PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
الجمهورية الجزائرية الديمقراطية الشعبية
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
وزارة التعليم العالي والبحث العلمي
École supérieure en Sciences et Technologies de l'Informatique et du Numérique
المدرسة العليا في علوم وتكنولوجيات الإعلام الآلي بجاية



Dissertation Submitted to the Department of 2nd Cycle Training in Partial Fulfillment of the Requirements for Engineer's Degree in Computer Science

Specialty: Cybersecurity

Submitted By:
Rahma Mihoub

Data-Driven Automatic Calibration of IoT Network Simulators

Supervised by:

Dr. Samir Si-Mohammed
Université de Lorraine

Dr. Maroua Daoudi
ESTIN

Members of jury:

• **Dr.** Boughaba Houda	President	ESTIN
• **Dr.** Zedek Rezika	Examiner	ESTIN
• **Dr.** Bechar Amine	Examiner	ESTIN

Academic year: 2025/2026

Dedication

سُبْحَانَكَ لَا عِلْمَ لَنَا إِلَّا مَا عَلَّمْتَنَا إِنَّكَ أَنتَ الْعَلِيمُ الْحَكِيمُ

(Glory be to You; we have no knowledge except what You have taught us. Indeed, it is You who is the Knowing, the Wise. – Qur'an 2:32)

To Allah the Almighty, for the countless blessings, guidance, and strength granted to me throughout this journey and beyond. I pray that this work may be a source of beneficial knowledge to others.

To my family, who believed in me and saw my potential since I was a child. Without you, I would not be who I am today. Thank you for making me feel special since day one, and for the immense trust you have placed in me.

To my father, for the immense sacrifices you made to raise us, for your tenderness, and for your unwavering trust in me. You have always been my role model in determination and perseverance, a spirit I am proud to have inherited from you.

To my mother, the kindest soul I know, for your unconditional love, warmth, and endless guidance, and for being not only the best mother, but also my best friend.

Acknowledgements

I would like to take a moment to express my deepest gratitude to all those who contributed, in one way or another, to the completion of this work. Their guidance, patience, encouragement, and trust have been a constant source of strength, allowing me to overcome the difficulties encountered throughout this academic journey.

I would like to thank the members of the jury for accepting to evaluate this work and for the time and attention dedicated to reviewing it.

I would also like to thank my school ESTIN for giving me this opportunity, and Université de Lorraine for hosting me throughout this internship.

My deepest thanks go to my supervisor at CRAN, Dr. Samir Si-Mohammed. No words can fully express how thankful I am for your trust in me and for so generously dedicating your time and energy to this work. Thank you for your continuous guidance, both professional and personal, and for making sure I got the most out of this experience. I am especially grateful for the countless opportunities you offered me. Thank you for always pushing me to go beyond my limits, for your constant words of encouragement, and simply for being the greatest supervisor I could have hoped for.

I would also like to thank Dr. Maroua Daoudi, my supervisor at ESTIN, for this opportunity, for her recommendation, and for believing in me. Thank you for all your efforts and guidance, without which this internship would not have been possible.

My thanks also go to the entire CRAN team, who welcomed me with open arms and included me at "the Cool Seniors' Table": Mohamed, Anouar, Jana, Carin, Chahrazed, Abdessamed, Zaid, and others. A special thanks goes to Ghada, for her kindness and support throughout this experience. Thank you all for your humor, and the well-deserved long lunch breaks we shared, it was a genuine pleasure getting to know you, even for such a short time.

I would also like to thank Dr. Fabrice Theoleyre for his invitation, and all the wonderful Network Team of the ICube laboratory for their warm welcome during my invited seminar there.

On a more personal note, I would like to thank my sister, Naila, my rock and my childhood friend. Thank you for listening to hours of me rambling and for keeping up with my endless stories. Thank you for always reminding me of who I am and for setting the bar high, encouraging me to aim further in every aspect of life.

To my little brother, Mohammed Amine, my favorite partner in questionable jokes and endless laughter, thank you for always being there to make life a little lighter. I am truly grateful to have grown up alongside you and to call you my brother.

To my partners throughout this journey, Houda and Aya, thank you for making ESTIN feel like a second home. Thank you for your continuous support, your time, and your genuine company, and to everyone else who made this college experience unforgettable.

To my lifelong friend and sister, Tasnim, thank you for your unconditional support and for believing in my potential since our teenage years. And to your family, who has always treated me as one of their own.

Finally, I extend my sincere thanks to everyone who, directly or indirectly, contributed to the completion of this work.

Abstract

Network simulators are widely used in Internet of Things (IoT) research, yet a persistent gap exists between simulated and real-world network behavior, mainly due to default radio model parameters that fail to capture real propagation conditions. This engineering thesis aims to build and validate a complete calibration pipeline that closes this gap by automatically adjusting the radio model of the Cooja/Contiki-NG simulator to match real deployment conditions. Three parameters of the LogisticLoss radio model were selected through a literature-guided analysis combined with correlation and Partial Dependence Plots, then calibrated using a Multilayer Perceptron (MLP) trained on a dataset combining Cooja simulations and FIT IoT-LAB experiments; an adaptive extension based on Page-Hinkley change detection was further designed to automatically trigger recalibration when network conditions change. The calibration solution was validated on multiple real deployment scenarios, showing a significant reduction of the simulation-to-reality gap, while the adaptive extension was successfully validated on both synthetic and real interference scenarios. These results yield a reliable calibrated digital twin, opening the way to explore what-if network scenarios for deployment planning and optimization.

Keywords: Network Digital Twin, Radio Model Calibration, Machine Learning, Adaptive Calibration, Cooja, FIT IoT-LAB.

Résumé

Les simulateurs de réseaux sont largement utilisés dans la recherche internet des objets (IoT), mais un écart persiste entre le comportement simulé et réel des réseaux, principalement dû à des paramètres de modèle radio par défaut qui ne reflètent pas les conditions réelles de propagation. Ce mémoire d'ingénieur vise à construire et valider un pipeline complet de calibration comblant cet écart en ajustant automatiquement le modèle radio du simulateur Cooja/Contiki-NG pour correspondre aux conditions réelles de déploiement. Trois paramètres du modèle radio LogisticLoss ont été sélectionnés à travers une analyse guidée par la littérature combinée à une corrélation et des Partial Dependence Plots, puis calibrés à l'aide d'un Multilayer Perceptron (MLP) entraîné sur un jeu de données combinant simulations Cooja et expériences FIT IoT-LAB ; une extension adaptative basée sur la détection de changement Page-Hinkley a ensuite été conçue pour déclencher automatiquement une recalibration lors de changements des conditions réseau. La solution de calibration a été validée sur plusieurs scénarios réels de déploiement, montrant une réduction significative de l'écart simulation-réalité, tandis que l'extension adaptative a été validée avec succès sur des scénarios synthétiques et réels d'interférence. Ces résultats aboutissent à un jumeau numérique calibré et fiable, ouvrant la voie à l'exploration de scénarios réseau hypothétiques (what-if) pour la planification et l'optimisation du déploiement.

Mots-clés : Jumeau Numérique de Réseau, Calibration de Modèle Radio, Machine Learning, Calibration Adaptative, Cooja, FIT IoT-LAB.

ملخص

تُستخدم محاكيات الشبكات على نطاق واسع في أبحاث إنترنت الأشياء، إلا أن فجوة مستمرة توجد بين السلوك المحاكى والواقعي للشبكات، ناتجة أساسًا عن معاملات نموذج الراديو الافتراضية التي لا تعكس ظروف الانتشار الحقيقية. تهدف هذه المذكرة الهندسية إلى بناء والتحقق من صحة خط أنابيب معايرة كامل يسد هذه الفجوة من خلال ضبط نموذج الراديو لمحاكي Cooja/Contiki-NG تلقائيًا ليتوافق مع ظروف النشر الحقيقية. تم اختيار ثلاثة معاملات لنموذج الراديو LogisticLoss من خلال تحليل موجه بالأدبيات مدمج مع الارتباط ورسوم Dependence Partial ،Plots ثم معايرتها باستخدام شبكة Perceptron Multilayer (MLP) مدربة على مجموعة بيانات تجمع بين محاكاة Cooja وتجارب FIT IoT-LAB؛ كما تم تصميم امتداد تكيفي يعتمد على كشف التغير Page-Hinkley لإطلاق إعادة المعايرة تلقائيًا عند تغير ظروف الشبكة. تم التحقق من صحة حل المعايرة على عدة سيناريوهات نشر حقيقية، مما أظهر انخفاضًا ملحوظًا في فجوة المحاكاة والواقع، بينما تم التحقق من صحة الامتداد التكيفي بنجاح على سيناريوهات اصطناعية وحقيقية للتداخل. تُفضي هذه النتائج إلى توأم رقمي معاير وموثوق، مما يفتح المجال لاستكشاف سيناريوهات الشبكة الافتراضية (what-fi) لأغراض التخطيط والتحسين في النشر.

الكلمات المفتاحية: التوأم الرقمي للشبكة، معايرة النموذج الراديوي، التعلم الآلي، المعايرة التكيفية، Cooja، FIT IoT-LAB.

Contents

General Introduction **1**

1 Background **3**
1.1 Introduction . . . 3
1.2 WSN and IoT Fundamentals . . . 3
1.2.1 The FIT IoT-LAB Testbed . . . 4
1.2.2 Network Simulation in the Development Workflow . . . 5
1.3 Radio Propagation Modeling . . . 6
1.3.1 From Free-Space Path Loss to Log-Normal Shadowing . . . 6
1.3.2 Radio Medium Models in Cooja . . . 7
1.3.3 The LogisticLoss Radio Medium . . . 8
1.3.4 Key Performance Indicators . . . 9
1.4 The Simulation-to-Reality Gap and the Calibration Problem . . . 10
1.5 State of the Art Synthesis and Positioning . . . 12
1.6 Conclusion . . . 13

2 Proposed Method **14**
2.1 Introduction . . . 14
2.2 Problem Statement . . . 15
2.3 Methodology . . . 16
2.3.1 Pipeline Architecture Overview . . . 16
2.3.2 Parameter Selection Analysis . . . 18
2.3.3 Dataset Generation Approach . . . 21
2.3.4 Data Processing . . . 26
2.3.5 Model Development . . . 27
2.3.6 Adaptive System Design . . . 29
2.3.7 Evaluation Approach . . . 33
2.4 Conclusion . . . 34

3 Implementation and Results **35**
3.1 Introduction . . . 35
3.2 Development Environment . . . 35
3.3 Implementation Details . . . 36
3.3.1 Per-Topology Training . . . 36
3.3.2 MLP Training Configuration . . . 36

3.3.3 Feature and Target Scaling . . . 38
3.4 Evaluation Metrics . . . 38
3.5 Results . . . 39
3.5.1 ML Model Results . . . 39
3.5.2 Static Calibration Validation Against FIT IoT-LAB . . . 44
3.5.3 Adaptive Validation: Synthetic Data . . . 48
3.5.4 Adaptive Validation: Real Interference Scenario . . . 50
3.5.5 Loss-Rate Distribution . . . 57
3.6 Discussion . . . 57
3.7 Conclusion . . . 58

Conclusion and Perspectives **60**
Key Insights and Contributions . . . 60
Limitations . . . 61
Future Work . . . 61

List of Figures

1.1 Distribution of M3 nodes at the Grenoble site 5
1.2 M3 node installed under a false ceiling 5
1.3 Radio irregularity vs. UDGM's circular range 7
1.4 LogisticLoss's PRR-vs-RSSI curve 9

2.1 Forward vs. inverse problem 15
2.2 Two-part pipeline architecture 17
2.3 Parameter-KPI Spearman correlation matrix 19
2.4 Partial dependence plots per parameter 20
2.5 SLURM parallelization of the dataset sweep 25
2.6 MLP architecture 27
2.7 Page-Hinkley detection mechanism 30
2.8 Three-phase adaptive recalibration loop 32

3.1 Development stack overview 36
3.2 Training/validation loss curves for all three topologies 41
3.3 Scenario 1 KPI distributions 42
3.4 Scenario 2 KPI distributions 43
3.5 Scenario 3 KPI distributions 44
3.6 Scenario 1 physical deployment 45
3.7 Scenario 2 physical deployment 45
3.8 Scenario 3 physical deployment 47
3.9 Synthetic 4-regime metric evolution 48
3.10 Synthetic packet loss evolution across regimes 49
3.11 Interference scenario metric evolution 52
3.12 Interference experiment topology 53
3.13 Cooja GUI during a recalibration 54
3.14 Real interference packet loss evolution 55
3.15 Monitoring UI: full pipeline state 56

List of Tables

1.1 Overview of Cooja's radio medium models 8
1.2 The three calibrated LogisticLoss parameters 8
1.3 Comparison of the CC2420 (Z1/Cooja) and AT86RF231 (M3/FIT IoT-LAB) radio chips 11

2.1 Parameter sweep ranges and step sizes 22
2.2 Data excerpt from the generated dataset 23
2.3 SLURM server specifications 25

3.1 The three validation topologies 37
3.2 MLP training configuration 37
3.3 Held-out regression performance per topology 39
3.4 Baseline model comparison across all three topologies 40
3.5 Scenario 1 behavior-space validation 45
3.6 Scenario 2 behavior-space validation 46
3.7 Scenario 3 behavior-space validation 47
3.8 Synthetic 4-regime calibration events 50
3.9 Interference experiment setup 51
3.10 Interference experiment phases 51
3.11 Interference experiment detection events 56

List of Abbreviations

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks.

ADAM Adaptive Moment Estimation.

ADWIN Adaptive Windowing.

AWGN Additive White Gaussian Noise.

CPU Central Processing Unit.

CSMA/CA Carrier-Sense Multiple Access with Collision Avoidance.

DDM Drift Detection Method.

DGRM Directed Graph Radio Medium.

EDDM Early Drift Detection Method.

EMA Exponential Moving Average.

GPU Graphics Processing Unit.

IEEE Institute of Electrical and Electronics Engineers.

IoT Internet of Things.

IP Internet Protocol.

KPI Key Performance Indicator.

MAE Mean Absolute Error.

ML Machine Learning.

MLP Multilayer Perceptron.

MRM Multi-path Ray-tracer Medium.

MSE Mean Squared Error.

NN Neural Network.

OS Operating System.

PDP Partial Dependence Plot.

PLE Path Loss Exponent.

PRR Packet Reception Rate.

RAM Random Access Memory.

ReLU Rectified Linear Unit.

RMSE Root Mean Squared Error.

RPL Routing Protocol for Low-Power and Lossy Networks.

