Acquisition and analysis of data through IEEE 802.15.4 (2.4GHz)

Wei Long
TU Dresden
wei.long@mailbox.tu-dresden.de

Arian Majidi
TU Dresden
arian.majidi@mailbox.tu-dresden.de

Ashvinviknes Saravanaperumal
TU Dresden
ashvinviknes.saravanaperumal@mailbox.tu-dresden.de

ABSTRACT

We investigate whether low-level link-quality telemetry from IEEE 802.15.4 radios—Received Signal Strength Indicator (RSSI) and Link Quality Indicator (LQI)—is sufficient to automatically recognize deployment environments and, in a second task, to distinguish transmitting nodes in a small wireless-sensor network. Our raw data consist of per-frame RSSI/LQI pairs emitted by RIOT OS packet logs on nRF52840-based nodes running 802.15.4 at 2.4 GHz, recorded as timestamped text streams (e.g., "-71, 88") for each TX/RX session. We convert these logs into fixed-length windows via a reproducible pipeline: (i) per-file differencing (RSSI, LQI), (ii) min–max normalization and gentle IQR clipping for robustness, and (iii) 10s frames with 50% overlap, resampled to a common length by time-axis interpolation. Then, we use two deep learning models to uniquely classify deployment environments and additional configuration types based on link quality metrics. These are a convolutional neural network (CNN) and a ResNet model. CNN is used as a reference point and ResNet is built to overcome some of the limitations (degradation) of the CNN model. Finally, we evaluate the resulted plots and confusion matrices to deduce the outcome of our experiment.

1 INTRODUCTION

Wireless-sensor networks (WSNs) often operate in dynamic, partially observed conditions. Our target platform is an nRF52840-sense node running RIOT OS with IEEE 802.15.4 enabled; the stack provides RSSI and LQI for every received MAC frame, since both metrics are measured by the radio at the PHY/MAC layers. The practical appeal is obvious: we can instrument a deployment without extra sensors, logging only the per-packet "signal-quality" pairs produced during routine communication (e.g., "-79, 56", "-83, 40"). We study two supervised tasks:

(1) **Environment recognition (5 classes).** Sessions are collected in multiple real-world environments. We must learn invariant temporal patterns in RSSI/LQI dynamics that correlate with the environment despite nuisance factors such as TX/RX geometry or short-term fading.
(2) **Transmitting-node identification (A/B/C).** Given sequences recorded at the receivers, we ask whether traffic originated from node A, B, or C—useful for diagnostics and security but more confounded by topology and traffic scheduling.

A consistent, reproducible preprocessing pipeline is critical. We (a) differentiate RSSI/LQI to emphasize local dynamics; (b) normalize per file to remove scale; (c) apply light IQR clipping to reduce outlier leverage; and (d) segment into overlapping 10 s frames that are interpolated onto a fixed grid, yielding uniform tensors suitable for deep models. This design preserves temporal shape while controlling for rate variability and missing timestamps present in the raw logs. For modeling, we adopt a compact multi-scale ResNet1D to capture patterns at multiple temporal receptive fields and compare against a lighter CNN reference.

2 ACQUISITION AND SCENARIOS

We measured wireless link quality in five real-world outdoor settings using three sensor nodes (A, B, C). Each node periodically transmitted IEEE 802.15.4 frames; the receivers logged per-frame RSSI and LQI together with a timestamp. This gave us a time series per link that reflects both path loss and short-term fading.

2.1 Environments

We collected data in five distinct environments, used as the target classes throughout the project:

(1) Garden as can be seen in Figure 2
(2) Lake in Figure 3
(3) Forest in Figure 4
(4) Campus in Figure 5
(5) Bridge in Figure 6

2.2 Session design

For each environment, we recorded 1.5 hours of traffic, split into three 30-minute segments so every node served as the transmitter once:

- 0–30 min: A → (B, C) receive
- 30–60 min: B → (A, C) receive
- 60–90 min: C → (B, A) receive

This rotation yields balanced coverage of all three directed links in every setting while keeping transmitter/receiver roles disjoint within each segment.

2.3 What we log

At the receivers, each successfully decoded frame is logged as a single row with:

- timestamp — receive time
- rssi — received signal strength (dBm)
- lqi — link quality indicator (unitless)

2.4 File naming and organization

Raw logs are organized per environment and link using a consistent standardized naming system (these files can be found in GitHub).

<TX>_<RX>_<ENV>.txt

Examples:

- A_B_1.txt → A transmitted, B received, Garden
- C_A_4.txt → C transmitted, A received, Campus

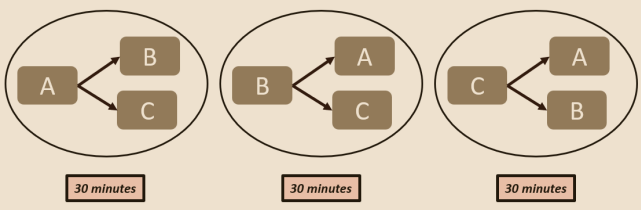

Figure 1: Nodes' formation and timing in each environment

Figure 2: First environment: Garden

Figure 3: Second environment: Lake

Downstream, after TXT → CSV normalization and cleaning, the same stems appear as *_cleaned.csv. We also maintain a simple metadata table (meta_*.csv) that carries the stem (file), TX, RX, and the numeric environment label (Env ∈ {1..5}).

