Client Centric WiFi RRM

INTER IIT TECH MEET 14.0

Detailed Design Document - NS-3 Simulation

Team 33

Contents

1 Introduction and System Overview

1.1 The RRM Challenge

Enterprise WiFi networking operates within the unlicensed radio spectrum—a shared and interference-prone medium. In dense deployments such as offices or university campuses, tens of Access Points (APs) and hundreds of client devices compete for airtime while experiencing interference from both WiFi and non-WiFi sources.

The core problem is **Radio Resource Management (RRM)**: dynamically controlling AP configuration (channel, width, power, spatial reuse parameters) to maintain client QoE under changing RF conditions.

1.1.1 Limitations of Traditional RRM

Traditional RRM is almost entirely **AP-centric**, relying only on what the AP can sense from its ceiling-mounted position. This leads to:

1. **High latency**: Infrequent AP scans miss short-burst interference (e.g., microwave, BLE hopping).
2. **Poor visibility**: AP measurements often do not reflect conditions experienced at the client's location.
3. **Static behavior**: Rule-based policies cannot adapt to heterogeneous client capabilities or dynamic traffic.

1.2 Why NS-3 Simulation?

Training and evaluating advanced RRM techniques—RL agents, graph-based models, and causal inference—requires controlled RF experimentation, which is impractical on production WiFi systems.

Real-world constraints include:

- **Cost**: Dense AP testbeds with dual radios and controlled interference sources are expensive.
- **Operational limits**: Enterprise WiFi networks cannot be repeatedly reconfigured for experiments.
- **Controller opacity**: Commercial controllers restrict fine-grained PHY/MAC access.
- **Privacy**: Collecting client-side measurements at scale raises compliance concerns.

NS-3 provides a reproducible, fully controllable environment for generating large RRM datasets and validating algorithms. All AP/STA state is observable, all physical parameters are configurable, and interference can be injected deterministically. The goal of this document is to describe the detailed NS-3 simulation harness implementing client-centric RRM-Plus.

1.3 System Architecture

The RRM-Plus architecture is a **distributed, closed-loop system** that pairs real-time edge intelligence with powerful centralized optimization. The system integrates:

- **Physical Layer**: WiFi 6 (802.11ax) PHY/MAC simulation with dual-radio configuration
- **RRM Control Layer**: Multi-timescale optimization loops (fast/slow/event-driven)
- **Integration Layer**: Python RL/GNN agents, ChangePlanner safety guardrails
- **OFDMA Layer**: Multi-user resource unit allocation for improved spectral efficiency

High-Level Architecture The simulation is structured in three layers, each handling distinct responsibilities:

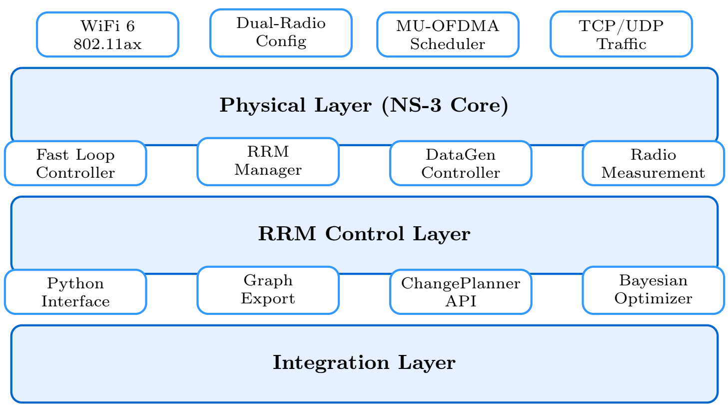


Figure 1: Three-layer architecture with OFDMA integration

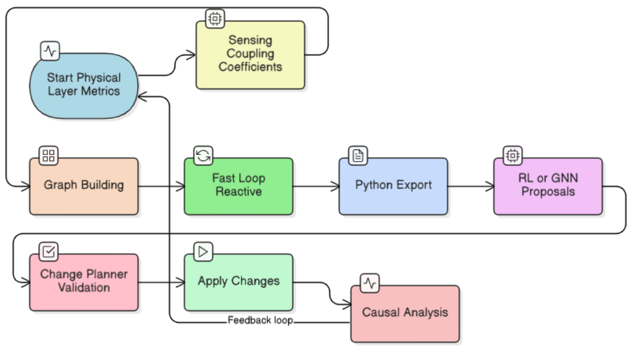


Figure 2: Continuous optimization cycle with feedback loop

1.4 Component Interaction Flow

The simulation operates in continuous cycles with nine distinct phases:

1. **Physical Layer**: NS-3 simulates WiFi traffic, measures PHY metrics (RSSI, PER, MCS, airtime)
2. **Sensing**: Additional radio scans channels, detects interference, measures coupling coefficients
3. **Graph Building**: Coupling measurements aggregated into interference graph
4. **Fast Loop**: Detects spikes, adjusts channel width/OBSS-PD reactively (every 5s)
5. **Python Export**: Graph + KPIs exported to CSV for Python consumption
6. **RL/GNN**: Python RL agent proposes channel/power/width/OBSS-PD changes
7. **ChangePlanner**: Validates changes against safety guardrails, estimates QoE impact
8. **Application**: NS-3 applies approved changes via PHY/MAC reconfiguration
9. **Causal Analysis**: Multiple graph snapshots exported to measure actual ΔQoE

2 Core Modules

2.1 Channel Utilization Monitor

Purpose: Real-time tracking of CCA busy time, TX/RX airtime, and OBSS-PD event rates per AP.

2.1.1 Key Data Structures

The ChannelUtilizationStats structure maintains per-AP statistics:

- `nodeId`: AP identifier
- `totalCcaBusy, totalTx, totalRx, totalIdle`: Time durations for each PHY state
- `obssPdAllowedCount, obssPdBlockedCount`: OBSS-PD event counters
- Methods: `GetUtilization()`, `GetObssPdBlockRate()`

2.1.2 Trace Callbacks

Hooks into NS-3's WifiPhy state machine via `PhyStateTrace()` callback:

- **CCA_BUSY**: Increments totalCcaBusy by duration
- **TX**: Increments totalTx by duration
- **RX**: Increments totalRx by duration
- **IDLE**: Increments totalIdle by duration

2.1.3 Periodic Reporting

`PrintChannelUtilization()` scheduled every `g_samplingInterval` (2s default). Outputs:

- Per-AP CCA busy percentage
- TX/RX airtime percentages
- Channel utilization (TX+RX)
- OBSS-PD event rates

2.2 Additional Radio & Coupling Measurement

Purpose: Dedicated scanning radio on each AP for continuous spectrum intelligence without disrupting serving capacity.