RSSI Received Signal Strength Indicator.

RX Receive.

SLICES-RI SLICES Research Infrastructure.

SLURM Simple Linux Utility for Resource Management.

SSH Secure Shell.

TER Travail d'Étude et de Recherche.

TX Transmit.

UDGM Unit Disk Graph Model.

UDP User Datagram Protocol.

WSN Wireless Sensor Network.

General Introduction

Wireless Sensor Networks (WSNs) now form the physical backbone of most IoT deployments, connecting resource-constrained devices over wireless links to support applications ranging from environmental monitoring to industrial automation. Designing, testing, and validating protocols and applications directly on hardware at scale is slow, costly, and difficult to reproduce, which is why network simulation has become a standard step in the IoT development pipeline. Simulators such as Cooja (Contiki-NG Cooja Simulator, 2024), bundled with the Contiki-NG (Contiki-NG Documentation, 2024) operating system, offer a particular advantage here: they run the exact same *firmware* (a type of software designed to provide low-level control of hardware) that would be flashed onto physical motes, so only the radio channel between nodes needs to be modeled and substituted for a real one, while the operating system, protocol stack, and application logic execute unmodified.

That substitution, however, is exactly where the trouble starts. A simulator is only as trustworthy as the model standing in for the real channel, and radio propagation models left at their default configuration are known to diverge from real channel behavior, since default parameters describe a generic environment rather than the specific conditions of any deployment site. A prior study (Piveteau-Wernert, 2025) measured this divergence directly on the FIT IoT-LAB testbed (Adjih et al., 2015) under Cooja's simplest radio model, and found a gap of roughly 30% between simulated and real network behavior. This is the simulation-to-reality gap this thesis sets out to close, and it is worth being precise about what it actually reflects: since the same firmware runs on both sides, the gap is not a hardware or protocol mismatch, it is a radio model fidelity problem, and closing it is fundamentally an inverse problem: given key performance indicators observed on a real deployment, recover the radio model parameters that make the simulator reproduce that same behavior.

Manually tuning several interacting radio parameters by trial and error to match observed network behavior is impractical, particularly since the mapping between parameters and resulting behavior is nonlinear and the parameters do not act independently of one another. This motivates a data-driven approach instead: rather than deriving the inverse mapping analytically, a regression model can be trained to learn it directly from data generated by running the simulator many times over a wide range of parameter values. But a calibration performed once is not the end of the story. Real deployments are not static, interference sources appear and disappear, batteries drain, and environmental conditions change over time, so a *digital twin* (a virtual representation of a real deployment that functions as its digital counterpart (Khan et al., 2022)) calibrated once and left unchanged will drift away from the real network it is meant to represent,

however accurate that initial calibration was. Closing the gap, in other words, is not enough on its own; keeping it closed as conditions evolve is a second problem this thesis has to address alongside the first.

This engineering thesis builds and validates a complete calibration pipeline addressing both problems for Cooja's LogisticLoss radio model (Contiki-NG LogisticLoss Source, 2024). Three of LogisticLoss's parameters, the Path Loss Exponent (PLE), the Received Signal Strength Indicator (RSSI) Inflection Point, and the Additive White Gaussian Noise (AWGN) sigma, are selected for calibration through a literature-guided analysis combined with correlation and Partial Dependence Plot analysis, then calibrated using a Multilayer Perceptron trained on a dataset combining Cooja simulations and real FIT IoT-LAB experiments. An adaptive extension, built around a two-sided Page-Hinkley change detector (Page, 1954), is designed on top of this static calibration to automatically trigger recalibration whenever real network conditions shift enough to make the current calibration stale, keeping the digital twin synchronized with its real counterpart over time rather than accurate only at the single moment it was calibrated.

The remainder of this thesis is organized as follows:

- **Chapter 1** introduces the technical background needed to situate this work: WSN and IoT fundamentals, the Cooja/Contiki-NG simulation environment and the FIT IoT-LAB testbed, radio propagation modeling and the LogisticLoss radio medium specifically, and a synthesis of the state of the art positioning the gap this thesis addresses.
- **Chapter 2** presents the proposed method in full: how the three calibration parameters were selected, how the training dataset was generated and processed, how the MLP-based calibration model was designed, and how the adaptive extension was built on top of it.
- **Chapter 3** presents the implementation of both components and the experimental results validating them against real FIT IoT-LAB deployments, in parameter space, in behavior space, and under both synthetic and real changing network conditions.

The thesis closes with a general conclusion drawing these contributions together against the objectives stated here, and outlining the most immediate directions for continuing this work.

Chapter 1

Background

1.1 Introduction

The general introduction positioned this thesis's objective: closing the simulation-to-reality gap in Cooja/Contiki-NG, framed as a radio model fidelity problem, and keeping that calibration valid as real deployments change over time. Before presenting the method that does this in Chapter 2, this chapter lays the technical groundwork it builds on.

Section 1.2 reviews WSN and IoT fundamentals together with the role of network simulation in the development workflow, and presents the Cooja/Contiki-NG simulation environment and the FIT IoT-LAB physical testbed used throughout this work. Section 1.3 turns to radio propagation modeling, with emphasis on the LogisticLoss radio medium used in Cooja and the Key Performance Indicators (KPIs) used throughout this thesis to quantify network behavior. Section 1.4 frames the simulation-to-reality gap as a radio model fidelity problem, and Section 1.5 synthesizes the state of the art and identifies the gap this thesis addresses.

1.2 WSN and IoT Fundamentals

A WSN is composed of battery-powered, memory- and compute-constrained devices, commonly referred to as motes or nodes, that sense their environment and relay data toward one or more collection points, typically over multi-hop wireless links. IoT deployments build on this foundation by connecting such networks to broader services, ranging from environmental monitoring and industrial automation to smart agriculture and building management. The resource constraints inherent to WSN hardware, limited energy, memory, and processing power, mean that communication protocols must be designed with efficiency as a first-class concern, which in turn makes the wireless channel itself a critical and often underestimated factor in overall network performance: Dezfouli et al. (2015) show that a protocol which is efficient on paper can still perform poorly if the channel model used to design or evaluate it does not reflect real propagation conditions.

Contiki-NG is an open-source operating system designed for such constrained IoT devices (Contiki-NG Documentation, 2024), providing a lightweight networking stack that includes the Routing Protocol for Low-Power and Lossy Networks (RPL) (Winter et al., 2012) for multi-hop

mesh networking, IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) adaptation (Kushalnagar et al., 2007), and Carrier-Sense Multiple Access with Collision Avoidance (CSMA/CA) medium access control (IEEE, 2020) for contention-based channel access. Cooja is the network simulator bundled with Contiki-NG, allowing developers to run the exact same firmware that would be flashed onto physical motes inside a simulated environment (Contiki-NG Cooja Simulator, 2024). This is a significant advantage over purely mathematical or trace-based simulators, since the application and networking logic under test is not reimplemented for simulation purposes: only the physical radio channel between simulated nodes is modeled and substituted for a real one, while the operating system, protocol stack, and application code execute unmodified.

That substitution is precisely where fidelity issues arise. Cooja offers several interchangeable radio medium models [1] to emulate wireless propagation between simulated nodes, ranging from simple deterministic models to statistically grounded models such as LogisticLoss, detailed in Section 1.3. Whichever model is chosen, its parameters determine how closely the simulated channel resembles a real deployment, and default parameter values are rarely representative of any specific real-world environment, since they are chosen to be generically reasonable rather than tuned to a particular site.

1.2.1 The FIT IoT-LAB Testbed

To validate simulation results against real conditions, this work relies on FIT IoT-LAB, a large-scale, open-access testbed for IoT experimentation offered as part of the SLICES Research Infrastructure (SLICES-RI) [2]. The Grenoble site of FIT IoT-LAB hosts several hundred M3 open nodes, each equipped with an AT86RF231 IEEE 802.15.4 radio transceiver (Atmel Corporation, 2014), deployed across multiple building floors under a false ceiling (FIT IoT-LAB Grenoble Deployment, 2024). Researchers can reserve nodes, flash arbitrary firmware, and collect experiment data (serial logs, radio statistics, energy consumption) through a web interface and command-line tools, without needing physical access to the hardware.

Figure 1.1 shows the physical distribution of M3 nodes across the Grenoble site, illustrating the scale and layout of the deployment used for dataset generation and validation in this thesis.

Figure 1.2 shows one such node as physically installed on site, mounted under the false ceiling described above.

Because FIT IoT-LAB nodes are deployed in a real indoor environment, subject to walls, furniture, human movement, and radio interference from co-located experiments and ambient Wi-Fi traffic, experiments run there naturally exhibit the propagation effects that idealized simulator defaults do not capture: multipath fading, shadowing from obstacles, and time-varying interference. This makes FIT IoT-LAB the natural ground truth against which a calibrated Cooja radio

[1] github.com/contiki-ng/cooja/blob/master/java/org/contikios/cooja/radiomediums/
[2] https://slices-fr.eu/slices-ri/

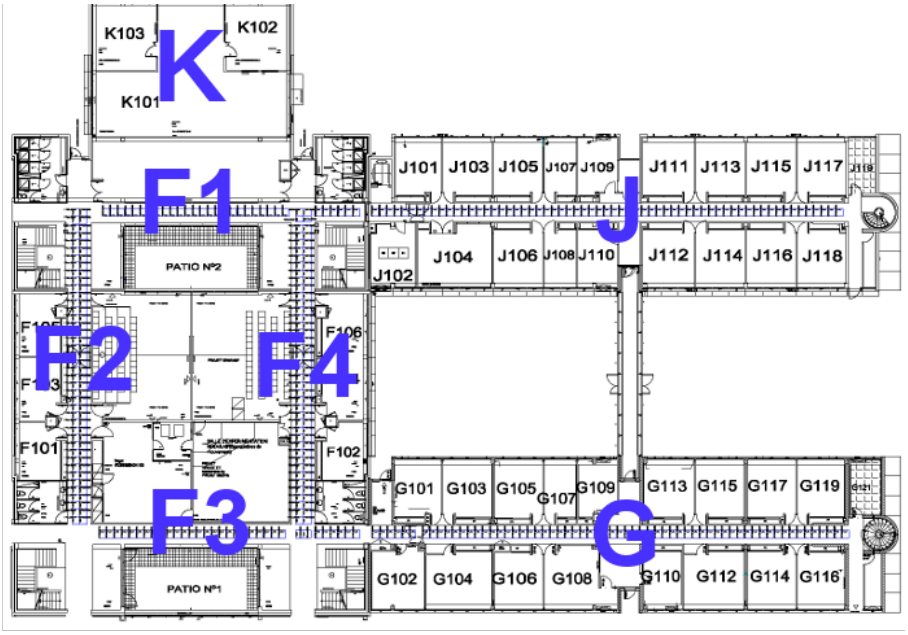


Figure 1.1: Distribution of M3 nodes across the FIT IoT-LAB Grenoble Deployment (2024).

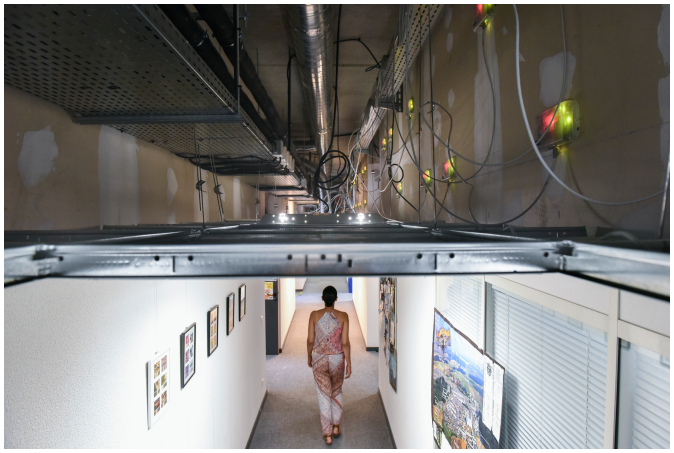


Figure 1.2: FIT IoT-LAB M3 node installed under a false ceiling at the Grenoble site.

model should be checked, and it is the testbed used for both dataset generation and validation throughout this thesis.

1.2.2 Network Simulation in the Development Workflow

Network simulation is typically used at two distinct stages of IoT development. Early in the design phase, it allows protocol and application logic to be tested quickly and cheaply, without requiring access to physical hardware or a testbed reservation, and without the variability that a real radio channel introduces between otherwise identical runs. Later, it can support deployment planning: for example, estimating how many nodes are needed to cover an area, or predicting the impact of a routing protocol change, before committing to a real rollout. Both use cases depend on the simulator's output being representative of real network behavior. A simulator whose radio

model is uncalibrated may still be useful for verifying that application logic behaves correctly, since routing decisions, retransmission logic, and application-level processing all execute identically to the real firmware; but it cannot be trusted to answer quantitative questions about packet delivery, delay, or energy consumption for a specific real deployment. Closing this gap is precisely what a calibrated *digital twin*, a virtual representation of a real deployment that functions as its digital counterpart (Khan et al., 2022), is meant to achieve, extending Cooja's usefulness from purely functional testing to quantitatively predictive deployment planning; building and validating such a digital twin is the objective of this thesis.

1.3 Radio Propagation Modeling

1.3.1 From Free-Space Path Loss to Log-Normal Shadowing

Radio propagation models describe how signal power decreases with distance and environmental conditions between a transmitter and a receiver. The simplest model, free-space path loss, assumes an unobstructed line of sight and predicts signal attenuation as a deterministic function of distance and frequency alone, following the log-distance path loss model:

$$PL(d) = PL(d_0) + 10\,n\,\log_{10}\left(\frac{d}{d_0}\right) \tag{1.1}$$

where $PL(d)$ is the path loss in dB at distance d, $PL(d_0)$ is a reference path loss measured at a reference distance d_0, and n is the Path Loss Exponent (PLE), which Singh (2021) shows increases with the density and number of obstacles in the propagation environment.

Real indoor and outdoor environments, however, introduce reflection, diffraction, and scattering from obstacles, which Equation (1.1) alone cannot represent, and which cause received signal strength to vary even between two points at the same distance from a transmitter. The log-normal shadowing model extends the log-distance model by adding a random shadowing term:

$$PL(d) = PL(d_0) + 10\,n\,\log_{10}\left(\frac{d}{d_0}\right) + X_\sigma \tag{1.2}$$

where X_σ is a zero-mean Gaussian random variable, in dB, with standard deviation σ; Singh (2021) describes this term as capturing the variability observed at a fixed distance due to obstacles and multipath effects. This combination makes the log-normal shadowing model a reasonable statistical approximation of real-world propagation without requiring a full deterministic description of the physical environment, such as wall placement or material properties, which would be impractical to obtain and encode in most simulation contexts.

