRM pipeline that combines additional-radio sensing, real-time learning, and QoE-aware policy control.

We developed a dedicated sensing module that continuously scans Wi-Fi and non-Wi-Fi bands without impacting client airtime. Using our custom Decision Tree classifier, the system identifies key interference sources such as BLE, Zigbee, and microwave activity. To efficiently utilize sensing time, we implemented an Adaptive Dwell Scheduling framework based on a Sense–Bandits approach, supported by an offline model factory and an online multi-armed bandit agent that dynamically allocates dwell times under strict timing budgets.

To ensure reliability, we built a hybrid change-detection pipeline combining EWMA, SPRT, and CUSUM, fused with a K-of-N voting mechanism to deliver stable, low-latency interference alerts. Beyond sensing, our system captures the client perspective through 802.11k measurements and ranks neighboring APs using RSSI, SNR, load, and capacity. A Constrained Bayesian Optimizer then tunes AP parameters—such as transmit power and OBSS-PD—while respecting QoE-based service constraints. Our custom QoE engine integrates signal quality, throughput, reliability, latency, and activity into a unified performance score.

Overall, the solution integrates sensing, classification, learning-based scheduling, change detection, and policy optimization into a cohesive RRM architecture, reflecting Team 15's systematic and data-driven approach to improving Wi-Fi performance.

2. Designs Delivered

2.1 Part 1: Additional-Radio Sensing & Scheduling

The first part requires an additional system which can sense and classify non-wifi signals like BLE, Zigbee, etc which are the usual causes of interference in the 2.4 GHz ISM band. Also, the system needs to have dynamic dwell times per channel and should be able to promptly detect and react to sudden changes in the channels.

Figure 2.1 explains the system we have developed to address this task.

2.1.1 Sensing

For the sensing signals, the AP will be having a dedicated antenna system, separate from the radios responsible for serving client traffic (the 2.4 GHz, 5 GHz, and 6 GHz data radios). It allows for continuous, **zero-impact spectrum analysis** without causing a performance hit (i.e., draining client airtime) to the main Wi-Fi radios. It can constantly scan all Wi-Fi channels and the surrounding non-Wi-Fi frequencies (like those used by Bluetooth and Zigbee in the 2.4 GHz ISM band).