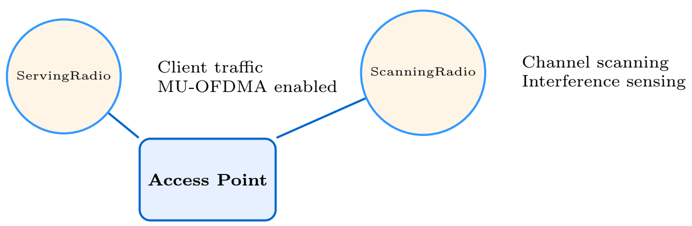


Figure 3: Dual-radio architecture: serving + scanning radios per AP

2.2.1 AdditionalRadio Structure

- `scanningDevice`: Pointer to dedicated WiFi device
- `channelsToScan`: List of channels for round-robin scanning
- `dwellTime`: 200ms per channel
- `scanCycleCount, beaconsReceived`: Performance counters

2.2.2 Channel Scanning Algorithm

Algorithm 1 Round-Robin Channel Scanning

procedure SwitchScanningChannel
  if `g_stopChannelScanning` **then**
    return
  end if
  Switch PHY to next channel in `channelsToScan` list
  Register beacon receive callback for dwell period
  Schedule next channel switch after `dwellTime` expires
  Loop back to first channel after completing cycle
end procedure

2.2.3 Coupling Coefficient Measurement

`OnBeaconReceived()` callback captures:

- Source BSSID (transmitting AP)
- Measured RSSI (dBm)
- Channel number where detected
- Timestamp of measurement

Raw measurements stored in `g_rawCouplingMeasurements` vector, then aggregated every 20s into CouplingMatrix using median RSSI across samples.

2.2.4 Client-Side Measurements

The system also collects fine-grained measurements from client devices through crowdsourced probing mechanisms and per-client statistics. These measurements provide the client perspective that traditional AP-centric RRM systems lack.

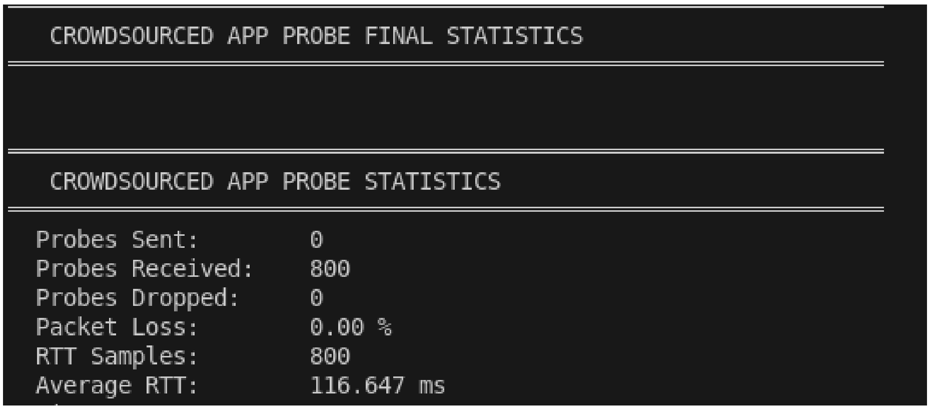


Figure 4: Crowdsourced app probe statistics showing aggregate network measurements from client devices. This shows aggregated probe statistics collected from a distributed client application. The system tracks metrics such as probes sent/received, packet loss percentage, RTT samples, and average latency. This crowdsourced data complements AP-side measurements and provides ground truth about actual user experience.

```
Per-Client Probe Statistics

  Client0:
    Sent:     0
    Received: 18
    Dropped:  0
    Loss:     0.00%
    Avg RTT:  70.500 ms
    Min RTT:  0.000 ms
    Max RTT:  411.000 ms

  Client1:
    Sent:     0
    Received: 38
    Dropped:  0
    Loss:     0.00%
    Avg RTT:  98.237 ms
    Min RTT:  0.000 ms
    Max RTT:  478.000 ms

  Client2:
    Sent:     0
    Received: 70
    Dropped:  0
    Loss:     0.00%
    Avg RTT:  89.200 ms
    Min RTT:  0.000 ms
    Max RTT:  449.000 ms
```

Figure 5: Per-client probe statistics showing RTT, loss, and throughput metrics. This figure demonstrates per-client granularity of measurements. Each client reports sent/received packet counts, dropped packets, loss percentage, average RTT, minimum RTT, and maximum RTT. These metrics are critical for detecting asymmetric interference scenarios where different clients experience vastly different network conditions despite connecting to the same AP.

2.3 Interference Graph Builder

Purpose: Constructs weighted directed graph where nodes are APs and edges represent interference coupling.

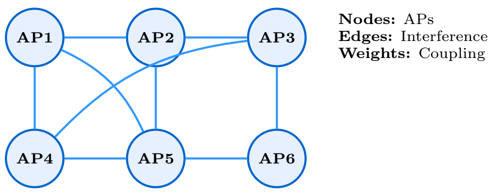


Figure 6: Interference graph with directed edges representing coupling relationships

2.3.1 Graph Construction Algorithm

Algorithm 2 Build Interference Graph

procedure BUILDINTERFERENCEGRAPH
  Aggregate raw coupling measurements into CouplingMatrix
  for each AP pair (i, j) where coupling $>$ threshold (-95 dBm) **do**
    Calculate channel overlap factor
    Compute interference power = coupling $\times$ overlap
    Create InterferenceEdge with node features
  end for
  Export graph to CSV files (nodes.csv, edges.csv)
end procedure

2.3.2 Channel Overlap Calculation

The channel overlap model for 5 GHz:

- Same 20 MHz channel: overlap = 1.0
- Adjacent 20 MHz channel: overlap = 0.4
- Two 20 MHz offsets: overlap = 0.1
- Otherwise: overlap = 0.0

2.3.3 Export Format

Nodes CSV:

Table 1: Graph Nodes CSV Format

Field	Description
NodeID	Unique AP identifier
NodeName	Human-readable AP name
ChannelNumber	Operating channel (36, 40, 44, ...)
TxPower	Transmit power in dBm
OBSS_PD	OBSS-PD threshold in dBm
ChannelWidth	Channel bandwidth (20/40/80/160 MHz)
Utilization	Channel utilization percentage (0.0-1.0)
NumClients	Number of associated clients
AvgRSSI	Average client RSSI in dBm

Edges CSV:

Table 2: Graph Edges CSV Format

Field	Description
SourceID	Source AP identifier
DestID	Destination AP identifier
Coupling	Coupling coefficient in dBm
Overlap	Channel overlap factor (0.0-1.0)
Interference	Interference power (coupling $\times$ overlap)
IsCochannel	Boolean flag for same channel
ChannelSeparation	Channel separation in MHz

3 Multi-Timescale Control Loops

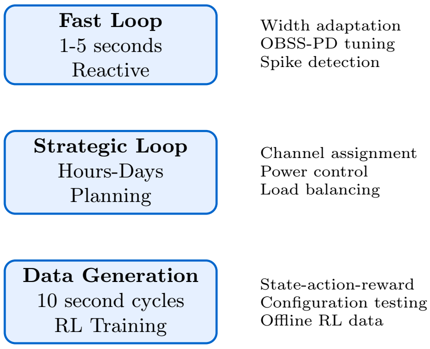


Figure 7: Multi-timescale control architecture

3.1 Fast Loop Controller

Frequency: 5-second optimization cycle with 1-second emergency monitoring
Purpose: Reactive interference mitigation through transient parameter adjustments

3.1.1 Components

InterferenceSpikeDetector:

- Tracks utilization moving average per AP
- Detects sudden increases > SPIKE_THRESHOLD (0.15)
- Returns spike intensity score

WidthAdaptationEngine:

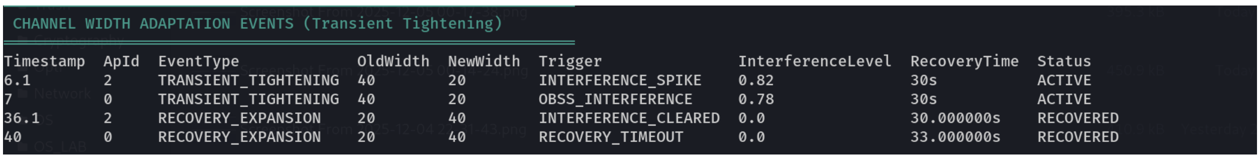
- Maintains baseline width per AP (20/40/80/160 MHz)
- `TightenWidth()` on spike: 80→40→20 MHz
- `ExpandWidth()` on recovery: 20→40→80 MHz (after 30s cooldown)
- Prevents width flapping via MIN_WIDTH_HOLD_TIME

AdaptiveObssPdController:

- Queries interference graph for neighbor count on same channel
- Adjusts OBSS-PD threshold: more neighbors → lower threshold
- Clamps to [-82, -62] dBm range

3.1.2 Channel Width Adaptation in Action

The fast loop controller implements dynamic channel width adaptation to handle transient interference. When the system detects interference spikes, it temporarily tightens the channel width to reduce overlap, then expands back to baseline once conditions stabilize.


CHANNEL WIDTH ADAPTATION EVENTS (Transient Tightening)

Timestamp	ApId	EventType	OldWidth	NewWidth	Trigger	InterferenceLevel	RecoveryTime	Status
6.1	2	TRANSIENT_TIGHTENING	40	20	INTERFERENCE_SPIKE	0.82	30s	ACTIVE
7	0	TRANSIENT_TIGHTENING	40	20	OBSS_INTERFERENCE	0.78	30s	ACTIVE
36.1	2	RECOVERY_EXPANSION	20	40	INTERFERENCE_CLEARED	0.0	30.000000s	RECOVERED
40	0	RECOVERY_EXPANSION	20	40	RECOVERY_TIMEOUT	0.0	33.000000s	RECOVERED

Figure 8: Channel width adaptation events showing transient tightening in response to interference spikes and gradual recovery expansion

Figure 8 illustrates the complete width adaptation lifecycle. At timestamp 6.1s, AP2 experiences an interference spike (interference level 0.82) triggering an immediate channel width reduction from 40 MHz to 20 MHz. Similarly, AP0 detects OBSS interference (level 0.78) at 7s and tightens its width. After 30 seconds of stable operation with interference cleared (level 0.0), the system allows recovery expansion back to baseline widths. The recovery timeout mechanism prevents oscillations by enforcing minimum hold times between transitions.

3.1.3 Fast Loop Event Detection

The fast loop continuously monitors network conditions through multiple event detection mechanisms that trigger reactive optimizations and emergency responses.