1.3.2 Radio Medium Models in Cooja

Cooja implements several radio medium models of varying fidelity, ranging from the Unit Disk Graph Model (UDGM) and the Directed Graph Radio Medium (DGRM) to the Multi-path Ray-tracer Medium (MRM) and LogisticLoss. And the choice of model directly determines what aspects of real propagation the simulation can capture. UDGM is the simplest and most widely used by default: it defines a fixed transmission range within which reception is guaranteed and outside of which it fails deterministically, optionally with a distance-dependent reception ratio to add some randomness. While computationally cheap and easy to reason about, UDGM has no grounding in a physical propagation model and cannot represent the gradual, probabilistic degradation of reception quality observed in real channels.

In particular, UDGM assumes that reception boundaries are perfectly circular around each transmitter, an assumption that does not hold in practice. Dezfouli et al. (2015) show that real radio links exhibit *radio irregularity*: signal attenuation varies with direction due to obstacles, antenna orientation, and environmental asymmetries, so that the actual reception boundary is irregular rather than circular. Figure 1.3 illustrates this discrepancy between the idealized circular range assumed by UDGM and the irregular reception boundaries observed empirically.

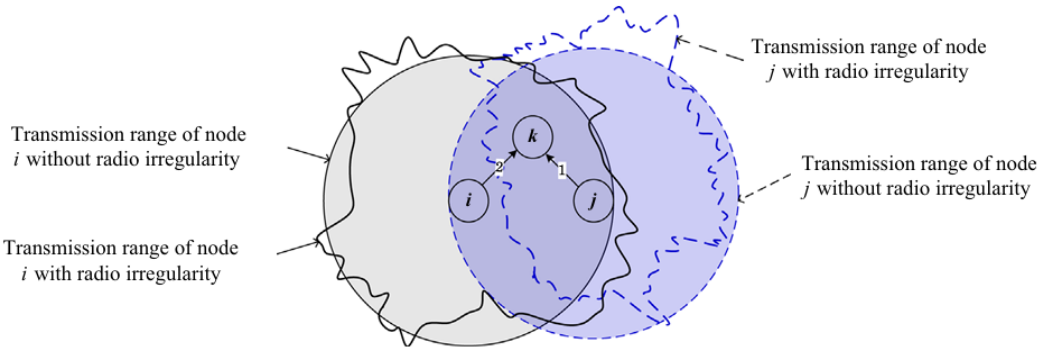


Figure 1.3: Radio irregularity changes the signal relationship between nodes, affecting the idealized circular range assumed by models such as UDGM. Reproduced from Dezfouli et al. (2015).

This limitation of UDGM is precisely what motivates the use of a statistically grounded model for calibration. Table 1.1 summarizes the main radio medium models available in Cooja and their relevance to this work.

LogisticLoss occupies a middle ground between the oversimplification of UDGM and the computational cost and environment-modeling burden of MRM: it is grounded in a recognized statistical propagation model, requires no explicit geometric description of the deployment environment, and remains computationally lightweight enough to run at the scale needed to generate a training dataset, which is why it was retained as the calibration target for this thesis.

Radio Medium	Description
UDGM	Deterministic or lightly randomized reception based purely on distance, with an idealized circular range; no grounding in a physical propagation model.
DGRM	Reception governed by an explicit, user-defined connectivity graph between node pairs; useful for reproducing a fixed topology but not generative.
MRM	Physically detailed ray-tracing model accounting for reflection and obstacles; high fidelity but computationally expensive and requires detailed environment geometry.
LogisticLoss	Statistical model combining log-normal shadowing with a logistic reception-probability curve; the model used and calibrated in this thesis.

Table 1.1: Overview of the main radio medium models available in Cooja (Contiki-NG, 2024).

1.3.3 The LogisticLoss Radio Medium

Cooja's LogisticLoss radio medium builds on the log-normal shadowing foundation described in Equation (1.2) but replaces the final step, converting received signal strength into a packet reception outcome, with a logistic (sigmoid) function of the Received Signal Strength Indicator (RSSI) rather than a hard threshold. Concretely, the Packet Reception Rate (PRR) predicted by the model increases smoothly from near zero to near one as RSSI increases past a characteristic value, rather than switching abruptly at a fixed sensitivity threshold. Dezfouli et al. (2015) explain that this better reflects how real receivers behave: reception at borderline signal strengths is probabilistic, not deterministic, because of noise and interference.

LogisticLoss exposes three parameters that jointly determine the shape of this reception curve and are the target of the calibration pipeline built in this thesis, summarized in Table 1.2.

Parameter	Role
Path Loss Exponent	Governs how fast the average received signal strength decreases with distance, inherited from the underlying log-normal shadowing model (Equation (1.2)).
RSSI Inflection Point	The RSSI value at which the logistic reception curve crosses 50% PRR, effectively setting the receiver's characteristic sensitivity.
AWGN sigma	Standard deviation of the Additive White Gaussian Noise (AWGN) term added to the signal, controlling how steep or gradual the transition between reliable and unreliable reception is around the inflection point.

Table 1.2: The three LogisticLoss parameters calibrated in this thesis.

Two additional parameters, transmitting range and RX sensitivity, exist in the model but are treated differently in this thesis; this distinction, and the rationale for restricting calibration to the three parameters above, is detailed in Chapter 2.

Figure 1.4 plots the resulting PRR-versus-RSSI curve at LogisticLoss's default RSSI Inflection Point, illustrating the smooth, probabilistic transition described above.

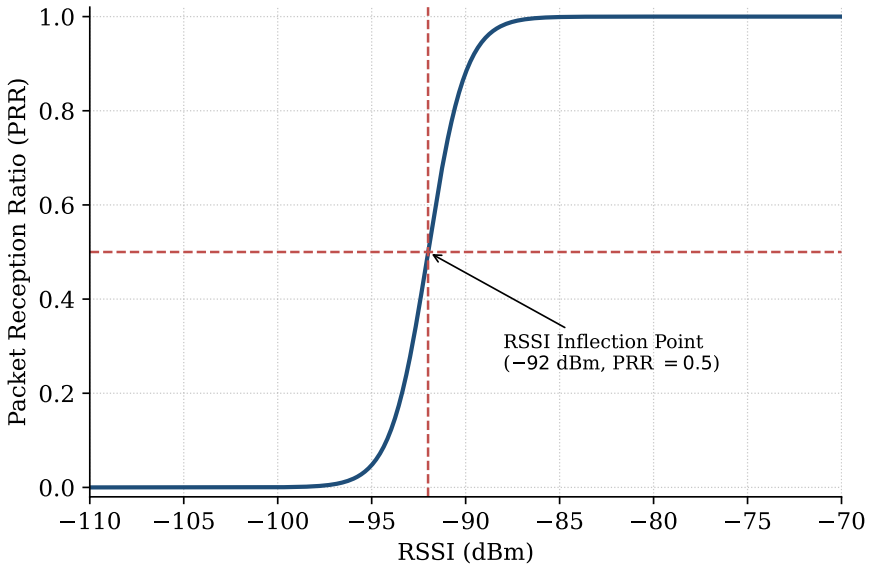


Figure 1.4: LogisticLoss's packet reception probability as a logistic function of RSSI, using the default RSSI Inflection Point (−92 dBm).

Left at their default values, these three parameters describe a generic propagation environment that does not correspond to any specific physical deployment. Since PRR is highly sensitive to all three, particularly the RSSI inflection point and AWGN sigma jointly, small mismatches between the assumed and actual environment translate into large discrepancies in simulated packet loss, delay, and throughput compared to a real deployment exhibiting the same topology and traffic pattern.

1.3.4 Key Performance Indicators

Throughout this thesis, network behavior is characterized through a set of Key Performance Indicators (KPIs), consistently measured on both the simulated and real sides so that they can be directly compared. The primary KPIs used are:

- **Packet Loss Rate:** the proportion of transmitted packets that are not successfully received, the most direct observable consequence of PRR.
- **End-to-end Delay:** the time elapsed between a packet's transmission and its reception, sensitive to retransmissions triggered by poor channel conditions.

- **Throughput:** the effective rate of successfully delivered data, jointly determined by loss rate, delay, and the underlying traffic pattern.
- **RSSI statistics:** the received signal strength associated with successfully received packets, used both as a diagnostic signal and, indirectly, as an input feature to the calibration model described in Chapter 2.

These KPIs, together with additional derived features described in Chapter 2, form the observable basis from which the calibration pipeline infers LogisticLoss parameters, and are also the quantities monitored by the adaptive extension to detect changes in network conditions over time.

1.4 The Simulation-to-Reality Gap and the Calibration Problem

A prior TER (Travail d'Étude et de Recherche) study conducted at the University of Strasbourg (Piveteau-Wernert, 2025) quantified the divergence between Cooja and FIT IoT-LAB by comparing simulated and real KPIs (throughput, delay, and packet loss rate) under Cooja's default UDGM configuration, identifying a simulation-to-reality gap of approximately 30%. That study establishes that a substantial, measurable gap exists between Cooja's default radio behavior and real FIT IoT-LAB conditions, and traces it to UDGM's radio model. As discussed in Section 1.3, UDGM assumes a perfectly circular, distance-only reception range with no statistical grounding in real propagation behavior, which cannot represent the gradual, probabilistic reception degradation and radio irregularity observed in practice.

This is precisely why LogisticLoss, rather than UDGM, was selected as the radio model to work with in this thesis: being grounded in the log-normal shadowing model (Section 1.3), it is documented in the literature to capture real channel variability, gradual reception probability, RSSI-dependent noise, considerably better than UDGM's idealized circular range (Dezfouli et al., 2015). However, this modeling advantage does not by itself close the simulation-to-reality gap: LogisticLoss's own default parameters, like UDGM's, are generic values not tuned to the specific conditions of any real deployment like FIT IoT-LAB sites, and using them out of the box provides no guarantee of matching real behavior. What motivates this thesis is therefore not a claim that LogisticLoss is already better calibrated than UDGM, but that it is more suitable radio model to *calibrate*, precisely because its parameters have a documented statistical relationship to real propagation behavior for a calibration procedure to exploit, one that UDGM's binary, distance-only model does not offer.

It is important to state precisely what the gap identified under UDGM reflects, since the same reasoning applies to LogisticLoss. The Cooja simulator runs the same Contiki-NG firmware that is deployed on the physical M3 nodes, and the same MAC and routing logic executes in both

environments; the divergence is not caused by a mismatch between simulated and real hardware behavior at the protocol level. Table 1.3 shows that: the CC2420 radio used by Cooja's Z1 mote and the AT86RF231 used by the physical M3 node are both IEEE 802.15.4-compliant 2.4 GHz transceivers, with sensitivity and output power figures in the same order of magnitude, so the hardware itself is not the primary source of divergence.

Table 1.3: Comparison of the CC2420 (Z1/Cooja) and AT86RF231 (M3/FIT IoT-LAB) radio chips

Parameter	**CC2420 (Z1)**	**AT86RF231 (M3)**
Standard	IEEE 802.15.4	IEEE 802.15.4
Band	2.4 GHz	2.4 GHz
Data rate	250 kbps	250 kbps (up to 2 Mbps, non-standard mode)
RX sensitivity	-95 dBm	-101 dBm
TX output power	0 dBm (typical)	Programmable, -17 to $+3$ dBm
Modulation	DSSS	O-QPSK
Interface	SPI	SPI

Instead, the divergence originates mainly in the radio medium model: default radio parameters describe generic propagation conditions rather than the specific conditions of the FIT IoT-LAB deployment site. This thesis therefore frames the problem explicitly as a *radio model fidelity* problem, and not as a hardware or firmware mismatch problem, a distinction that shapes the entire calibration approach adopted here.

Framed this way, closing the gap is fundamentally an inverse problem: given KPIs observed on a real network, infer the LogisticLoss parameters that would make the simulator reproduce that same behavior. Manually tuning three interacting parameters by trial and error to match observed KPIs is impractical, especially since the mapping between parameters and resulting KPIs is nonlinear and the parameters interact with one another; changing the RSSI inflection point, for instance, shifts the effective operating range in a way that also changes how sensitive the observed loss rate is to AWGN sigma. Existing calibration approaches range from empirical, manual parameter extraction to data-driven and Machine Learning (ML)-based inverse models, the latter motivated in particular by Adeogun (2019), who show that ML models can bypass manual parameter extraction entirely by learning the inverse mapping directly from data rather than requiring an analytical derivation of it. This motivates the data-driven Multilayer Perceptron (MLP)-based calibration pipeline developed in this thesis, described in full in Chapter 2.

Beyond a one-time calibration, real deployments are not static, interference sources appear and disappear, battery levels drain, and environmental conditions change over time, all of which can silently invalidate a calibration performed at a single point in time. A digital twin calibrated once and left unchanged will therefore drift away from the real network it is meant to represent as

conditions evolve, even if the initial calibration was accurate. This motivates extending the static calibration pipeline with an adaptive mechanism capable of detecting such changes and triggering recalibration automatically, keeping the digital twin synchronized with its real counterpart over time. This adaptive extension is likewise developed and validated later in this thesis.

1.5 State of the Art Synthesis and Positioning

Radio propagation modeling for wireless communications has been studied extensively at a general level, with log-normal shadowing established as a standard statistical approximation of real channel behavior Singh (2021). Applied to low-power IoT networks specifically, Dezfouli et al. (2015) characterize how simulators such as Cooja model the radio channel and document the limitations of simplified models like UDGM, including the anisotropic path loss behavior discussed in Section 1.3; Miranda et al. (2013) independently confirm this need for measurement-derived, rather than default, path loss and shadowing parameters through field measurements of IEEE 802.15.4 networks across free-space, in-building, and industrial environments. Ivanov et al. (2007) show that the same fidelity problem extends beyond Cooja, calibrating the widely used ns-2 simulator's radio model against a real wireless network before comparing simulated, emulated, and real behavior side by side. On the calibration side, Adeogun (2019) demonstrate that a neural network can learn the inverse mapping from observed channel statistics to propagation model parameters directly from data, without requiring an analytical or iterative parameter-extraction procedure, a result obtained in the context of channel impulse responses rather than network-level KPIs. Bharti et al. (2022) confirm this general strategy is not specific to neural networks, proposing a kernel-based, likelihood-free alternative that returns a full posterior over parameter values rather than a point estimate. Complementary work addresses related aspects of WSN performance modeling and radio link quality (Kanaris et al., 2019; Zinonos & Vassiliou, 2012), while more recent perspectives on IoT network and digital twin modeling are contributed by Tarafder et al. (2026) and, closer to this thesis's per-link, KPI-driven approach, by Si-Mohammed and Theoleyre (2025), who propose a lightweight, data-driven method for building self-adaptive digital twins from a single KPI, Packet Delivery Ratio, without calibrating an underlying physical radio propagation model.