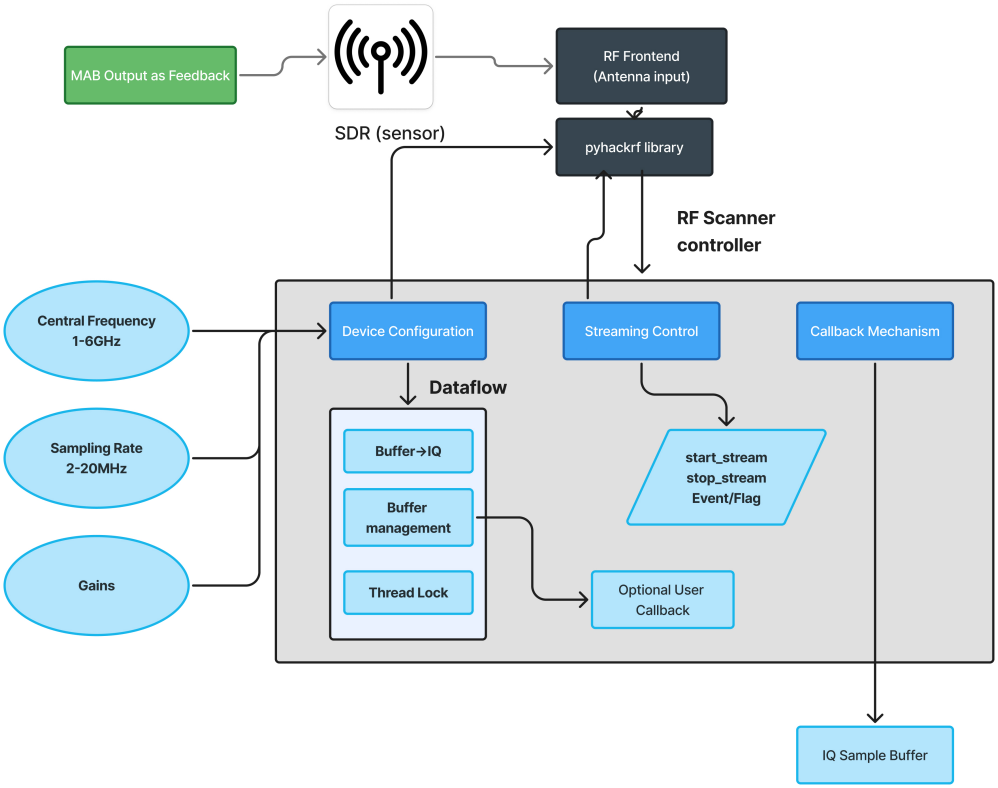

Figure 2.1: Sensing and Streaming System

2.1.2 Non - Wifi Classifier: Decision Tree

We adopt a decision tree classifier for its interpretability, low computational cost, and strong performance on engineered spectral features. Specifically, we use sklearn.tree.DecisionTreeClassifier with the criterion set to 'entropy', which applies Shannon information gain as the split metric. This approach closely follows the logic of Quinlan's C4.5 decision tree, an evolution of ID3 that supports continuous-valued attributes and post-pruning.

While the original C4.5 algorithm applies gain-ratio for attribute selection and reduced-error pruning, we approximate pruning using the ccp alpha (cost-complexity pruning) parameter provided by scikit-learn. By tuning ccp alpha typically via cross-validation, we identify the smallest tree that retains near-optimal validation accuracy. This regularization reduces overfitting while preserving interpretability and decision fidelity.

Why Decision Trees?

Decision trees offer several advantages particularly suited to our RF classification pipeline:

Transparency and Simplicity: They produce human-readable rules (e.g., threshold conditions on features like bandwidth or duty cycle), making them easy to inspect, debug, and interpret in real-time deployments.

Efficient Inference: Unlike support vector machines (SVMs) or deep neural networks, decision trees require negligible memory and CPU time, making them ideal for real-time classification on embedded systems or constrained compute platforms.

Probabilistic Output: Decision trees can output class probabilities based on label histograms in leaf nodes, providing not just a hard prediction but also a confidence score useful for downstream decision-making or fusion.

We evaluated multiple classifiers for non-WiFi device detection and found that decision trees achieved comparable accuracy to SVMs, while being significantly faster and more lightweight. Random forests, while more accurate, lose transparency and demand more memory. CNNs or other deep models would require substantially larger datasets and careful architecture tuning, making them less viable for our current scope.

Given these trade-offs and the structure of our feature space, a pruned decision tree strikes the optimal balance between performance, interpretability, and deployment readiness.

The structure of the decision tree has been shown in figure.

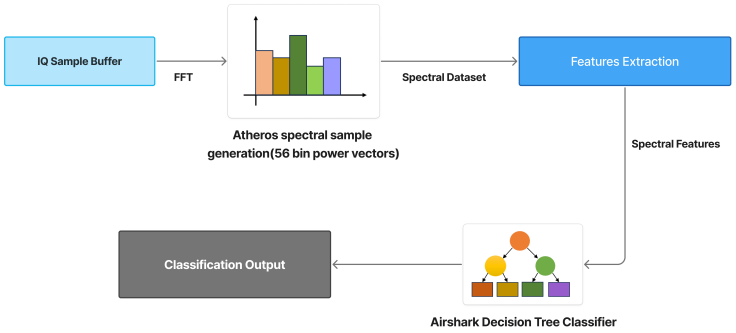

Figure 2.2: Non - Wifi Classifier

2.1.3 Adaptive Dwell & Scheduling: MAB Dwell Time Optimization

The Adaptive Dwell & Scheduling mechanism employs a **Sense-Bandits framework** to optimize the allocation of sensing time across Wi-Fi channels. This approach utilizes a dedicated radio to provide continuous spectrum intelligence while adhering to a strict execution budget (e.g., 1000ms Epoch Budget). The system operates in two phases: an Offline Model Factory for generalized learning and an Online Learning & Allocation Loop for real-time fine-tuning.

Phase 1: The Offline Model Factory (Pre-computation)

This sparsely performed phase builds a robust look-up table to address the cold start problem and provide a strong foundation for the online agents.

1. Massive Dataset Generation: Millions of (State, Action, Reward) tuples are generated in a simulator across a huge variety of interference scenarios (e.g., microwave, Bluetooth, other Wi-Fi).State (e_t): The Sensing Fingerprint (SF) of the environment. Action (a): A quantized dwell time chosen from a discrete set of actions, typically ranging from 25ms up to 500ms.Reward (r_t): The measured noise/interference utility resulting from the action.

2. Clustering and Utility Mapping: A K-Means clustering algorithm (using Euclidean distance) is run on this dataset to group similar channel states. The result is a set of centroids, radii, and the average utility (μ_k) (reward) for each cluster. This allows the system to predict the expected reward for any channel state by simply finding its nearest historical cluster.