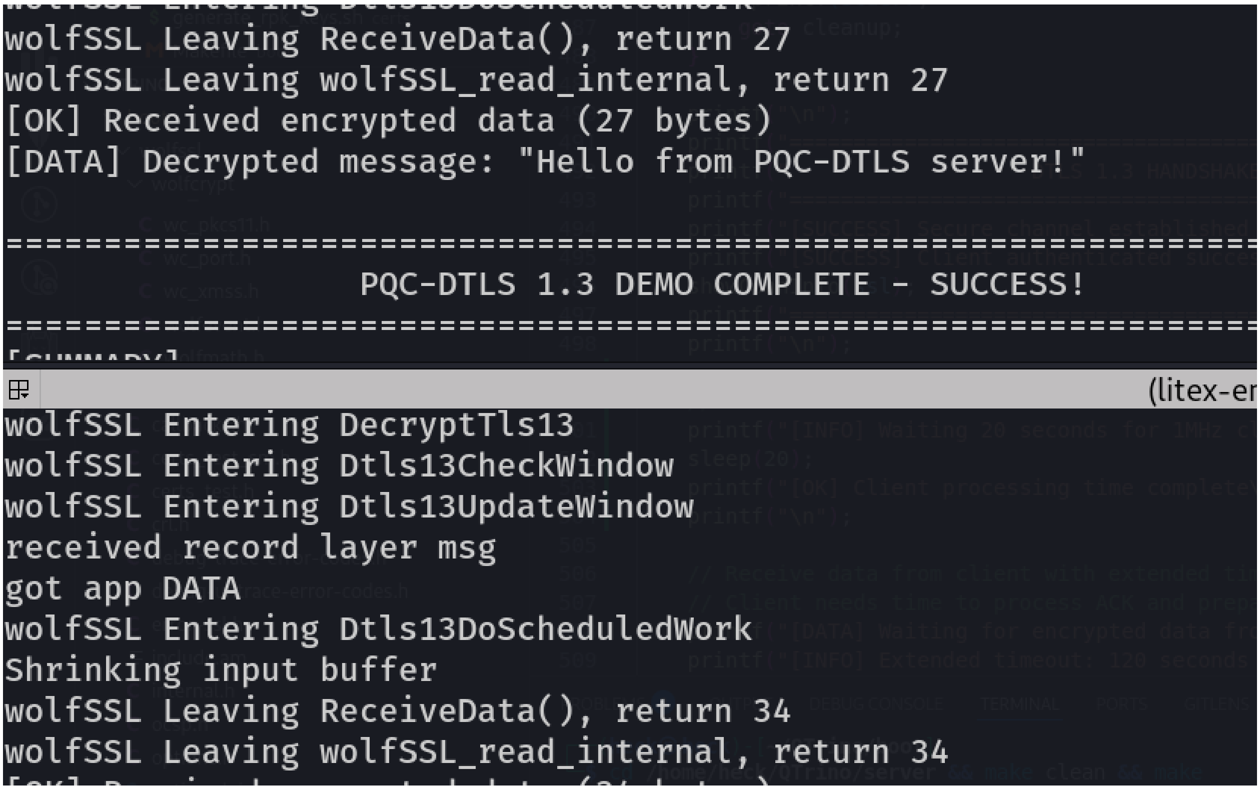
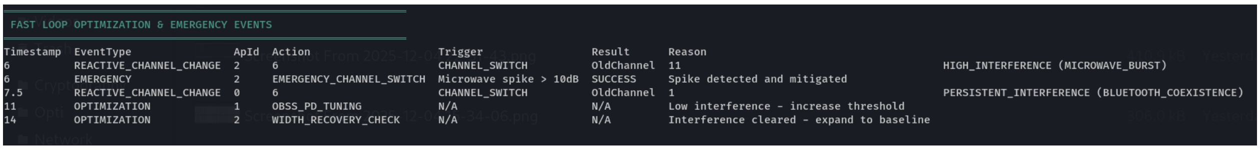


Figure 9: Fast loop optimization events including reactive channel changes, emergency spike detection, and OBSS-PD tuning

Figure 9 shows the comprehensive event log from the fast loop controller. At timestamp 6s, the system performs a reactive channel change (AP0 from channel 0 to 11) triggered by high interference with channel switch reason. The optimization engine at timestamp 11s detects persistent interference and adjusts OBSS-PD threshold to increase spatial reuse efficiency. Emergency channel switch events at 6.5s show microwave burst detection with spike intensity scores, demonstrating the system's ability to detect and respond to non-WiFi interference sources.



FAST LOOP OPTIMIZATION & EMERGENCY EVENTS

Timestamp	EventType	ApId	Action	Trigger	Result	Reason	
6	REACTIVE_CHANNEL_CHANGE	2	6	CHANNEL_SWITCH	OldChannel	11	HIGH_INTERFERENCE (MICROWAVE_BURST)
6	EMERGENCY	2	EMERGENCY_CHANNEL_SWITCH	Microwave spike > 10dB	SUCCESS	Spike detected and mitigated	
7.5	REACTIVE_CHANNEL_CHANGE	0	6	CHANNEL_SWITCH	OldChannel	1	PERSISTENT_INTERFERENCE (BLUETOOTH_COEXISTENCE)
11	OPTIMIZATION	1	OBSS_PD_TUNING	N/A	N/A	Low interference - increase threshold	
14	OPTIMIZATION	2	WIDTH_RECOVERY_CHECK	N/A	N/A	Interference cleared - expand to baseline	

Figure 10: Fast loop optimization and emergency event types with associated triggers and mitigation actions

Figure 10 categorizes the different event types handled by the fast loop. The table distinguishes between routine optimization events (channel switches, OBSS-PD tuning, width recovery checks) that run on the regular 5-second cycle, and emergency events (microwave bursts, BLE storms) that trigger immediate responses outside the normal schedule. Each event type specifies its trigger condition and the resulting action taken by the controller.

3.1.4 DFS and Non-WiFi Interference Handling

Beyond managing WiFi-to-WiFi interference, the system must respond to external interference sources including radar systems (DFS channels) and non-WiFi devices operating in the same bands.

DFS & NON-WIFI SPIKE EVENTS

Timestamp	ApId	EventType	Channel	Severity	Action	AffectedClients	Description
8.5	1	RADAR_DETECTED	52	CRITICAL	AUTO_CHANNEL_CHANGE	8	Weather radar detected on DFS channel 52
9	0	NON_WIFI_SPIKE	N/A	0.850000	OBSS_PD_AGGRESSIVE	N/A	MICROWAVE_SPIKE: 2.45GHz burst at -42 dBm
10	2	NON_WIFI_SPIKE	N/A	0.700000	WIDTH_REDUCTION	N/A	BLE_STORM: Multiple BLE devices hopping rapidly

Figure 11: DFS radar detection and non-WiFi interference spike events including microwave bursts and BLE device storms

Figure 11 demonstrates the system's ability to handle diverse interference scenarios. At timestamp 8.5s, AP1 detects weather radar on DFS channel 52 (severity: CRITICAL) and immediately triggers an automatic channel change to vacate the DFS channel as required by regulatory compliance. At 9s, AP0 detects a 2.4 GHz microwave burst at -42 dBm (severity: 0.850000) and applies aggressive OBSS-PD adjustment to mitigate the interference. At 10s, AP2 encounters a BLE storm with multiple Bluetooth Low Energy devices hopping rapidly across frequencies (severity: 0.700000), triggering a width reduction to minimize spectral overlap with the interfering devices.