Taken together, this body of work establishes two things relevant to this thesis: first, statistically grounded radio models such as log-normal shadowing and its LogisticLoss implementation are a sound basis for representing real propagation conditions in simulation, and that this holds beyond Cooja specifically, as Ivanov et al.'s ns-2 calibration and Miranda et al.'s field measurements confirm. Second, data-driven ML-based and kernel-based inverse modeling is a viable alternative to manual calibration, already validated at the channel level by Adeogun and Bharti et al., and, in a network-level by Si-Mohammed and Théoleyre's self-adaptive digital twin work. What remains absent from this literature is a pipeline that applies this data-driven inverse modeling approach directly to Cooja's LogisticLoss parameters using network-level KPIs measured

on a real IoT testbed, combined with an adaptive mechanism to keep such a calibration valid as conditions change over time. This is the gap this thesis addresses: Chapter 2 presents the proposed calibration pipeline in detail, and Chapter 3 presents its implementation and experimental validation.

1.6 Conclusion

This chapter established the technical context for the rest of this thesis: WSNs and IoT deployments rely increasingly on simulation for both development and deployment planning, Cooja and Contiki-NG provide a simulation environment that reuses real firmware and therefore isolates the radio channel as the main source of simulation-to-reality divergence, and FIT IoT-LAB provides the physical ground truth against which simulated behavior can be validated. The LogisticLoss radio medium was introduced together with its three key calibration parameters and the KPIs used to observe network behavior throughout this thesis. The simulation-to-reality gap was framed explicitly as a radio model fidelity problem rather than a hardware mismatch, grounded in prior measurements of this gap under UDGM, and positioning calibration as an inverse problem to be solved with a data-driven approach. The state of the art synthesis identified a clear gap: no existing work combines data-driven inverse modeling with Cooja's LogisticLoss parameters at the network-KPI level, validated against a real IoT testbed and extended with adaptive recalibration. The next chapter details the proposed method addressing this gap: how the three LogisticLoss parameters were selected for calibration, how the dataset was built, how the MLP-based calibration model was designed, and how the adaptive extension was built on top of it.

Chapter 2

Proposed Method

2.1 Introduction

Chapter 1 established that the gap between Cooja's default behavior and real FIT IoT-LAB deployments is a radio model fidelity problem, and framed closing it as an inverse problem: recovering the LogisticLoss parameters that make the simulator reproduce observed real-world network behavior. This chapter presents, in full, the method designed and built to solve that inverse problem, together with its extension to non-stationary deployments.

The chapter is organized as follows. Section 2.2 restates the calibration problem in precise, operational terms, introducing the notation used throughout the rest of the thesis. Section 2.3 then presents the complete methodology in seven parts: the overall pipeline architecture (Section 2.3.1); the analysis used to decide which of LogisticLoss's parameters are relevant for the calibration (Section 2.3.2); how the training dataset is generated, including the computational effort this required (Section 2.3.3); how the raw dataset was processed before training (Section 2.3.4); the design of the MLP used to learn the inverse mapping (Section 2.3.5); the design of the adaptive extension that keeps the calibration valid as real conditions change (Section 2.3.6); and the strategy used to evaluate both the static and adaptive components (Section 2.3.7). Section 2.4 closes the chapter with a summary that sets up Chapter 3, where implementation details and experimental results are presented.

Three design principles run through every part of this chapter and are worth stating up front, since they motivate several choices made later. First, *parsimony*: rather than calibrating every parameter LogisticLoss exposes, only those shown to meaningfully affect observable network behavior are targeted, keeping the learning problem well-posed. Second, *separation of concerns*: the pipeline is organized as a primary static calibration procedure, where the inverse mapping is learned once from simulation data, extended by a lighter adaptive recalibration mechanism that applies and re-triggers that learned mapping as needed, so that the computationally expensive part (dataset generation and training) never needs to happen on the critical path of monitoring a live deployment. Third, *grounding in real data at every stage*: both the static calibration and the adaptive extension are validated against real FIT IoT-LAB measurements rather than against simulation alone, since simulation-only validation cannot, by construction, say anything about the simulation-to-reality gap this thesis sets out to close. All code, scripts, and configuration

files described in this chapter are publicly available in the project's GitHub repository for reproducibility.[1]

2.2 Problem Statement

Let $\theta = (\text{PLE}, \text{RSSI}_{50\%}, \sigma)$ denote the vector of LogisticLoss parameters targeted for calibration (the rationale for restricting θ to exactly these three components, out of the five LogisticLoss exposes, is developed in Section 2.3.2). Let f denote the *forward model*: the process of running a Cooja simulation, for a fixed network topology and traffic pattern, with LogisticLoss configured at parameters θ, and extracting a vector of Key Performance Indicators (KPIs) $\mathbf{k}$ from the resulting serial log. Formally:

$$\mathbf{k}_{\text{sim}} = f(\theta) \tag{2.1}$$

f has no closed-form expression: it is the composition of Cooja's discrete-event simulation engine, the Contiki-NG networking stack (RPL, CSMA/CA), and the KPI extraction procedure applied to the resulting packet trace, none of which reduce to a tractable analytical function of θ. Given this, the calibration problem addressed in this thesis is to find the *inverse model* f^{-1}, such that, given KPIs $\mathbf{k}_{\text{real}}$ measured on a real deployment:

$$\hat{\theta} = f^{-1}(\mathbf{k}_{\text{real}}) \tag{2.2}$$

yields parameters $\hat{\theta}$ that, when fed back into the forward model, reproduce KPIs close to $\mathbf{k}_{\text{real}}$: $f(\hat{\theta}) \approx \mathbf{k}_{\text{real}}$. Figure 2.1 illustrates this forward/inverse relationship, and the role a learned approximation of f^{-1} plays in connecting the two.

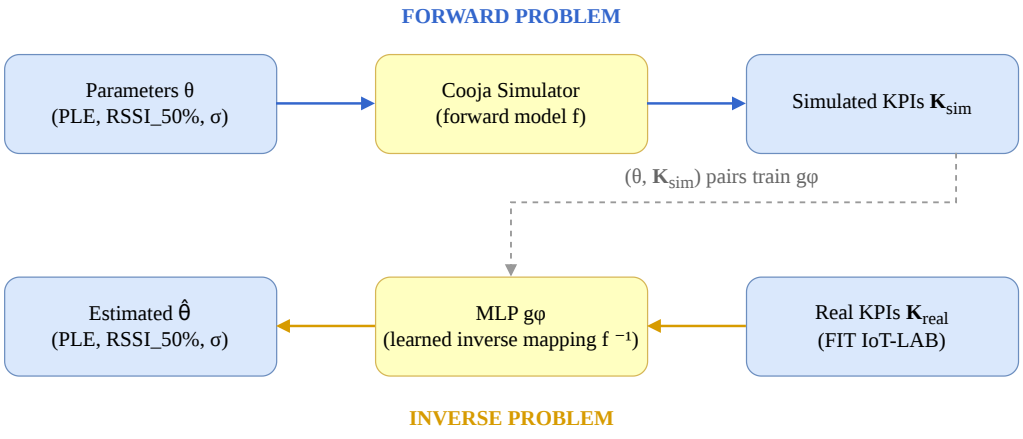


Figure 2.1: The forward problem (simulating KPIs from known parameters) versus the inverse problem addressed in this thesis (estimating parameters from observed KPIs).

[1] github.com/rahmamhb/calibration

Since f has no analytical form, neither does f^{-1}; it cannot be derived symbolically, only approximated. We propose in this work to approximate f^{-1} with a learned regressor g_ϕ, with parameters ϕ, trained on a dataset of $(\theta_i, \mathbf{k}_i)$ pairs obtained by evaluating the forward model f many times over a range of θ_i values (Section 2.3.3), so that $g_\phi(\mathbf{k}) \approx f^{-1}(\mathbf{k})$ for KPI vectors $\mathbf{k}$ within the range covered by training.

A second, complementary problem follows directly from the non-stationarity of real deployments discussed in Section 1.4: a $\hat{\theta}$ estimated at one point in time is not guaranteed to remain valid as real conditions evolve. Denoting by $\mathbf{k}_{\text{real}}(t)$ the real KPIs observed at time t, the second problem this thesis addresses is designing a procedure that (i) continuously monitors $\mathbf{k}_{\text{real}}(t)$, (ii) detects, with minimal delay and few false positives, the times t at which the underlying network conditions have changed enough to invalidate the current $\hat{\theta}$, and (iii) triggers a fresh evaluation of g_ϕ to produce an updated $\hat{\theta}_t$ at those times. This repeated re-evaluation is only practical because g_ϕ is a trained regressor rather than an analytical or iterative inverse solved anew each time, a single forward pass is enough to obtain $\hat{\theta}_t$, which is precisely the efficiency that has motivated machine learning approaches to inverse problems of this kind more broadly (Adeogun, 2019).

Sections 2.3.5 and 2.3.6 address these two problems, learning g_ϕ and building the monitoring-and-recalibration procedure around it, in turn

2.3 Methodology

2.3.1 Pipeline Architecture Overview

Figure 2.2 shows the overall architecture of the proposed solution: a primary static calibration pipeline, and a smaller adaptive recalibration extension built on top of it, introduced in Section 2.3.

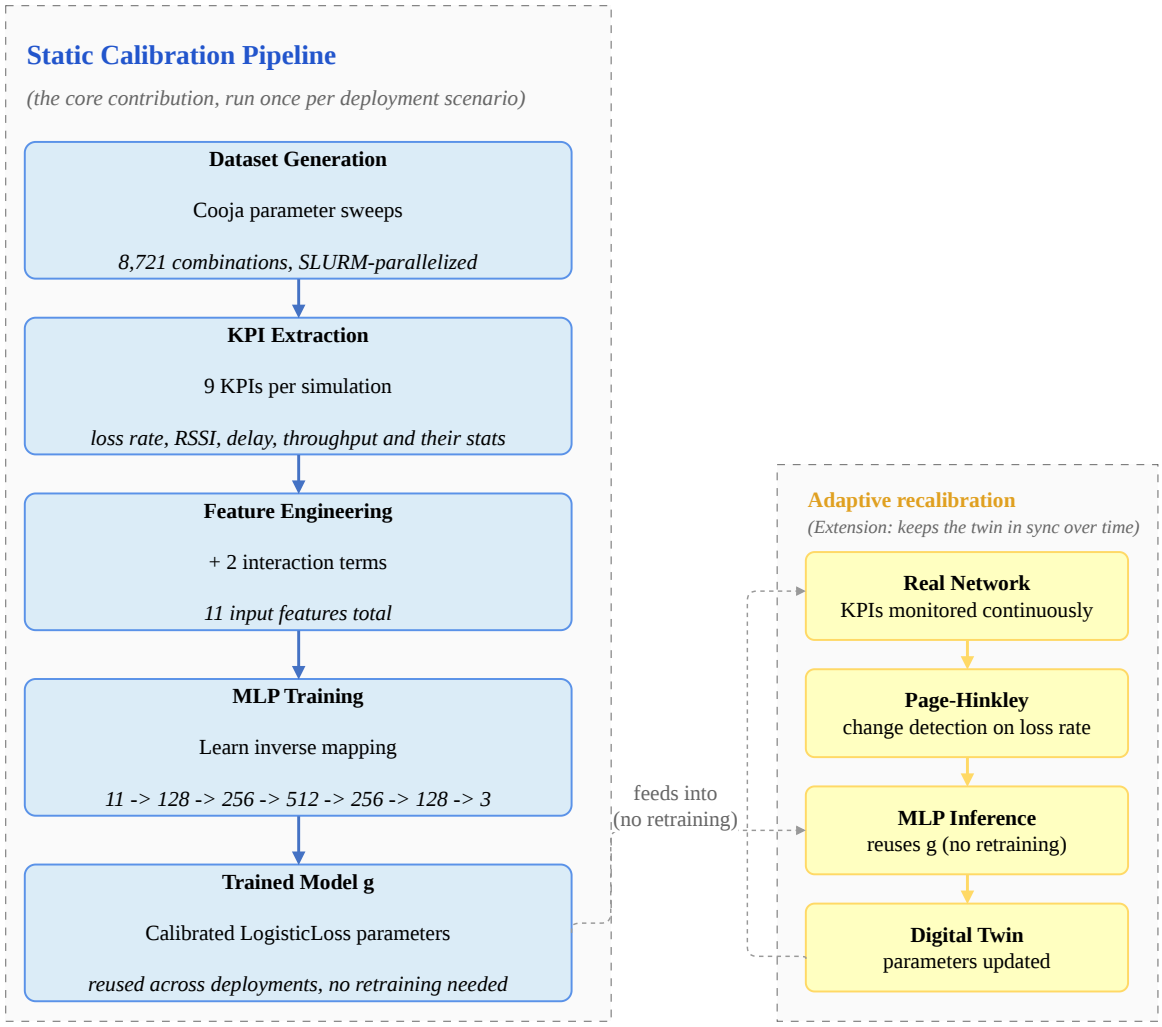


Figure 2.2: Two-part architecture of the proposed method: a dominant, sequential process on the left, and a lighter, continuously running loop on the right that depends on it.

The *static calibration pipeline* (left column of Figure 2.2) is the core contribution of this thesis, and is run once per deployment scenario, ahead of any live monitoring: it generates a labeled dataset by evaluating the forward model f across a wide, systematically chosen range of θ values (Section 2.3.3), extracts KPIs and engineers features from that raw dataset (Section 2.3.4), and trains g_ϕ on the result (Section 2.3.5). None of this needs to happen again for a new deployment of the same topology and traffic pattern, since g_ϕ is reusable; retraining is only needed if the topology or traffic pattern changes materially, since f itself, and therefore the data g_ϕ was trained on, is specific to those choices.

The *adaptive recalibration extension* (right column of Figure 2.2) builds on the trained model rather than duplicating the effort behind it: it runs continuously against a live deployment, monitoring real KPIs, flagging when they have shifted enough to warrant recalibration, and invoking the same g_ϕ on demand to produce an updated $\hat{\theta}$, which is pushed to update the digital twin's LogisticLoss configuration. This extension is computationally lightweight, a single forward pass through g_ϕ takes a fraction of a second, in contrast to the static calibration pipeline, which is

where essentially all of the computational cost of this thesis's method is incurred (Section 2.3.3). It is presented as an extension precisely because it adds no new modeling effort of its own, its value depends entirely on the calibration pipeline it sits on top of. Details about methods in each step are given in the following.