3. Model Storage: The output-centroids.dat, radii.dat, and utilities.dat—completes the offline phase.

Phase 2: The Online Learning & Allocation Loop (Real-Time)

This continuous loop runs on the device, using the offline model as a "head start" and adapting to non-stationary environments.

1. **Initialization and State Loading**: Upon startup, an independent Online Agent is initialized for each channel. By performing a quick GetSensingFingerprint(), the agent compares the initial state to the offline centroids to load its starting cluster, k^*. This instantly loads the cluster's average utility (μ_k) and initializes the agent's memory (e.g., $b_{regression}$ and $X_{regression}$) to zero/identity, effectively mitigating the cold-start problem.

2. **Estimate & Update Phase (The "RL Brain")**: In each epoch, agents evaluate their last experience and adapt their learning parameters. Change Detection: The agent checks for environmental change by comparing its last state fingerprint to its current centroid using euclidean distance.Non-Stationary Adaptation: If $distance > radius$ (Environment Changed), the agent resets its fine-tuning memory ($b_{regression}$ to $\mathbf{0}$, $X_{regression}$ to $\mathbf{I}$) and finds the new best cluster k^*. This allows for rapid adaptation to sudden environmental shifts.Stable Learning: If $distance \leq radius$ (Environment Stable), the agent updates $b_{regression}$ and $X_{regression}$ using the last reward and action vector, performing the core "learning from reward" step.Prediction: The agent solves for a new correction vector ($\tilde{\mu}_{t,e}$) and calculates an optimistic score

(UCB) for all possible dwell times: $Score(a) = (\mu_k) + (\tilde{\mu}_{t,e}) + (CB_{t,e})$. This produces an 11 × number of actions matrix of optimistic scores.

3. **Allocate Phase (Knapsack Solver)**: The central controller uses the score matrix and the remaining execution budget to determine the optimal plan. It executes a Knapsack algorithm to find the combination of 11 dwell times (one per channel) that maximizes the sum of optimistic scores without exceeding the budget. This ensures the sensing plan is globally optimal for the current epoch's budget constraints.

4. **Execution and History Storage**: The controller executes the chosen dwell times, measures the actual rewards (r_t) and stores the new channel fingerprints, rewards, and action vectors as the historical data for the next epoch's update phase. This ensures the system is constantly exploring new channel states and exploiting the learned utility for scheduling.

2.1.4 Change Detection

To detect real-time RF anomalies, our system implements a **Hybrid Change Detection Pipeline** that applies a different Sequential Change Detection (SCD) algorithm to each RF metric based on its statistical behavior. The outputs of these detectors are combined using a lightweight voting rule with debounce and cooldown logic to ensure high precision and low latency.

Input Metrics

The system utilizes four key Wi-Fi Radio Frequency (RF) metrics, which are monitored over time. These metrics are categorized by the detection algorithm used as given below:

- CCA Busy (%) – Channel utilization based on Clear Channel Assessment.
- WiFi Airtime (%) – Time occupied by Wi-Fi frames.
- Noise Floor (dBm) – Ambient wideband noise level.
- SNR (dB) – Signal-to-noise ratio from the AP perspective.

Each metric is mapped to a detector optimized for its temporal and noise characteristics, as shown below:

Metric	Algorithm Used
CCA Busy %	EWMA
WiFi Airtime %	EWMA
Noise Floor dBm	SPRT
SNR dB	CUSUM

Baseline Modeling

For each metric x_t, baselines are computed offline from clean data using robust estimators:

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

These are fixed at runtime to avoid warm-up delay.

Detection Algorithms

Three distinct Sequential Change Detection (SCD) algorithms are employed, each best suited for different types of distributional changes:

1. **Exponentially Weighted Moving Average (EWMA):**

   `Used for: cca_busy_pct, wifi_airtime_pct`

   Mechanism: EWMA is a smoothing-based change detector well-suited for noisy, bursty RF metrics. It computes a recursively updated smoothed mean:

   $$Z_t = \alpha X_t + (1 - \alpha) Z_{t-1}$$

   where $0 < \alpha < 1$ controls the responsiveness.

   A change is signaled when the smoothed deviation from the baseline μ_0 exceeds a statistically scaled limit:

   $$|Z_t - \mu_0| > L \cdot \sigma_0 \sqrt{\frac{\alpha}{2 - \alpha}}$$

   This makes EWMA highly effective at filtering transient spikes while detecting sustained upward or downward trends.