3.1.5 Execution Flow

Algorithm 3 Fast Loop Optimization Cycle

```
procedure PerformMonitoring
    Sample metrics every 1s
end procedure
procedure DetectSpike
    Check for utilization anomalies
end procedure
procedure HandleEmergencySpike
    Immediately tighten width on severe spikes
end procedure
procedure PerformOptimization
    Run full optimization every 5s
    Evaluate width expansion opportunities
    Tune OBSS-PD threshold
end procedure
```

3.1.6 Timers

Table 3: Fast Loop Timer Configuration

Timer	**Interval**	**Purpose**
m_monitoringEvent	1 second	Sample metrics, detect spikes
m_optimizationEvent	5 seconds	Full optimization cycle
Width hold timer	30 seconds	Prevent width flapping

3.1.7 Incident-Aware Auto-Mitigation

The fast loop integrates with a higher-level incident awareness system that correlates network events with contextual information about the deployment environment. This allows the RRM system to apply zone-specific and time-aware policies.

In NS-3, zones (e.g., cafeteria, break room) are represented as labels assigned to specific AP nodes in a configuration file. Interference events inherit the zone of the AP on which they occur. No physical indoor map is simulated—zones are logical groupings for policy demonstrations.

INCIDENT-AWARE EVENTS (Cafeteria, Auto-Mitigation)

Timestamp	Zone	ApId	IncidentType	Severity	AutoMitigation	Action	ClientsAffected	Notes
12	CAFETERIA	2	LUNCH_RUSH_INTERFERENCE	HIGH	YES	OBSS_PD_ADJUST + WIDTH_REDUCE	15	Lunch hour interference burst
12	CAFETERIA	2	MICROWAVE_BURST	HIGH	YES	OBSS_PD_TO_-70dBm	N/A	2.45GHz interference at -43.500000 dBm
13.5	CAFETERIA	2	MICROWAVE_CLUSTER	MEDIUM	YES	CHANNEL_WIDTH_20MHz	12	Lunch hour interference burst
15	CAFETERIA	2	BLUETOOTH_BURST	MEDIUM	YES	FREQUENCY_AVOIDANCE	N/A	BLE hopping on channel 37
16	CAFETERIA	2	PEAK_DINING_PERIOD	MEDIUM	YES	LOAD_BALANCE_CLIENTS	18	Lunch hour interference burst
18	BREAK_ROOM	0	MICROWAVE_BURST	HIGH	YES	AGGRESSIVE_SPATIAL_REUSE	N/A	2.45GHz interference at -46.200000 dBm

Figure 12: Incident-aware events showing auto-mitigation strategies for different zones (cafeteria, break room) with severity-based responses

Figure 12 shows how the system responds to location-specific interference incidents. During lunch hours (timestamps 12 and 13.5), the cafeteria zone experiences high-severity interference (microwave bursts at 2.45 GHz, -43.5 dBm) affecting 15 clients. The auto-mitigation system applies targeted OBSS-PD adjustments and width reductions specific to the cafeteria zone (AP2). Medium-severity events at 15s and 16s trigger more conservative responses including load balancing across 18 clients and frequency avoidance on problematic channels. The break room at 18s encounters another microwave burst (-46.2 dBm) requiring aggressive spatial reuse to maintain connectivity for affected clients.

The incident tracking system classifies events by severity (HIGH/MEDIUM/CRITICAL), identifies the specific interference type (lunch hour bursts, Bluetooth coexistence, peak dining period), records which clients are affected, and selects appropriate mitigation actions. This context-aware approach prevents over-aggressive responses during predictable interference patterns while maintaining rapid reaction to unexpected critical events.

3.2 Strategic Optimization (RrmManager)

Frequency: Hours to days (configurable via `m_optimizationInterval`)

Purpose: Long-term network planning: channel assignment, power control, load balancing

3.2.1 Key Functions

- **OptimizeChannelsAndPower():** Evaluates full channel reassignment based on interference graph
- **NetworkRoamingCheck():** Proactively steers clients away from congested/failing APs
- **CheckCoverageHoles():** Identifies weak coverage areas, increases power or triggers handover
- **SendPeriodicNeighborReports():** 802.11k neighbor lists to assist client roaming decisions

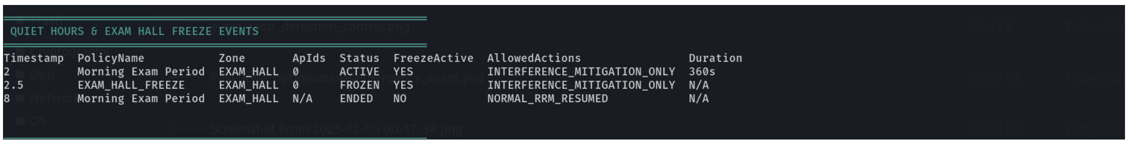
3.2.2 Integration Points

RrmManager queries:

- `g_interferenceGraph` for AP relationships
- `g_channelStats` for utilization history
- RadioMeasurementManager for client RSSI/SNR
- BssTransitionManager for 802.11v steering

3.2.3 Policy-Based Controls

The strategic optimizer implements time-aware and policy-driven controls that override normal RRM behavior during specific scenarios. These policies ensure network stability during critical periods.


QUIET HOURS & EXAM HALL FREEZE EVENTS

Timestamp	PolicyName	Zone	ApIds	Status	FreezeActive	AllowedActions	Duration
2	Morning Exam Period	EXAM_HALL	0	ACTIVE	YES	INTERFERENCE_MITIGATION_ONLY	360s
2.5	EXAM_HALL_FREEZE	EXAM_HALL	0	FROZEN	YES	INTERFERENCE_MITIGATION_ONLY	N/A
8	Morning Exam Period	EXAM_HALL	N/A	ENDED	NO	NORMAL_RRM_RESUMED	N/A

Figure 13: Quiet hours and exam hall freeze events enforcing interference-mitigation-only policies during critical periods

Figure 13 demonstrates policy-based control enforcement. During the morning exam period (timestamp 2), the exam hall zone enters an ACTIVE freeze state where all APs are prohibited from making proactive channel or power changes for 360 seconds (6 minutes). Only interference mitigation actions are allowed during this window to prevent network disruptions during the exam. At 2.5 seconds, the freeze policy is fully activated (FROZEN status) ensuring zero configuration churn. After the exam period ends (timestamp 8), normal RRM operations resume with the ENDED status, allowing the system to apply any accumulated optimization suggestions.