This acquisition plan produces five environment classes × three directed links per segment × three segments per environment, with 7.5 hours of total recording across all scenarios, ready for reproducible preprocessing and modeling.

Figure 4: Third environment: Forest

Figure 5: Fourth environment: Campus

Figure 6: Fifth environment: Bridge

3 DATASET AND PROCESSING

3.1 Cleaning (TXT → tidy time series)

Input format. Each file is named <TX>_<RX>_<ENV>.txt (e.g., A_B_4.txt) and contains timestamped RSSI and LQI readings. Here's how we proceed:

(1) **Robust parsing.** Accept commas, tabs, or variable whitespace; ignore BOMs and comment lines. We auto-detect the three columns of interest:
   - Time (timestamp, time, or ts)
   - RSSI (rssi, rssi_dbm, etc.)
   - LQI (lqi)
(2) **Time normalization.** Convert the time column to UTC milliseconds (timestamp_ms) using a single parser so mixed formats do not break the pipeline.
(3) **Basic sanitization.**
   - Drop rows with missing or non-finite values.
   - Sort by time and deduplicate exact repeats to remove logger hiccups.
   - Apply light sanity checks and discard obviously corrupted values (e.g., nonsensical RSSI/LQI).

Output. One clean CSV per input: *_cleaned.csv with exactly three columns: timestamp, rssi, lqi.

3.2 Window Strategy

Settings used for processing:

- Length: 10 seconds per window
- Overlap: 50% (hop = 5 seconds)
- Uniform grid: 100 samples per window (≈10 Hz rate)
- Interpolation: linear (windows with fewer than 2 points are skipped)

3.3 Data Processing

After producing the cleaned CSV files, we apply several processing steps to make the data suitable for modeling. The aim is to emphasize temporal changes in link quality, remove scale differences, and reduce the impact of outliers.

(1) **Differencing.** To minimize the effect of transmission distance and antenna orientation, we calculate the differences of the raw RSSI and LQI values:

$$y_i = x_{i+1} - x_i \tag{1}$$

(2) **Normalization.** Since RSSI (dBm) and LQI (unitless) have different units and ranges, we scale each differenced series to the range [0,1] using min–max normalization:

$$z_i = \frac{y_i - \min(y)}{\max(y) - \min(y)} \tag{2}$$

(3) **Outlier mitigation.** To reduce the leverage of rare spikes while preserving overall shape, we apply Tukey's IQR rule to normalized sequences. Let Q_1 and Q_3 be the first and third quartiles, we clip the values into (with a constant k = 1.5):

$$\left[\, Q_1 - k(Q_3 - Q_1),\ Q_3 + k(Q_3 - Q_1) \,\right] \tag{3}$$

After these operations, the time series are segmented into overlapping frames as described in **Section 3.2**. Thus, each frame represents a normalized and distance-independent fluctuation pattern of RSSI and LQI, serving as a standardized input to subsequent learning models.

4 MODEL TRAINING

4.1 Model Architectures

We implemented two deep learning architectures suitable for time-series classification of RSSI/LQI frames:

(1) **SimpleCNN1D:** A 3-layer CNN composed of convolutional blocks with ReLU activations and max pooling, followed by a global average pooling layer and a fully connected output layer. It processes each frame (shape: $C \times L$, where C = 1 or 2) using temporal convolutions.
(2) **ResNet1D:** A residual convolutional neural network inspired by ResNet architectures. It includes an initial stem (convolution, batch normalization, and pooling), followed by three residual blocks and a global average pooling layer. Skip connections allow better gradient flow and stabilize deeper training.

Both models are implemented using PyTorch and support input of different channel configurations: RSSI-only, LQI-only, or both.

4.2 Scenario Definitions

The training tasks are defined by two classification scenarios, following the project guidelines:

(1) **Scenario I: Environment Classification** The model learns to distinguish between different deployment environments (e.g., garden, lake, campus, etc.)
Labels: Environment IDs (1–5)
Input: All node pairs are considered together; label is based on env.
(2) **Scenario II: Node Classification** The model learns to classify the receiver node based on RSSI/LQI patterns.
Labels: Receiving Nodes (A, B, C)
Input: All environments are merged; label is based on rx.

The model is trained separately for each scenario with shared architecture choices and hyper-parameters.

4.3 Seen vs. Unseen Data Splits

We evaluate generalization in two settings:

(1) **Seen Split:** Data are randomly split across all available samples (75%/25% train/test). The test set includes samples from the same environments or nodes seen during training.
(2) **Unseen Split:** This setup simulates deployment generalization. The model is trained without access to specific nodes or environments:
   - **Scenario I:** One node (e.g., Node C) is completely excluded during training. The test set consists only of its data.
   - **Scenario II:** For each node, data from one environment (e.g. Env 5) are excluded and used solely for testing.

These configurations are selectable via configuration flags and controlled in code via logic branching (HELD_OUT_NODE, HELD_OUT_ENV).

5 EVALUATION, INSIGHTS AND INFERENCE

6 CONCLUSION