2.3.2 Parameter Selection Analysis

Candidate Parameters

LogisticLoss exposes five configurable parameters. Path Loss Exponent (PLE), RSSI Inflection Point, and AWGN sigma, introduced in Chapter 1 (Table 1.2), together shape the logistic reception curve directly. RX sensitivity, which is the minimum RSSI at which the receiving radio can detect a signal at all, sets a hard floor below which a packet is not received regardless of the other parameters. Transmitting range is a cutoff distance beyond which LogisticLoss does not attempt to model reception at all, used internally to bound the computation rather than to shape the reception curve itself.

Calibrating all five simultaneously would enlarge the regression problem g_ϕ has to solve into a needlessly high-dimensional output space, without necessarily improving the fidelity of the resulting digital twin: if some of these parameters have weak, unstable, or physically inappropriate relationships with the observable KPIs, including them only adds noise to the learning problem. Narrowing the candidate set before generating any data was therefore a deliberate methodological decision, not an incidental simplification. The following two subsections walk through exactly how this narrowing was carried out, starting from all five parameters and progressively eliminating those unsuited to calibration.

Literature-Guided Elimination

RX sensitivity was excluded from the candidate set on physical grounds rather than empirical ones because it characterizes the receiving radio's hardware floor, a property of the transceiver chip (e.g. the AT86RF231 on FIT IoT-LAB's M3 nodes) rather than of the propagation environment. Since it is a documented hardware characteristic rather than something that varies with deployment conditions, it is fixed to its datasheet-derived value (Atmel Corporation, 2014) rather than treated as a calibration target. This left four candidates: PLE, RSSI Inflection Point, AWGN sigma, and transmitting range.

Statistical Screening

The remaining four parameters were screened using two complementary analyses, applied to the raw generated dataset. The screening uses four *primary* KPIs, mean RSSI, loss rate, throughput, and mean delay, each with a direct physical interpretation and a one-to-one correspondence to a distinct dimension of network performance: signal quality, reliability, capacity, and latency,

respectively. The full set of KPIs used as model input features, and how these four relate to it, is detailed in Section 2.3.4.

- **Spearman correlation**, computed between each parameter and each of the four primary KPIs, to quantify the strength of any *monotonic* relationship between the two, whether or not that relationship is linear. Spearman correlation is the Pearson correlation coefficient computed on the ranks of the data rather than on the raw values:

$$\rho_{\theta_j,k} = \frac{\sum_i (R(\theta_{j,i}) - \overline{R(\theta_j)})(R(k_i) - \overline{R(k)})}{\sqrt{\sum_i (R(\theta_{j,i}) - \overline{R(\theta_j)})^2}\sqrt{\sum_i (R(k_i) - \overline{R(k)})^2}} \tag{2.3}$$

  where $R(\cdot)$ denotes rank. Spearman correlation was preferred over Pearson correlation here because several of the parameter-KPI relationships in this dataset are visibly step-like rather than strictly linear, and rank-based correlation captures a consistent monotonic trend even when its shape is nonlinear, which a linear correlation coefficient would understate.

- **Partial Dependence Plots (PDPs)**, which visualize the marginal effect of a single parameter θ_j on a KPI of interest, averaging over the other parameters, capturing relationships a single correlation coefficient collapses into one number: in particular, whether that effect is *consistent* across the parameter's range, or whether it reverses direction partway through, a distinction Spearman correlation alone cannot make.

Figure 2.3 shows the resulting Spearman correlation matrix between the four candidate parameters and the four primary KPIs, and Figure 2.4 shows the corresponding PDPs.

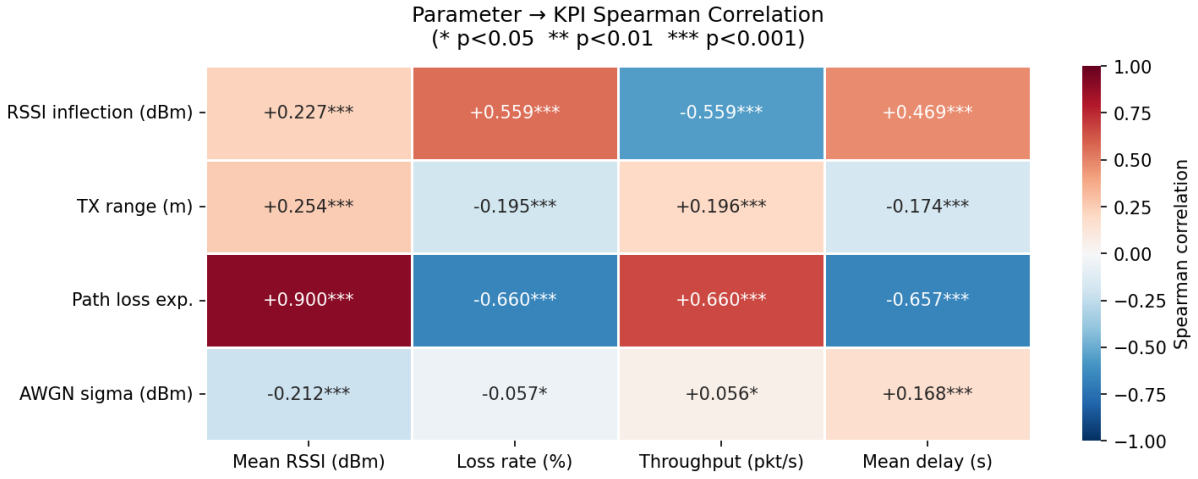


Figure 2.3: Spearman correlation between each of the four candidate LogisticLoss parameters and the four primary KPIs. All correlations shown are statistically significant ($p < 0.001$ unless otherwise marked).

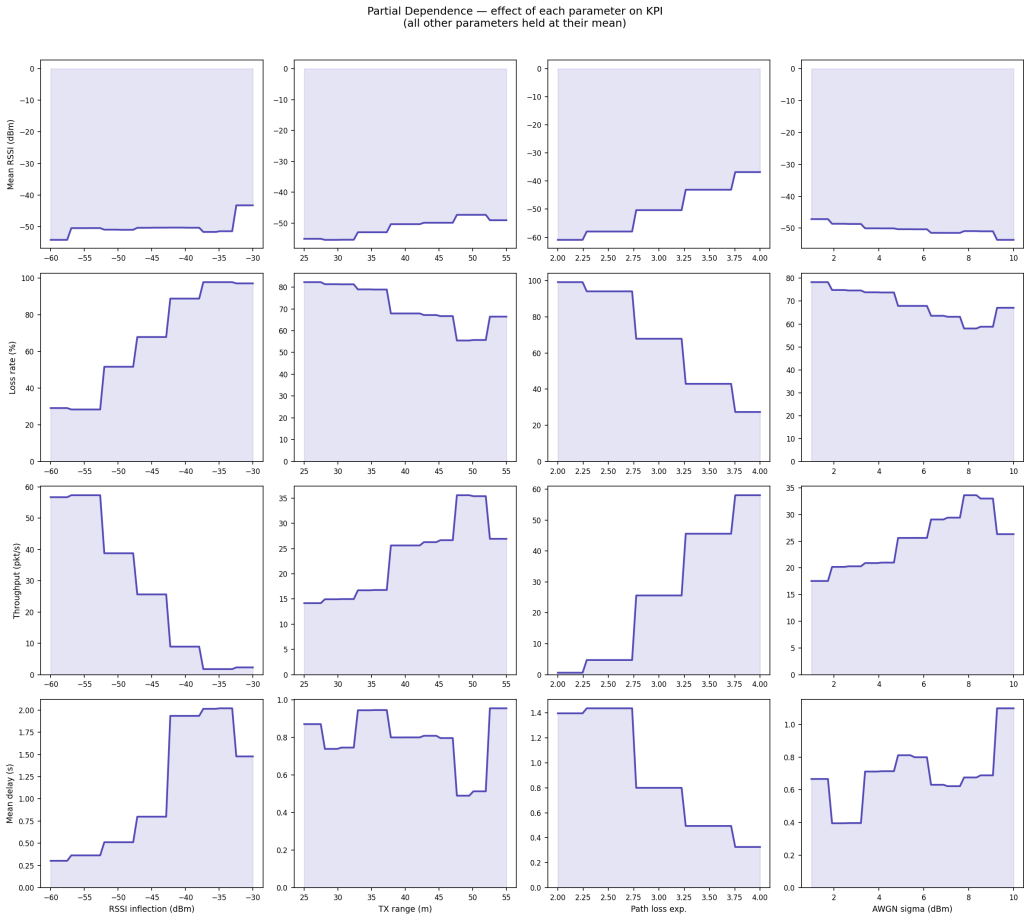


Figure 2.4: Partial dependence of each KPI on each candidate parameter, all other parameters held at their mean. Path Loss Exponent and RSSI Inflection Point show strong, consistently monotonic effects; TX range shows a visible reversal around 48-52 m; AWGN sigma shows a weaker but comparatively stable effect.

Two observations follow directly from these figures. First, PLE is the dominant parameter by a wide margin: it correlates with mean RSSI at $\rho = 0.900$, and with loss rate, throughput, and delay at $|\rho| \approx 0.66$ (Figure 2.3), and its PDP curves (Figure 2.4, third column) rise or fall consistently across its full range, with no reversal. RSSI Inflection Point is the second-strongest parameter, with correlations in the $|\rho| \approx 0.23$-0.56 range and similarly consistent PDP trends.

Second, and less immediately obvious from the correlation matrix alone, TX range and AWGN sigma are not clearly separated by correlation magnitude: TX range's correlations ($|\rho| \approx 0.17$-0.25) are, if anything, marginally *larger* than AWGN sigma's ($|\rho| \approx 0.06$-0.21). Correlation strength alone would therefore not justify retaining AWGN sigma over TX range. The PDPs resolve this: TX range's marginal effect on every KPI shows a visible reversal in the 48-52 m region, loss rate dips then rises again, throughput spikes then drops, mean delay dips then jumps back up, rather than changing consistently across the swept range. AWGN sigma's PDP curves, while shallower, do not show this kind of reversal; its effect is weaker but more consistent. This is precisely the failure mode PDPs were included to catch: a parameter whose correlation coefficient looks unremarkable, or in this case comparable to a parameter that is ultimately discarded,

can still be either usefully informative (if its effect is consistent) or unreliable (if its effect reverses direction), and only the shape of the PDP, not the correlation number, distinguishes the two cases.

Based on this analysis, **PLE, RSSI Inflection Point, and AWGN sigma are retained as the three parameters calibrated throughout this thesis**. The first two for their strong and consistent relationship with the observable KPIs, and AWGN sigma for showing a comparatively weaker but still stable, non-reversing effect. TX range is excluded despite comparable raw correlation magnitude to AWGN sigma, on the basis of the instability visible in its PDPs. This decision is further corroborated in Chapter 3 by a dedicated attempt to include TX range as a fourth regression target, which failed to produce reliable predictions, consistent with the instability observed here. $\theta = (\text{PLE}, \text{RSSI}_{50\%}, \sigma)$, as introduced in Section 2.2, is therefore the three-parameter vector calibrated by g_ϕ throughout the remainder of this thesis.

2.3.3 Dataset Generation Approach

Learning $g_\phi \approx f^{-1}$ requires a dataset of $(\theta_i, \mathbf{k}_i)$ pairs spanning a wide range of propagation conditions, obtained by evaluating the forward model f (Equation (2.1)) at many different θ_i. This section describes how that dataset was generated: the simulation scenario used for every run, the parameter sweep design, and the computational cost this incurred, which was substantial enough to shape several downstream decisions in this thesis.

Simulation Configuration

To ensure that the generated observations differ only as a result of the calibration parameters, the network configuration is kept fixed throughout the parameter sweep. The simulations use the same topology, traffic pattern, and Contiki-NG firmware, while only the three selected LogisticLoss parameters are varied.

For the dataset generated in this section, the simulation configuration consists of one designated receiver and 20 sender nodes. Each sender generates UDP traffic toward the receiver at a constant rate of 15 packets/s. Each simulation runs for 30 simulated minutes, over which each sender transmits roughly 27,000 packets (15 packets/s $\times$ 1,800 s), or about 540,000 packets network-wide across the 20 senders. This configuration is therefore not itself a variable of the dataset generation process; rather, it defines the fixed experimental setting under which the different LogisticLoss parameter combinations are evaluated.

At the end of each simulation, the resulting serial log is processed to extract the KPIs described in Section 2.3.4. Consequently, each parameter combination produces one dataset observation of the form

$$(\theta_i, \mathbf{k}_i), \tag{2.4}$$

where $\theta_i = (\text{PLE}_i, \text{RSSI}_{50\%,i}, \sigma_i)$ denotes the LogisticLoss parameters used in simulation i, and $\mathbf{k}_i$ denotes the corresponding vector of measured simulation KPIs.

Parameter Sweep

The three selected calibration parameters, RSSI Inflection Point, Path Loss Exponent (PLE), and AWGN sigma, are varied systematically using a full factorial design. Rather than selecting parameter combinations randomly, every combination of the predefined parameter values is evaluated. This provides systematic coverage of the parameter space and ensures that each combination contributes one observation to the dataset.

The two parameters excluded from calibration, RX sensitivity and transmitting range, remain fixed at the values determined in Section 2.3.2. The ranges and step sizes used for the three calibration targets are given in Table 2.1.

Parameter	**Role**	**Range and step**
RX sensitivity	Fixed (Section 2.3.2)	-100 dBm
Transmitting range	Fixed (Section 2.3.2)	40 m
RSSI Inflection Point	Calibration target	-120 to -30 dBm, step 5 (19 values)
Path Loss Exponent	Calibration target	1.0 to 5.0, step 0.25 (17 values)
AWGN sigma	Calibration target	1.0 to 14.0, step 0.5 (27 values)

Table 2.1: Parameter ranges and step sizes used for the full factorial dataset sweep.

With 19 possible values for RSSI Inflection Point, 17 values for PLE, and 27 values for AWGN sigma, the full factorial design contains

$$19 \times 17 \times 27 = \textbf{8,721 parameter combinations}. \tag{2.5}$$

Each combination is evaluated in one independent Cooja simulation. The resulting dataset therefore contains one generated observation for each of the 8,721 parameter combinations, before the subsequent data-processing steps described in Section 2.3.4.