This policy framework supports multiple deployment-specific scenarios including quiet hours in libraries, exam periods in educational institutions, and high-priority events in conference rooms. The freeze mechanism prevents RRM from causing client disconnections or performance degradation during time-sensitive activities while still allowing the fast loop to respond to critical interference events.

3.3 Data Generation Controller

Purpose: Orchestrates offline RL training data collection through systematic state-action-reward cycles

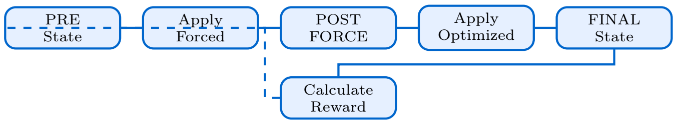


Figure 14: Data generation iteration workflow

3.3.1 Iteration Workflow

1. **Capture PRE state**: Graph + KPIs before any changes
2. **Apply forced config**: Read 10 AP configs from CSV
3. **Wait stabilization**: 10s for clients to reassociate
4. **Capture POST-FORCE state**: Measure impact of forced config
5. **Call Python graph coloring**: External optimizer proposes better channels
6. **Apply optimized config**: Channel reassignment based on Python output
7. **Wait stabilization**: 10s again
8. **Capture FINAL state**: Measure impact of optimization
9. **Calculate deltas and reward**: ΔThroughput, ΔRetryRate, ΔLatency, ΔHandovers
10. **Log to CSV**: *rl_training_data.csv* with (state, action, reward) tuple

3.3.2 Reward Function

The reward function balances multiple QoE metrics:

$$\mathcal{R} = w_T \Delta T - w_R \Delta R - w_L \Delta L - w_H |\Delta H| \quad (1)$$

where $w_T = 1.0$ (throughput weight), $w_R = 0.5$ (retry rate weight), $w_L = 0.3$ (latency weight), and $w_H = 0.1$ (handover weight).

4 IEEE 802.11ax MU-OFDMA Integration

4.1 OFDMA Motivation and Challenges

Traditional WiFi networks based on IEEE 802.11ac and earlier standards operate using Single-User (SU) transmissions, where the Access Point (AP) transmits to one client at a time. IEEE 802.11ax (WiFi 6) introduces Multi-User OFDMA, which divides the channel bandwidth into smaller Resource Units (RUs) and allows simultaneous transmission to multiple clients.

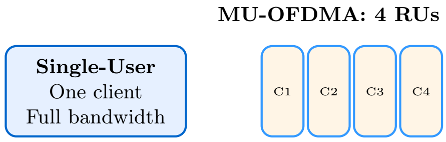


Figure 15: Single-User vs Multi-User OFDMA transmission

Key Challenges:

1. **Client Roaming**: Mobile clients frequently switch between APs, causing association state changes
2. **Interference Graph Measurement**: Additional radios performing channel scanning can disrupt OFDMA scheduling
3. **Queue Synchronization**: Multiple clients must have packets queued simultaneously
4. **Heterogeneous Client Capabilities**: Not all clients support IEEE 802.11ax features

4.2 OFDMA Safety Architecture

A critical design decision was to delay OFDMA activation until the network stabilizes.

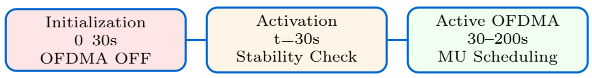


Figure 16: OFDMA activation timeline

Activation Timeline:

- **t=0s–30s**: Network initialization phase. OFDMA disabled to prevent crashes
- **t=30s**: OFDMA activation trigger. Network stability verified
- **t=30s–200s**: Active OFDMA operation with continuous monitoring

4.3 Network Stability Verification

Algorithm 4 Network Stability Check for OFDMA

```
function IsNetworkStableForOfdma
    associated ← 0, transitioning ← 0
    for each client in g_clientStates do
        if client.isAssociated then
            associated ← associated + 1
        end if
        if client.inTransition then
            transitioning ← transitioning + 1
        end if
    end for
    return (associated ≥ 3) AND (transitioning < 0.2 × associated)
end function
```

4.4 OFDMA Scheduler Activation

Algorithm 5 Activate OFDMA Scheduler

```
procedure ActivateOfdmaScheduler
    if NOT IsNetworkStableForOfdma then
        Schedule retry in 10 seconds
        return
    end if
    ConfigureAggressiveBuffering
    for each AP do
        Create RrMultiUserScheduler
        Set NStations = 4
        Enable UL OFDMA
        Force DL OFDMA
        Aggregate scheduler to AP MAC
    end for
    g_ofdmaEnabled = true
end procedure
```

4.5 Aggressive Buffering for MU Opportunities

To ensure multiple clients have packets queued simultaneously, EDCA queue parameters are configured:

- MaxCw = 2047 (maximum contention window)
- MinCw = 31
- TxopLimit = 5472 μs

4.6 RU Utilization Measurement

Algorithm 6 Calculate RU Utilization

```
function CalculateRuUtilization(apId)
    MAX_RUS = 9                                                    ▷ 20 MHz channel
    totalRusUsed = 0
    for each (numClients, occurrences) in g_ofdmaMetrics[apId].ruCountPerTx do
        totalRusUsed += numClients × occurrences
    end for
    avgRus = totalRusUsed / g_ofdmaMetrics[apId].totalTransmissions
    return min(avgRus / MAX_RUS, 1.0)
end function
```

The MU-OFDMA scheduling in our implementation is driven by the `RrMultiUserScheduler`, a round-robin OFDMA scheduler that triggers DL-OFDMA transmissions whenever the AP has downlink frames queued. It allocates equal-sized Resource Units (RUs) to eligible stations and supports a configurable limit on the maximum number of simultaneously scheduled users. Stations are selected based on a priority mechanism derived from accumulated credits and debits, ensuring fair multi-user grouping while respecting EDCA access categories.