2. **Sequential Probability Ratio Test (SPRT):**

   `Used for: noise_floor_dbm`

   Mechanism: SPRT is a likelihood-ratio-based detector optimal for real-time identification of shifts between two known distributions. For each new observation X_t, it computes the log-likelihood ratio:

   $$S_t = S_{t-1} + \log\left(\frac{f_1(X_t)}{f_0(X_t)}\right),$$

   where f_0 and f_1 denote the pre-change and post-change probability densities.

   An alarm is raised when the statistic crosses an upper threshold:

   $$S_t \geq \log B.$$

   SPRT is particularly effective for abrupt shifts in noise-floor behavior.

3. **Cumulative Sum (CUSUM):**

   `Used for: nonwifi_duty_pct, snr_db`

   Mechanism: CUSUM is effective at detecting abrupt, step-like shifts in the mean of a process. It maintains two cumulative sums: $\mathbf{S_{pos}}$ (for an upward shift in the mean) and $\mathbf{S_{neg}}$ (for a downward shift in the mean). Both sums subtract a slack parameter ($k = 0.5$), making it insensitive to minor noise. An alarm is raised if either $\mathbf{S_{pos}}$ or $\mathbf{S_{neg}}$ exceeds the limit ($h = 5$).

Fusion Engine (Alert Logic)

The Fusion Engine combines the binary alert outputs from the four individual detectors into a single system alert, enforcing three key operational rules:

1. **Rule 1 — K-of-N Voting (K=2):** An alert is only considered if at least 2 of the 4 metrics detect an unusual change. This minimizes false positives that might arise from an isolated metric fluctuation.
2. **Rule 2 — Debounce (D = 2 frames):** Once the K-of-N rule is met, the condition must persist for 2 consecutive frames before a final alert is raised. This filters out transient spikes and ensures the event is stable.
3. **Rule 3 — Cooldown (30 frames):** After a final alert is successfully raised, the system enters a cooldown period of 30 frames (approximately 30 seconds). During this time, no new alerts can be generated.

Event Stitching

To prevent rapid repetitive alerts during a single interference burst, alerts occurring within a 30-second gap are merged into a single event interval. This produces clean, interpretable output suitable for higher-layer policy modules.

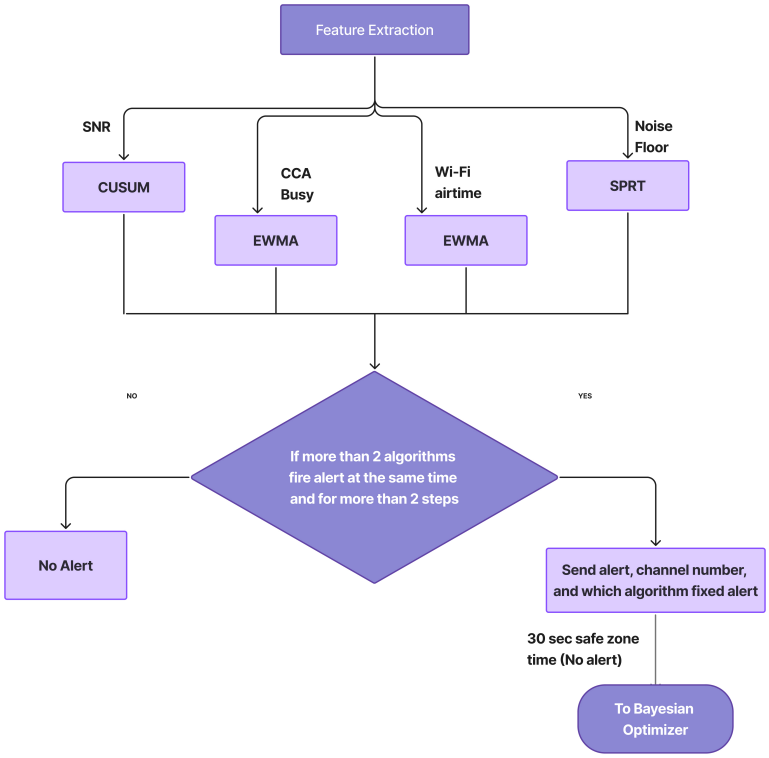

Figure 2.3: Change Detection Pipeline

2.2 Part 2: Policy & Guardrails

The first component, Client-View Acquisition, is implemented by the Neighbor AP Ranking Module. This module fulfills the requirement for obtaining client-side data (like RSSI and SNR) via 802.11k beacon/link measurements, and uses this data to compute a lightweight, real-time ranked list of candidate Access Points (APs) for roaming. The ranking utilizes a scoring function that weights RSSI, SNR, and an optional historical bonus from the NeighborDB. This ranked list is essential input for an 802.11v BSS-TM implementation, which aids in crafting effective roam recommendations.