Example of Generated Data

To illustrate how the parameter sweep is translated into dataset observations, Table 2.2 presents a small excerpt from the generated dataset.

In this excerpt, RSSI Inflection Point is fixed at -120 dBm and PLE takes two consecutive values, 1.0 and 1.25, while AWGN sigma varies across part of its swept range. Each row

corresponds to one simulation and contains the three parameter values used for that simulation together with the KPIs extracted from its serial log.

The purpose of this excerpt is only to illustrate the structure of the generated dataset: the parameter values form the target variables, while the resulting simulation KPIs form the corresponding observations used later for feature extraction and model training. The complete dataset contains all 8,721 parameter combinations.

id	RSSI_50%	PLE	AWGN	m_rssi	s_rssi	min_rssi	loss_pct	thrput	delay	s_delay	s_loss	min_prr
1	−120.0	1.00	1.0	−88.00	3.013	−96	77.21	27.35	1.100	0.688	3.205	0.172
2	−120.0	1.00	1.5	−87.79	3.418	−98	76.27	28.48	1.020	0.664	2.877	0.181
3	−120.0	1.00	2.0	−87.95	3.620	−100	77.84	26.58	1.197	0.854	6.226	0.117
4	−120.0	1.00	2.5	−88.50	3.911	−100	76.94	27.66	1.154	0.846	6.044	0.145
5	−120.0	1.00	3.0	−87.83	4.125	−100	76.98	27.62	1.084	0.672	2.600	0.176
6	−120.0	1.00	3.5	−88.23	4.841	−100	78.07	26.32	1.117	0.700	3.527	0.159
7	−120.0	1.00	4.0	−87.71	5.337	−100	76.48	28.23	1.042	0.671	3.573	0.174
8	−120.0	1.00	4.5	−87.62	5.715	−100	76.72	27.94	1.081	0.670	2.957	0.179
9	−120.0	1.00	5.0	−87.63	5.717	−100	77.86	26.58	1.119	0.725	4.110	0.158
10	−120.0	1.00	5.5	−87.80	6.224	−100	77.48	27.02	1.093	0.721	4.181	0.147
11	−120.0	1.00	6.0	−87.80	6.599	−100	76.88	27.75	1.094	0.745	4.902	0.132
12	−120.0	1.00	6.5	−87.66	6.959	−100	77.58	26.90	1.131	0.773	4.689	0.153
13	−120.0	1.00	7.0	−87.29	7.408	−100	77.05	27.54	1.112	0.673	2.770	0.185
14	−120.0	1.00	7.5	−87.49	7.598	−100	78.29	26.06	1.174	0.700	2.778	0.175
15	−120.0	1.00	8.0	−87.25	8.238	−100	76.73	27.92	1.074	0.670	2.934	0.187
16	−120.0	1.00	8.5	−87.04	8.409	−100	76.55	28.14	1.085	0.653	2.485	0.189
17	−120.0	1.00	9.0	−87.40	8.817	−100	77.67	26.80	1.162	0.727	3.707	0.165
18	−120.0	1.00	9.5	−87.18	9.079	−100	77.95	26.46	1.172	0.764	4.457	0.143
19	−120.0	1.00	10.0	−86.92	9.425	−100	77.31	27.23	1.144	0.672	1.938	0.195
20	−120.0	1.00	10.5	−86.71	9.659	−100	77.38	27.13	1.127	0.707	3.584	0.169
21	−120.0	1.00	11.0	−86.80	10.026	−100	77.73	26.72	1.155	0.675	2.420	0.173
22	−120.0	1.00	11.5	−86.55	10.371	−100	78.02	26.37	1.179	0.710	3.224	0.176
23	−120.0	1.00	12.0	−86.66	10.625	−100	77.87	26.56	1.145	0.687	2.291	0.183
24	−120.0	1.00	12.5	−85.69	11.134	−100	77.56	26.93	1.179	0.731	3.936	0.162
25	−120.0	1.00	13.0	−86.15	11.369	−100	78.29	26.07	1.234	0.709	2.047	0.175
26	−120.0	1.00	13.5	−85.97	11.708	−100	78.32	26.03	1.226	0.734	3.415	0.163
27	−120.0	1.00	14.0	−85.70	12.015	−100	76.29	28.46	1.096	0.662	2.870	0.182
28	−120.0	1.25	1.0	−85.09	3.834	−95	76.76	27.90	1.088	0.669	2.633	0.191
29	−120.0	1.25	1.5	−85.24	4.315	−96	76.45	28.26	1.045	0.670	3.801	0.176
30	−120.0	1.25	2.0	−85.24	5.022	−99	76.20	28.56	0.999	0.616	2.722	0.180
32	−120.0	1.25	3.0	−85.38	4.658	−100	76.82	27.81	1.040	0.694	4.063	0.184
33	−120.0	1.25	3.5	−84.64	5.680	−100	77.01	27.60	1.097	0.687	3.380	0.172

Table 2.2: DATA EXCERPT — Representative rows from the generated dataset, showing the LogisticLoss parameter values used for each simulation and the corresponding KPIs extracted from its output.

The size of this factorial sweep, 8,721 combinations, is a property of the specific ranges and step sizes chosen for the three calibration parameters in this thesis (Table 2.1), not a fixed requirement of the method itself. A coarser sweep, wider steps or narrower ranges, would produce fewer combinations at the cost of sparser coverage of the parameter space, while a more demanding scenario, a broader operating range to cover, additional propagation regimes, or finer resolution needed to resolve a parameter's effect near a critical region, would require a denser sweep and correspondingly more combinations. The number of combinations is therefore a tunable design choice governed by how much coverage a given deployment scenario demands, not a constant dictated by the method, and it is this choice, rather than the calibration procedure itself, that determines the computational cost discussed next.

Computational Cost

Every simulation is fixed at 30 simulated minutes, with Cooja configured to run at unlimited speed rather than real time. This distinction matters: a fixed simulated duration does not translate into a fixed wall-clock duration, since Cooja's actual processing time depends on how much there is to simulate, not on the simulated clock alone (Contiki-NG Cooja Simulator, 2024). In particular, wall-clock time scales with topology complexity: scenarios with more sender nodes and/or a higher per-node packet rate generate proportionally more discrete events for Cooja to process within the same 30 simulated minutes, and can take well over 30 minutes of wall-clock time to complete, while lighter scenarios, fewer senders, a lower packet rate, can finish in as little as 10 minutes. For the specific topology and traffic pattern used in this parameter sweep (a heavy-traffic scenario: 20 sender nodes, each transmitting at 15 packets/s, a congested, high-loss regime deliberately chosen because it exercises the widest range of loss conditions and is therefore the most informative for training g_ϕ), each individual simulation took approximately **36–37 minutes of wall-clock time** (measured directly from job logs, averaging around 2,190 seconds), covering the Cooja run itself, log conversion, and KPI extraction. This figure is specific to that topology and should not be read as a general property of 30-minute Cooja simulations. Run sequentially on a single machine, generating the full dataset would have required:

$$8{,}721 \times 36.5 \text{ min} \approx 318{,}300 \text{ min} \approx 5{,}305 \text{ hours} \approx \textbf{221 days} \tag{2.6}$$

which is not feasible on a project timeline measured in months. Since every simulation in the sweep is fully independent of every other, no shared state or synchronization is required between runs, the workload is what is commonly called *embarrassingly parallel*, and is thus well suited to distributing across many machines or cores at once. Dataset generation was therefore run on the host laboratory's SLURM-managed compute server, whose specifications are summarized in Table 2.3.

Concretely the full set of 8,721 parameter combinations was first enumerated into `combinations.csv`, then `submit_dataset.sh` submitted one SLURM job per row of that file as a job array, with

up to 24 jobs executing concurrently, one per CPU core; each job invoked `run_one_sim.py` to run a single Cooja simulation for its assigned parameter combination end to end, from launching Cooja through KPI extraction, independently of every other job. This reduced the total generation time to approximately:

$$\frac{221 \text{ days}}{24} \approx \textbf{9 days} \tag{2.7}$$

Component	Specification
CPU	24 cores
RAM	128 GB
Storage	5.5 TB
GPU	NVIDIA, 32 GB VRAM
OS	Ubuntu Server 22.04 LTS
IP Address	193.50.39.51
Hostname	cran-server

Table 2.3: Specifications of the SLURM-managed compute server used for dataset generation.

Figure 2.5 illustrates this parallelization.

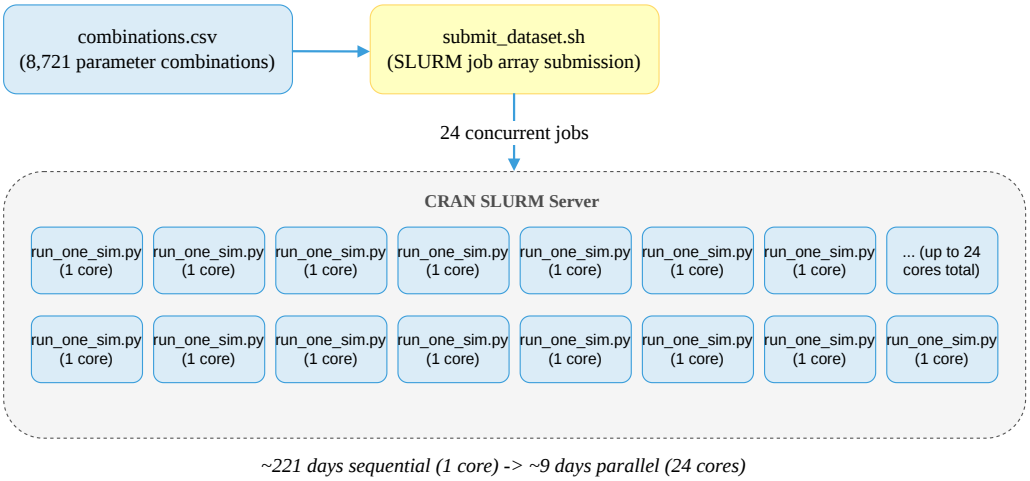


Figure 2.5: Distributing the 8,721 independent simulations across the server's 24 SLURM-managed cores brought total generation time down from roughly 221 days to roughly 9 days.

This cost is specific to the synthetic Cooja sweep; it is not incurred again for a new deployment unless the topology or traffic pattern changes, since g_ϕ, once trained, is reused directly (Section 2.3.1).

Real FIT IoT-LAB Scenarios

In addition to the synthetic sweep, a small number of real experiments were run on FIT IoT-LAB, described in full in Chapter 3, both to provide ground truth for validating the static calibration and, for the adaptive extension specifically. Unlike the synthetic sweep, these are single, fixed-duration experiments run directly on the physical testbed (on the order of tens of minutes of real time each) rather than a combinatorial parameter search, and their cost is governed by FIT IoT-LAB's testbed reservation and scheduling process rather than by wall-clock simulation time.

2.3.4 Data Processing

The raw dataset produced by the sweep in Section 2.3.3 cannot be used for training as-is; two processing steps were applied first.

Feature Extraction

Nine base KPIs are computed from each simulation's serial log: mean RSSI, standard deviation of RSSI, minimum RSSI, packet loss rate, throughput, mean delay, standard deviation of delay, standard deviation of loss rate across nodes, and the minimum per-node Packet Reception Ratio (PRR). These are computed from a single pass over the serial log that matches each packet's send and receive events; a packet whose receive event cannot be matched to a preceding send event, or whose receive line appears before its corresponding send line due to logging artifacts (Santi et al., 2021), is correctly treated as lost rather than silently dropped from the statistics, which would otherwise bias the loss rate downward.

The three RSSI-based KPIs (mean, standard deviation, minimum) are undefined whenever a simulation's loss rate reaches 100%: with no packets received at all, there is no RSSI to average. Rather than dropping these rows, which would discard otherwise valid loss, throughput, and delay measurements, the undefined RSSI values are filled with sentinel values: -100 dBm for mean and minimum RSSI (a floor below the receiver's sensitivity, Section 2.3.2), and 0 for RSSI standard deviation (no signal means no spread to measure). This keeps a complete-loss configuration in the dataset as a valid, fully-featured training example rather than an incomplete one that would need special-casing downstream.

Beyond these nine base KPIs, two interaction features are engineered and appended to the feature vector: `rssi_x_loss`, the product of mean RSSI and loss rate, and `throughput_per_loss`, throughput normalized by loss rate. These are included because the relationship between the base KPIs and the target parameters θ is not purely additive: a given loss rate corresponds to different regions of parameter space depending on the concurrent RSSI level, for instance. Encoding this interaction explicitly makes it directly available to g_ϕ as an input feature, rather than requiring the model to reconstruct it implicitly from the base features during training. In total, this yields **11 input features** per simulation, forming the feature vector $\mathbf{x}$ that g_ϕ actually consumes (as

opposed to the raw KPI vector **k** used in the problem formalization of Section 2.2, of which **x** is a deterministic transformation). No rebalancing or resampling is applied to this feature set: the dataset is used in its naturally generated distribution across loss-rate regimes, a choice discussed alongside the resulting distribution and its consequences for the adaptive extension in Chapter 3.

2.3.5 Model Development

Choice of Regressor

The inverse mapping $g_\phi \approx f^{-1}$ (Equation (2.2)) is a multi-output regression problem: given the 11-dimensional feature vector **x**, predict the 3-dimensional parameter vector θ. An MLP was chosen for this task over simpler regressor families for its ability to model the nonlinear, and in some cases non-monotonic, relationships between KPIs and LogisticLoss parameters surfaced by the parameter selection analysis (Section 2.3.2), and for its ability to share representational capacity across all three output parameters jointly within a single network, rather than treating each as an independent prediction target.

Architecture

Figure 2.6 shows the resulting architecture: the 11 input features feed into five hidden layers of widths 128, 256, 512, 256, and 128, before producing the three output parameters. Each hidden layer is followed by Batch Normalization, a ReLU activation, and Dropout.