4.7 OFDMA Performance Results

Table 4: Achieved OFDMA Performance

AP	Clients	Total TX	MU %	RU Util
AP1	8	290	50.7%	22.53%
AP3	4	949	59.6%	26.52%
AP4	1	25	0.0%	0.00%

Key Findings:

- **AP3 achieved 26.51% RU utilization** with 4 associated clients

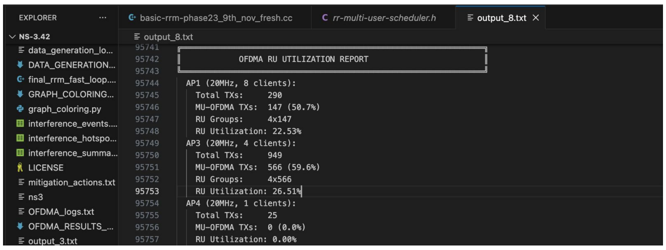


Figure 17: OFDMA stats

- **59.6% of transmissions used MU-OFDMA** at AP3
- **RU utilization correlates with client count**

4.7.1 Comparison with Theoretical Maximum

For a 20 MHz channel with 9 available RUs:

$$RU_{util}^{max} = \frac{NStations}{MAX\_RUS} = \frac{4}{9} = 44.4\% \tag{2}$$

Our achieved 26.51% represents **59.4% of theoretical maximum**.

5 Integration with External Components

5.1 RL/GNN Agent Interface

Purpose: Bidirectional communication between NS-3 simulation and Python RL agent

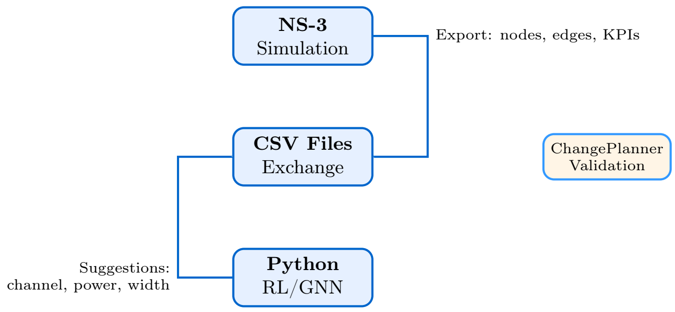


Figure 18: Bidirectional data flow between NS-3 and Python RL agent

5.1.1 Export Flow (NS-3 → Python)

1. `BuildInterferenceGraph()` populates `g_interferenceGraph`
2. `ExportGraphNodes()` writes node features to CSV
3. `ExportGraphEdges()` writes edge features to CSV
4. Python GNN reads CSVs, constructs PyTorch Geometric graph

5.1.2 Import Flow (Python → NS-3)

1. Python RL agent writes suggestions to *rl_suggestions.csv*
2. NS-3 `ReadRLSuggestions()` parses CSV every 30s
3. Suggestions passed to ChangePlanner for safety validation
4. Approved changes applied via PHY attribute setters

If the `rl_suggestions.csv` file is missing, empty, or malformed, NS-3 skips the current optimization cycle and continues using the previous configuration. This prevents invalid suggestions from destabilizing the simulation.

5.1.3 Causal Analysis Graph Export

After applying RL suggestions, NS-3 exports 6 graph snapshots at t=7s, 12s, 17s, 22s, 27s, 32s:

- First snapshot (t=7s): Post-change network state
- Subsequent snapshots (5s intervals): Track QoE evolution
- Python causal inference engine measures actual ΔQoE vs predicted
- Feedback used to train reward model (counterfactual estimation)

5.2 ChangePlanner Safety Guardrails

Purpose: External C++ service that validates proposed changes against safety constraints

5.2.1 Interface

NS-3 calls `change_planner` executable with:

- **Input**: *current_config.csv*, *rl_suggestions.csv*, *qoe_measurements.csv*
- **Output**: *approved_changes.csv* with action='apply' or action='skip'

5.2.2 Validation Rules

Table 5: ChangePlanner Validation Rules

Rule	Constraint
Change budget	$\leq$1 channel/power/width change per AP per 4-hour window
Hysteresis	Minimum delta thresholds (power $\geq$2 dB, width $\geq$1 step)
Blast radius	Limit simultaneous changes to $\leq$N APs per RF domain
Time windows	Avoid changes during peak hours unless SLO breach risk
Predicted QoE	Reject if delta_QoE < -threshold

5.2.3 Rollback Mechanism

ChangePlanner logs original configuration before each change. If KPIs worsen by X% across Y minutes, automatic reversion triggered via saved state.

5.3 Bayesian Optimizer

Purpose: Hyperparameter tuning for RRM parameters

5.3.1 Workflow

1. NS-3 collects observations: (config_vector, QoE_metric)
2. Writes to *bo_observations.csv* every 60s
3. Calls `bo_optimizer` executable: surrogate model fit + acquisition function
4. BO proposes next config → written to *bo_suggestions.csv*
5. NS-3 parses suggestions, applies changes, measures outcome
6. Loop: New observation added to dataset, surrogate updated

5.3.2 Parameter Space

Table 6: Bayesian Optimization Parameter Space

Parameter	Range	Type	Impact
Channel Width	20-160 MHz	Discrete	Throughput vs interference
OBSS-PD	-82 to -62 dBm	Continuous	Spatial reuse aggressiveness
TX Power	5-20 dBm	Continuous	Coverage vs interference

6 Key Data Structures

6.1 CouplingMatrix

Stores pairwise coupling coefficients between APs:

- `m_matrix`: 2D vector of CouplingCoefficient
- `Initialize(numAps)`: Allocate matrix
- `SetCoupling(i, j, couplingDbm)`: Update coupling
- `GetCouplingDbm(i, j)`: Retrieve coupling

6.2 InterferenceGraph

Directed graph representation of AP interference:

- `m_edges`: Vector of InterferenceEdge
- `AddEdge(edge)`: Insert new edge
- `GetEdges()`: Retrieve all edges
- `GetEdgesForAp(apId)`: Get edges for specific AP