The second component is the Policy & Guardrails system, which uses a Constrained Bayesian Optimizer (CBO) to tune critical AP parameters, such as channel width, transmit power (tx power dBm), and OBSS-PD. The CBO approach is highly sample-efficient ("limited online BO") and ensures that performance is maximized (Reward) while complex Service Level Objectives (SLOs) are respected (Constraints). This is achieved using separate Gaussian Process models to predict the primary reward and the safety margins for each constraint (e.g., maximizing P50 throughput while keeping P95 retries below 8%). The core optimization relies on the Expected Improvement Constrained (EIC) acquisition function, which balances exploration with the safety requirement, thus managing configuration churn effectively. The QoE Evaluation Engine serves as the feedback mechanism for this system, providing the continuous, multi-dimensional performance score (0.0–1.0) required to measure the true reward and log crucial post-roam QoE deltas. The QoE score combines factors like Signal Quality, Reliability, Throughput, Latency, and Activity.

In summary, the combined system first uses standards-based measurements to understand the client's perspective and potential roam targets, and then uses a constrained, learning-based engine (BO fed by QoE data) to continuously and safely optimize the network configuration to guarantee service quality and policy adherence

2.3 Part 3: Client-View Acquisition

Schedulers and Data Acquisition

The system uses several independent schedulers that execute at fixed intervals to gather measurements and compute analytics. Each scheduler follows a uniform lifecycle: it wakes, collects the required data, performs computations, stores results, and returns to sleep. This ensures predictable timing and prevents bursts of measurement traffic.

The Link Measurement Scheduler collects IEEE 802.11k Link Measurement Reports from stations lacking recent data, storing RSSI, link margin, and transmit power. The Beacon Measurement Scheduler gathers IEEE 802.11k Beacon Reports, including RCPI, channel utilization, and AP capabilities, which are aggregated for neighbor ranking. The QoE Scheduler computes the Quality of Experience at frequent intervals using link measurements, throughput statistics, retry rates, and activity data. The Neighbor Ranking Scheduler processes aggregated beacon reports and generates a sorted list of candidate APs. The BSS Transition Scheduler evaluates QoE and applies steering logic before issuing IEEE 802.11v BSS Transition Management (BSS-TM) requests when needed.

Quality of Experience (QoE) Computation

QoE reflects how well a station's current AP is serving its needs. It is calculated using the weighted formula:

$\mathrm{Q} = \mathrm{w}_s S + w_t T + w_r R + w_l L + w_a A$

Where the components are: S = Signal Quality T = Throughput R = Reliability L = Latency A = Activity The weights sum to 1 and are configured as: $w_s = 0.28, w_t = 0.32, w_r = 0.15, w_l = 0.15, w_a = 0.10$ Signal Quality is derived from RSSI values mapped linearly between 90 dBm (score = 0) and 30 dBm (score = 1). Throughput uses the geometric mean of TX and RX bitrates normalized by the station's PHY peak.Reliability decreases as retry and FCS error rates increase. Latency is based on inactivity time, where prolonged inactive periods reduce the score. Activity reflects overall usage through frame counts normalized by a reference maximum. These components together produce a single QoE score between 0 and 1.

Neighbor Ranking Algorithm

To identify suitable roaming targets, each neighbor AP is assigned a score using:

Ni = $y_1 RSSI_i + y_2 capacity_i y_3 load_i Where y_1 = 0.55, y_2 = 0.35, y_3 = 0.10$. RSSI, capacity, and channel load are normalized. Multiple beacon reports are aggregated to reduce noise, and APs are sorted by score. Optional filters may enforce minimum RSSI, SSID matching, or band preferences. A future upgrade will integrate post-roam QoE deltas into this scoring.

Transition Management Logic

The steering engine evaluates every station's QoE and compares it against the threshold of 0.55. If QoE is acceptable, steering is skipped. If QoE is low and valid neighbor rankings exist, the system constructs a BSS-TM request including an ordered list of recommended APs. Safety mechanisms—including minimum time between steers and required RSSI improvements—prevent unnecessary roaming. Steering actions are logged, and client responses (accept, reject, ignore) are recorded for future adaptive logic.

Future Enhancements

A major upcoming feature is Post-Roam QoE Delta Analysis. After a roam completes, QoE will be measured again and the improvement computed:

$$\Delta Q = Q_{\text{after}} - Q_{\text{before}}$$

This value will be incorporated into future ranking calculations. Additional planned improvements include load-aware steering, predictive steering using QoE trends, and device capability fingerprinting to match clients to optimal APs.