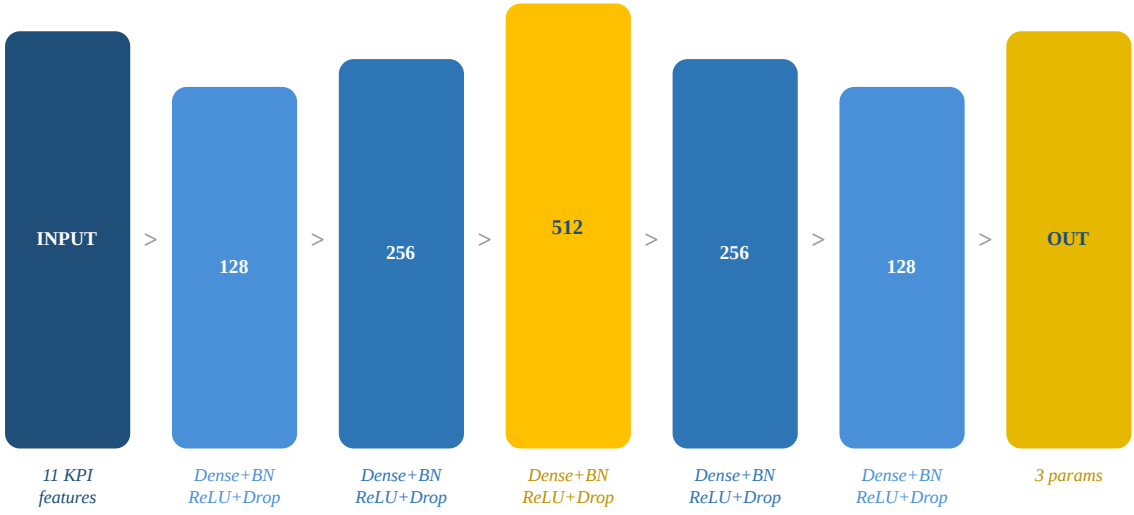


Figure 2.6: MLP architecture used to approximate the inverse mapping g_ϕ: 11 input features, five hidden layers (128-256-512-256-128) each with Batch Normalization, ReLU, and Dropout, and 3 output parameters.

Three architectural choices are worth motivating individually:

- **The funnel-expand-funnel width pattern** ($128 \to 256 \to 512 \to 256 \to 128$): the widest layer sits at the center of the network, giving it the most representational capacity at the

point where features from all three target parameters are jointly combined, while the narrower layers on either side act as a compression stage from the raw 11-dimensional input and an expansion-then-compression stage back down to the 3-dimensional output, encouraging the network to learn a shared intermediate representation before specializing toward the three outputs.

- **Batch Normalization**, applied after every hidden layer's linear transformation, normalizes layer activations across each training batch, which stabilizes training and allows a higher learning rate to be used without divergence, particularly relevant here given that the 11 input features span very different natural scales (RSSI in dBm, loss rate as a percentage, delay in seconds, and so on).

- **Dropout**, applied after each activation, randomly zeroes a fraction of activations during training, regularizing the network against overfitting to the specific 8,721 simulated points seen during training, which matters because the ultimate target of inference, real FIT IoT-LAB KPIs, was never part of the training distribution and the model must generalize to it.

Training Procedure

The network is trained to minimize the mean squared error between predicted and true parameter vectors, summed over the three output dimensions:

$$\mathcal{L}(\phi) = \frac{1}{N} \sum_{i=1}^{N} \| g_\phi(\mathbf{x}_i) - \theta_i \|_2^2 \tag{2.8}$$

over the training set described in Section 2.3.4, used in its naturally generated distribution. The exact training hyperparameters, optimizer, learning rate and schedule, batch size, dropout rate, and the train/validation/test split used, are listed in Table 3.2 in Chapter 3, alongside the training curves and final performance metrics they produced; this chapter has described the architecture and training objective as part of the proposed method, while their specific numerical settings and the results they yield are experimental outcomes reported with the rest of Chapter 3.

Algorithm 1 summarizes the static calibration procedure end to end, tying together dataset generation, processing, training, and inference into a single procedure.

Algorithm 1: Static LogisticLoss Calibration via Learned Inverse Mapping

Input: Parameter ranges for PLE, $\text{RSSI}_{50\%}$, σ (Table 2.1); fixed network topology and traffic pattern

Output: Calibrated parameters $\hat{\theta}$ for a given real deployment

$\mathcal{D} \leftarrow \emptyset$;

foreach *parameter combination* $\theta_i = (PLE_i, RSSI_{50\%,i}, \sigma_i)$ **do**

  $\mathbf{k}_i \leftarrow \text{RunCoojaSimulation}(\theta_i)$; `// forward model` f, `Eq. (2.1)`

  $\mathbf{x}_i \leftarrow \text{ExtractKPIsAndInteractionFeatures}(\mathbf{k}_i)$;

  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{x}_i, \theta_i)\}$;

$g_\phi \leftarrow \text{TrainMLP}(\mathcal{D})$; `// minimize` $\mathcal{L}(\phi)$ `in standardized space, Eq. (2.8), Section 3.3.3`

$\mathbf{k}_{\text{real}} \leftarrow \text{CollectRealKPIs}(\text{FIT IoT-LAB})$;

$\mathbf{x}_{\text{real}} \leftarrow \text{ExtractKPIsAndInteractionFeatures}(\mathbf{k}_{\text{real}})$;

$\hat{\theta} \leftarrow \text{InverseScale}\big(g_\phi(\text{Scale}(\mathbf{x}_{\text{real}}))\big)$; `// inference, Eq. (2.2), Section 3.3.3`

return $\hat{\theta}$

2.3.6 Adaptive System Design

Motivation and Choice of Detector

The static calibration procedure of Algorithm 1 produces a single $\hat{\theta}$, valid for the conditions under which $\mathbf{k}_{\text{real}}$ was collected. As discussed in Section 1.4, real deployments are not static, so keeping the digital twin synchronized over time requires continuously monitoring for the point at which the previously estimated $\hat{\theta}$ has become stale. This is a *concept drift detection* problem: detecting, from a stream of observed values, the point at which the underlying data-generating process has changed. Among standard drift-detection methods, this thesis adopts the Page-Hinkley test (Page, 1954), over alternatives such as the Drift Detection Method (DDM) (Gama et al., 2004), the Early Drift Detection Method (EDDM) (Baena-García et al., 2006), or ADWIN (Bifet & Gavaldà, 2007), for two practical reasons: it requires only a single running statistic per monitored KPI rather than a retained window of raw samples, keeping the adaptive extension lightweight, and it directly targets sustained shifts in the mean of the monitored quantity, which matches the kind of change a network experiencing new interference, battery drain, or environmental change actually exhibits, a step-like or gradually drifting shift in loss rate, rather than isolated outliers.

Page-Hinkley Mechanism

Given a stream of observations y_t (here, the loss rate at time t), the detector maintains *two* independent Page-Hinkley statistics in parallel, one sensitive to sustained increases in y_t and one to sustained decreases, so that both network degradation (rising loss) and network recovery

into a new stable regime (falling loss) can trigger recalibration:

$$m_t^{\uparrow} = m_{t-1}^{\uparrow} + (y_t - \bar{y}_t - \delta) \qquad m_t^{\downarrow} = m_{t-1}^{\downarrow} + (\bar{y}_t - y_t - \delta) \tag{2.9}$$

where $\bar{y}_t$ is the *true cumulative mean* of y since the last detector reset, and δ is a tolerance margin below which small fluctuations are not accumulated. $M_t^{\uparrow} = \min_{i \le t} m_i^{\uparrow}$ and $M_t^{\downarrow} = \min_{i \le t} m_i^{\downarrow}$ track the running minimum of each statistic, and the two test statistics are:

$$\mathrm{PH}_t^{\uparrow} = m_t^{\uparrow} - M_t^{\uparrow}, \qquad \mathrm{PH}_t^{\downarrow} = m_t^{\downarrow} - M_t^{\downarrow} \tag{2.10}$$

A change is flagged as soon as either $\mathrm{PH}_t^{\uparrow}$ or $\mathrm{PH}_t^{\downarrow}$ exceeds the threshold λ. Figure 2.7 illustrates one instance of this mechanism, the upward-detecting side, on an idealized loss-rate trace with a single step increase: $\mathrm{PH}_t^{\uparrow}$ stays near zero while the monitored quantity fluctuates around a stable mean, and grows sharply once a sustained shift begins, crossing λ shortly after the shift occurs; the downward-detecting statistic behaves as a mirror image for a step decrease.

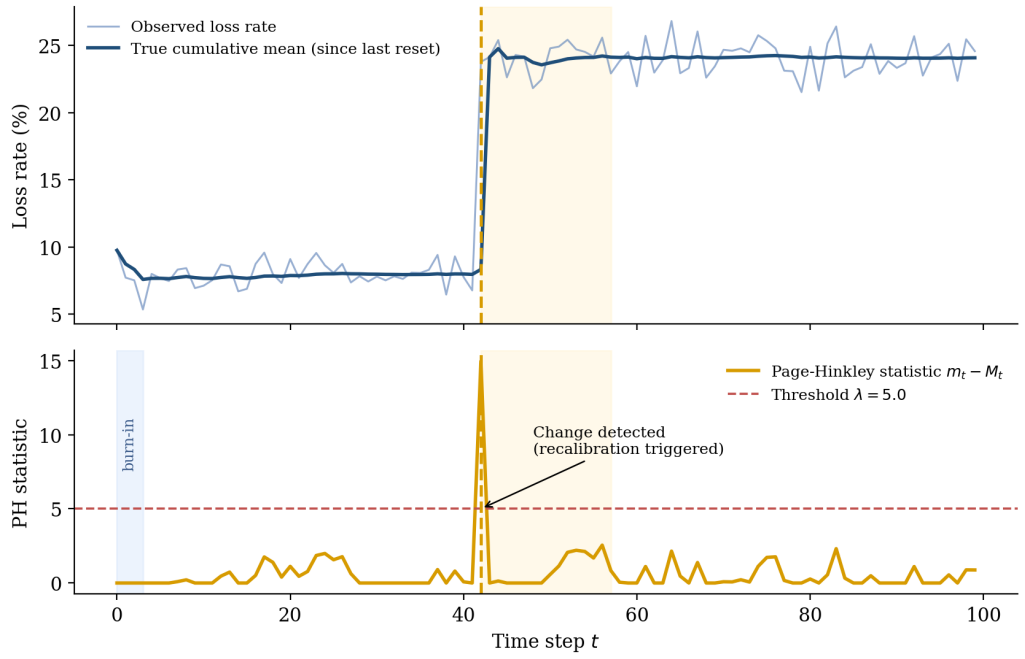


Figure 2.7: Illustrative Page-Hinkley detection mechanism for a single step increase (the upward-detecting statistic $\mathrm{PH}_t^{\uparrow}$): (top) an observed KPI with a step change and its true cumulative mean, reset after detection; (bottom) the resulting statistic, crossing the threshold λ shortly after the true change point. The downward-detecting statistic $\mathrm{PH}_t^{\downarrow}$ operates identically for step decreases.

Two further design decisions, beyond the two-sided formulation itself, are worth highlighting. First, $\bar{y}_t$ in Equation (2.9) is computed as the *true* cumulative mean since the last reset, rather than an exponential moving average (EMA), a common alternative in streaming statistics. An earlier design using an EMA was found to produce false alarms after genuine phase transitions: because an EMA lags behind a step change, tracking toward the new mean only gradually,

it keeps signaling drift for a period after the network has already stabilized in its new regime, since the gap between the still-adjusting EMA and the now-stable observations is mistaken for continued drift. The true cumulative mean, by contrast, is recomputed from all samples since the last reset and therefore reflects the current regime immediately once enough samples have accumulated, eliminating this failure mode.

Second, the accumulation in Equation (2.9) only begins after the burn-in period (Section 2.3.6) has elapsed: during burn-in, $\bar{y}_t$ is still computed and updated on every sample, but $m_t^{\uparrow}$ and $m_t^{\downarrow}$ are held at zero rather than accumulating deviations against a mean that has not yet stabilized. Accumulating deviations against an unstable early mean would risk seeding the detector with spurious drift before it has a trustworthy baseline to compare against.

Detector Parameters

The detector is configured with a tolerance margin $\delta = 0.5$, a detection threshold $\lambda = 5.0$, and a burn-in period of $b = 3$ samples during which $\bar{y}_t$ is computed and updated but $m_t^{\uparrow}$ and $m_t^{\downarrow}$ are held at zero (Section 2.3.6), so the deviation statistics only start accumulating once the mean has had a chance to stabilize. After a detected change, a cooldown period of $c = 15$ samples follows, during which incoming observations are not merely excluded from the detection check but dropped from the running statistics entirely, $\bar{y}_t$, $m_t^{\uparrow}$, and $m_t^{\downarrow}$ stay frozen at their post-reset values for the full cooldown duration. This is a stronger guarantee than simply disabling detection: it also prevents a transient spike occurring inside the cooldown window from being silently absorbed into the new mean before the test resumes, which would otherwise mask a second, genuine change immediately following the first.

Three-Phase Adaptive Loop

Combining the detector with the trained model g_ϕ yields the adaptive recalibration loop shown in Figure 2.8, structured around three repeating phases.

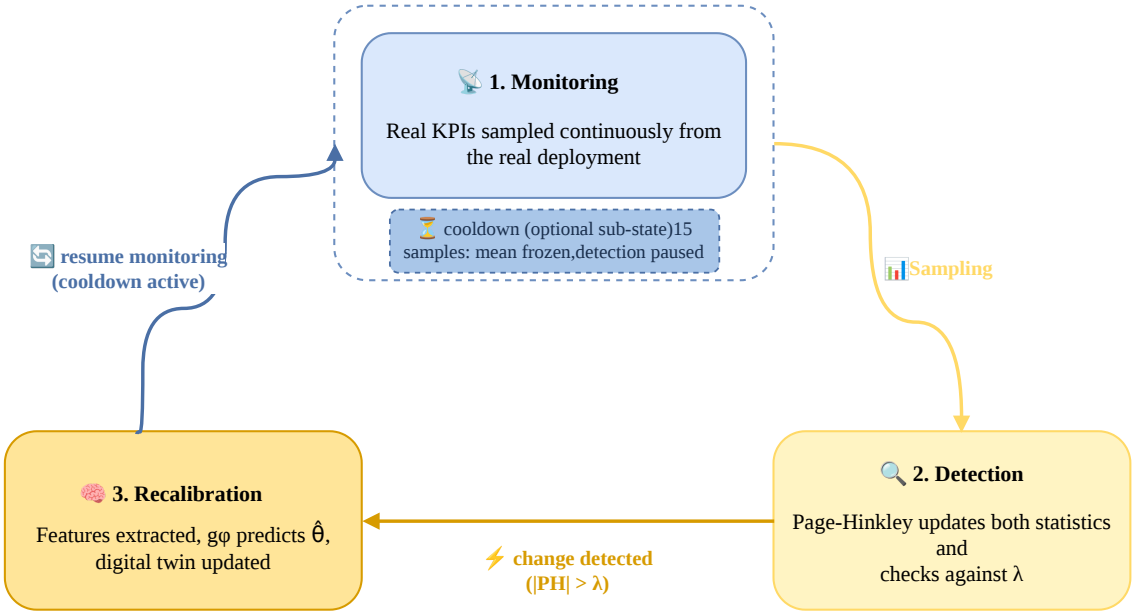


Figure 2.8: The three-phase adaptive recalibration loop: continuous monitoring, Page-Hinkley-based detection, and MLP-based recalibration, followed by a cooldown before monitoring resumes.