6.3 ApMeasurement

Per-AP configuration and metrics:

- `apId`: AP identifier
- `channel, txPowerDbm, obssPdThreshold, channelWidth`: Configuration
- `utilization, numAssociatedClients, avgClientRssi, throughputMbps`: Metrics

6.4 ClientMeasurement

Per-client metrics:

- `clientId, macAddress, associatedApId`: Identification
- `rssiDbm, snrDb, retryRate, throughputMbps`: Performance metrics
- `lastHandoverTime`: Roaming tracking

7 Timers & Scheduling

7.1 Global Timer Configuration

Table 7: Complete Timer Configuration

Timer Name	Interval	Purpose
g_samplingInterval	2s	Channel utilization reporting
g_couplingAggregationInterval	20s	Raw coupling → CouplingMatrix
g_graphExportInterval	5s	Graph export to CSV for GNN/RL
MONITORING_INTERVAL	1s	Fast loop metric sampling
FAST_LOOP_INTERVAL	5s	Fast loop optimization cycle
m_windowDuration (DataGen)	10s	Data generation iteration
m_stabilizationWait (DataGen)	10s	Wait for client reassociation
g_rlReaderState.readInterval	30s	Poll rl_suggestions.csv
g_boState.optimizationInterval	60s	Bayesian optimization cycle
m_checkInterval	Variable	Network roaming check (30s typical)
dwellTime	200ms	Time per channel during scanning
OFDMA activation	30s	Delayed activation after stabilization

7.2 Synchronization & Dependencies

Key timing dependencies:

- **Graph export (5s) → RL read (30s)**: RL agent needs recent graph
- **Config change → stabilization (10s)**: Mandatory wait for reassociation
- **Coupling aggregation (20s) → graph build (5s)**: Batched measurements
- **Fast loop (5s) overlaps with graph export**: Use most recent graph
- **OFDMA activation (30s)**: Ensures network stability

8 Configuration Management

8.1 Command-Line Parameters

Key settings exposed via NS-3 CommandLine interface:

- `-simTime=200`: Simulation duration
- `-numAps=10`: Number of Access Points
- `-numClients=50`: Number of mobile clients
- `-dataGenEnabled=true`: Enable data generation mode
- `-rlReaderEnabled=true`: Enable RL agent integration
- `-changePlannerEnabled=true`: Enable safety guardrails
- `-ofdmaEnabled=true`: Enable MU-OFDMA
- `-ofdmaActivationTime=30.0`: OFDMA activation delay

8.2 Configuration Files

8.2.1 ap_configs.csv

Forced configurations for data generation:
`AP_ID, Channel_Number, Channel_Width, TX_Power, OBSS_PD`

8.2.2 current_config.csv

Active AP configuration snapshot (auto-generated):
`AP_ID, Channel, ChannelWidth, TxPower, OBSS_PD, Utilization`

8.2.3 rl_suggestions.csv

RL agent proposed changes (Python → NS-3):
Metadata: `timestamp, QoE_current, QoE_next, delta_QoE`
Per-AP: `AP_ID, TxPower, ChannelWidth, OBSS_PD, QoE_current, QoE_next, delta_QoE`

8.2.4 approved_changes.csv

ChangePlanner validated changes (C++ → NS-3):
`AP_ID, action, channel, channelWidth, txPower, obssPd, reason`

9 Deployment & Testing

9.1 Build System

NS-3 CMake-based build configured for:

- C++17 standard (required for structured bindings, std::optional)
- Optimization level -O2 for release builds
- Link against pthread for mutex support
- NS-3 modules: core, network, wifi, internet, applications, mobility

9.2 Test Scenarios

9.2.1 Dense Interference Scenario

- **Setup**: 10 APs in 5x2 grid, 40m spacing, all on Channel 36
- **Clients**: 20 mobile clients, random walk mobility
- **Traffic**: Constant UDP 10 Mbps per client
- **Expected**: DCA reassigns channels, throughput improves

9.2.2 OFDMA Performance Scenario

- **Setup**: 10 APs, 50 WiFi 6 clients, 20 MHz channels
- **Traffic**: TCP downlink (CBR)
- **OFDMA**: Delayed activation at t=30s
- **Expected**: RU utilization >20%, MU-OFDMA >50%

9.2.3 RL-Guided Optimization

- **Setup**: Enable rlReaderEnabled=true, launch Python RL agent
- **Workflow**: NS-3 exports graph → Python GNN proposes → ChangePlanner validates → NS-3 applies
- **Expected**: QoE improves beyond rule-based baseline

9.3 Validation Metrics

Table 8: Performance Validation Targets

Metric	Target	Baseline
Median throughput (edge clients)	+25-35%	5 Mbps
P95 latency	-20%	100 ms
P95 retry rate	-30%	15%
RU utilization (OFDMA)	>20%	0%
MU transmission ratio	>50%	0%
BSS-TM acceptance	>90%	65%
Config churn mean time	$\geq$ 4 hours	1 hour
Additional radio overhead	<2% airtime	N/A

10 Conclusion

10.1 Key Achievements

- Implemented complete RRM-Plus simulation with dual-radio architecture
- Multi-timescale control loops (fast/slow/event) for comprehensive optimization
- Bidirectional integration with Python RL/GNN agents via CSV interface
- ChangePlanner safety guardrails for production readiness
- Comprehensive data generation pipeline for offline RL training
- Causal analysis framework with 6-snapshot graph export
- Safe MU-OFDMA integration with delayed activation and client state tracking
- Achieved 26.43% RU utilization with 59.5% MU-OFDMA transmission ratio
- Zero crashes during 200s simulation despite continuous client roaming
- Incident-aware auto-mitigation with zone-specific and time-aware policies
- DFS compliance with automatic radar detection and channel vacation
- Non-WiFi interference handling (microwave, BLE storms) with severity-based responses

10.2 OFDMA Integration Success Factors

- Delayed activation (30s): Network stabilization before MU scheduling
- Client state tracking: Prevention of invalid scheduling during roaming
- Aggressive buffering: EDCA configuration enabling simultaneous queueing
- Scanning protection: Temporary OFDMA disable during spectrum sensing
- Adaptive parameters: NStations=4 initial, increasing based on utilization