1. **Monitoring**: real KPIs are sampled continuously from the live deployment and fed to the Page-Hinkley detector one sample at a time.
2. **Detection**: the detector updates both running statistics (Equations (2.9)–(2.10)) and checks each against λ; if neither is exceeded, the loop returns to monitoring.
3. **Recalibration**: if a change is flagged, the current KPIs are passed through the same feature extraction used during dataset processing (Section 2.3.4) and then through g_ϕ to obtain an updated $\hat{\theta}_t$, which is pushed to update the digital twin; the detector is then reset and enters its cooldown period before monitoring resumes.

Algorithm 2 details this loop precisely, including the reset and cooldown logic.

Monitoring Interface

A lightweight Flask web interface was built on top of this loop to expose live monitoring: the current KPI trend, the detector's internal state ($m_t^{\uparrow}$, $m_t^{\downarrow}$, $M_t^{\uparrow}$, $M_t^{\downarrow}$, and both test statistics), and the calibration history (each past detected change and the $\hat{\theta}_t$ it produced) are visualized in real time. The interface is deployed on the same server and accessed remotely through an SSH tunnel, so that monitoring does not require direct network access to the server hosting the deployment. Its implementation is detailed in Chapter 3.

Algorithm 2: Adaptive Recalibration via Two-Sided Page-Hinkley Monitoring

Input: Trained MLP g_ϕ; tolerance $\delta = 0.5$; threshold $\lambda = 5.0$; burn-in $b = 3$; cooldown $c = 15$

$n \leftarrow 0$, sum $\leftarrow 0$, $m^{\uparrow} \leftarrow 0$, $m^{\downarrow} \leftarrow 0$, $M^{\uparrow} \leftarrow 0$, $M^{\downarrow} \leftarrow 0$, cooldown_remaining $\leftarrow 0$;
while *true* **do**
  $y_t \leftarrow$ ObserveRealKPI() ; `// loss rate at current time step`
  if *cooldown_remaining* > 0 **then**
    cooldown_remaining $\leftarrow$ cooldown_remaining $- 1$; `// observation dropped, mean frozen`
  else
    $n \leftarrow n + 1$, sum $\leftarrow$ sum $+ y_t$;
    $\bar{y}_t \leftarrow$ sum$/n$; `// true cumulative mean since last reset`
    if $n > b$ **then**
      $m^{\uparrow} \leftarrow m^{\uparrow} + (y_t - \bar{y}_t - \delta)$, $M^{\uparrow} \leftarrow \min(M^{\uparrow}, m^{\uparrow})$;
      $m^{\downarrow} \leftarrow m^{\downarrow} + (\bar{y}_t - y_t - \delta)$, $M^{\downarrow} \leftarrow \min(M^{\downarrow}, m^{\downarrow})$;
      if $m^{\uparrow} - M^{\uparrow} > \lambda$ ***or*** $m^{\downarrow} - M^{\downarrow} > \lambda$ **then**
        $\mathbf{k}_t \leftarrow$ CollectKPIWindow();
        $\mathbf{x}_t \leftarrow$ ExtractKPIsAndInteractionFeatures($\mathbf{k}_t$);
        $\hat{\theta}_t \leftarrow g_\phi(\mathbf{x}_t)$; `// recalibrate`
        UpdateDigitalTwin($\hat{\theta}_t$);
        $n \leftarrow 1$, sum $\leftarrow y_t$; `// reset, reseeded with the triggering value`
        $m^{\uparrow} \leftarrow 0$, $m^{\downarrow} \leftarrow 0$, $M^{\uparrow} \leftarrow 0$, $M^{\downarrow} \leftarrow 0$;
        cooldown_remaining $\leftarrow c$;

2.3.7 Evaluation Approach

The static and adaptive components of the pipeline require different evaluation strategies, since they are validated against different questions: whether g_ϕ recovers parameters that make the simulator behave like the real deployment, and whether the monitoring loop correctly detects and reacts to real condition changes.

Static Calibration

The static calibration model is evaluated along two complementary axes. First, in *parameter space*: comparing g_ϕ's predicted $\hat{\theta}$ against the true θ on held-out simulated data, using standard regression metrics reported in Chapter 3. Second, and more importantly given the goal of this thesis, in *behavior space*: feeding g_ϕ's predictions, obtained from real FIT IoT-LAB KPIs, back into Cooja and comparing the resulting simulated KPIs against the real measurements they were derived from, across multiple real deployment scenarios not used during training. This second axis is what actually measures whether the simulation-to-reality gap has closed, since a model could in principle achieve good parameter-space accuracy on synthetic data while still failing to

produce simulator behavior that matches reality, and it is this behavior-space comparison that is reported as the primary validation result in Chapter 3.

Adaptive Extension

The adaptive extension is validated using a two-stage strategy. It is first validated on synthetic data covering four distinct, deliberately engineered network regimes. These regimes are chosen so that the ground-truth timing and nature of each regime change is known exactly. This makes it possible to check the detection loop's behavior, whether it fires at the right time, how many samples of delay it incurs, whether it produces false positives during stable regimes, under fully controlled conditions, before it is trusted on real data. It is then validated on a real FIT IoT-LAB interference scenario, where the actual timing of condition changes is known from the experiment design (when interference sources were activated) but the KPIs themselves are subject to the same real-world variability as any other FIT IoT-LAB measurement, confirming that the detection-and-recalibration behavior verified on synthetic data continues to hold once the input is real, noisy network data rather than a clean synthetic trace. The full experimental setup and results for both validation stages are presented in Chapter 3.

2.4 Conclusion

This chapter presented the proposed calibration pipeline in full. Section 2.2 formalized the calibration problem as learning an inverse model $g_\phi \approx f^{-1}$, together with the complementary problem of detecting when a previously estimated $\hat{\theta}$ needs to be recomputed. Section 2.3 then developed the full method addressing both: the static calibration pipeline, the core contribution of this thesis, in which the three calibration-target parameters were selected through a literature-guided and statistical analysis (Section 2.3.2), a dataset of 8,721 labeled simulations was generated at a computational cost that required parallelizing across a 24-core server infrastructure to bring generation time down from roughly 221 days to about 9 days (Section 2.3.3), nine base KPIs and two engineered interaction features were extracted into an 11-dimensional feature vector used in its naturally generated distribution (Section 2.3.4), and an MLP was designed to learn the inverse mapping from this data (Section 2.3.5); and the adaptive recalibration extension built on top of it, in which a Page-Hinkley-based monitoring loop, using a true cumulative mean rather than an exponential moving average, was designed to detect condition changes and automatically trigger recalibration through the same trained model (Section 2.3.6). Section 2.3.7 laid out how both are evaluated, in parameter space and, more importantly, in behavior space against real FIT IoT-LAB measurements. The next chapter presents the implementation details, exact hyperparameters, and experimental results for every component introduced here.

Chapter 3

Implementation and Results

3.1 Introduction

Chapter 2 laid out the proposed method on paper: a static calibration pipeline learning the inverse mapping g_ϕ from network KPIs to LogisticLoss parameters, extended by an adaptive loop that keeps that calibration synchronized with a live deployment as conditions change. This chapter builds and runs the pipeline against real FIT IoT-LAB deployments, and reports what came out of it.

The chapter follows the pipeline's own logic. Before any results can be trusted, the environment and configuration behind them need to be pinned down: Section 3.2 covers the tools the implementation was built on, and Section 3.3 fills in the training configuration that Chapter 2 deliberately left open, together with a practical consequence of the method that only becomes visible once it is actually deployed, g_ϕ has to be retrained for every new topology, which turns out to matter a great deal for how the rest of this chapter is organized. Section 3.4 then fixes exactly what "working" means for the results that follow, so that the numbers reported later can be read consistently.

With that groundwork in place, Section 3.5 presents the results themselves, starting from how well g_ϕ learns the inverse mapping in the first place, then moving to whether that translates into simulated behavior that actually matches reality across several real deployments, and finally to whether the adaptive extension can keep that match up as the network changes underneath it, first in a controlled synthetic setting, then on a real, noisy interference experiment. Section 3.6 steps back to interpret what these results say together, and Section 3.7 closes the chapter.

3.2 Development Environment

Nearly everything behind this chapter, dataset generation, feature engineering, model training, and the adaptive extension's monitoring interface, ran on the host laboratory's server introduced in Table 2.3, GPU included wherever PyTorch [1] could make use of it. That includes the FIT IoT-LAB side of data collection too: node reservation, log retrieval, KPI extraction, training, the adaptive loop, all of it ran on the server rather than locally.

[1] https://pytorch.org/

That server-side pipeline is built in Python [2], with bash handling the orchestration scripts that tie the pieces together, job submission to SLURM (`submit_dataset.sh`, Section 2.3.3) being the clearest example. Within Python, PyTorch trains the MLP of Section 2.3.5, scikit-learn [3] handles the data splitting, feature and target scaling of Section 3.3.2, and the evaluation metrics reported later in this chapter, and Flask [4] serves the monitoring interface of Section 2.3.6. Figure 3.1 puts these together, and the full implementation is available in the project's GitHub repository [5].



Figure 3.1: The development stack used throughout this thesis's implementation.

3.3 Implementation Details

3.3.1 Per-Topology Training

As established in Section 2.3.1, g_ϕ is specific to the network topology and traffic pattern it was trained on, and must be retrained whenever those change materially. This is not a hypothetical caveat: validating the static calibration pipeline against multiple real FIT IoT-LAB deployments (Section 3.5.2) required generating a separate dataset and training a separate model for each validation topology, using the same architecture, dataset generation procedure (Section 2.3.3), and training configuration (Section 3.3.2) throughout, only the topology and traffic pattern behind the underlying Cooja sweep differ between them. Table 3.1 lists the three topologies used.

3.3.2 MLP Training Configuration

The MLP architecture matches the design presented in Section 2.3.5 exactly: 11 input features, hidden layers of width 128, 256, 512, 256, and 128, and 3 output parameters, with Batch Nor-

[2] https://www.python.org/
[3] https://scikit-learn.org/
[4] https://flask.palletsprojects.com/
[5] github.com/rahmamhb/calibration

Scenario	Senders	Rate	Regime
1	10	5 pkt/s	Light traffic, near-zero loss
2	20	15 pkt/s	Heavy traffic, high loss (≈85%); the topology used as the worked example in Section 2.3.3
3	5	15 pkt/s	Intermediate traffic

Table 3.1: The three real FIT IoT-LAB topologies used for static calibration validation, each with its own dataset and trained model.

malization and ReLU (Glorot et al., 2011; Ioffe & Szegedy, 2015) after every hidden linear layer. Dropout is applied after each hidden layer as well, at a rate of 0.1 for the first four hidden layers and 0.05 for the fifth (the layer closest to the output), a lighter rate late in the network leaves more of the final representation intact going into the 3-dimensional output. Table 3.2 lists the full training configuration.

Setting	Value
Loss function	Mean Squared Error (Equation (2.8))
Optimizer	Adam, weight decay 10^{-5}
Learning rate	10^{-3} initial
LR schedule	ReduceLROnPlateau (patience 40, factor 0.7, on validation loss)
Gradient clipping	max norm 1.0
Batch size	64
Train/test split	80% / 20%, `random_state=42`; the 20% held-out test split doubles as the validation set monitored during training (Section 3.3.2), no separate third split is used
Feature/target scaling	`StandardScaler`, fit on train only (Section 3.3.3)
Dropout	0.1 (layers 1–4), 0.05 (layer 5)
Max epochs	500
Early stopping patience	100 epochs (no validation loss improvement)
Checkpointing	best validation loss saved
Random seed	123

Table 3.2: MLP training configuration, identical across all three topologies of Table 3.1.

Training minimizes the mean squared error of Equation (2.8) using the Adaptive Moment Estimation (Adam) (Kingma & Ba, 2015) with a small weight decay for additional regularization alongside dropout. The learning rate is reduced by a factor of 0.7 whenever validation loss plateaus for 40 consecutive epochs, allowing the optimizer to take large steps early in training and refine its solution once progress stalls. Gradient norms are clipped to 1.0 before each optimizer step, guarding against the occasional large gradient that can otherwise destabilize training on a

dataset spanning several orders of magnitude across features (Section 2.3.4). Training runs for at most 500 epochs but stops early if validation loss fails to improve for 100 consecutive epochs, with the checkpoint corresponding to the best validation loss retained as the final model rather than whichever epoch training happened to stop at. All results in this chapter use a fixed random seed for reproducibility.

3.3.3 Feature and Target Scaling

The 11 input features and 3 output parameters are on very different natural scales, RSSI in tens of dBm, loss rate as a percentage, delay in seconds, PLE as a small unitless multiplier, and training directly on raw values would let the largest-magnitude quantities dominate the loss gradient regardless of their actual importance to the prediction. Both $\mathbf{x}$ and θ are therefore standardized before training: a `StandardScaler` is fit on the training split only, for the features and targets separately, and applied to transform both the training and test splits; the network is trained and evaluated entirely in this standardized space, and predictions are inverse-transformed back to physical units (dBm, unitless PLE, dBm sigma) only at the point they are reported or fed back into Cooja. Fitting the scalers on the training split alone, rather than on the full dataset before splitting, keeps the test split's statistics from leaking into training, consistent with the held-out evaluation described in Section 3.4. Each topology's scalers are saved alongside its trained model (Section 3.3.1) and reused unchanged at inference time, so a real KPI vector is standardized with exactly the statistics the corresponding g_ϕ was trained on.

3.4 Evaluation Metrics

Consistent with the two-axis evaluation strategy described in Section 2.3.7, the static calibration model is validated primarily in *behavior space*: for each validation topology, g_ϕ's prediction $\hat{\theta}$ is obtained from real FIT IoT-LAB KPIs, fed back into Cooja under that same topology, and the resulting simulated KPIs are compared directly against the real measurements they were derived from.

For each of the four primary KPIs (Section 2.3.2), the comparison is reported as the real value, the simulated value, and their difference $\Delta = \text{Real} - \text{Simulated}$, in each KPI's own natural unit, percentage points for loss rate (already a percentage), packets per second for throughput, seconds for delay, and decibels for RSSI, rather than as a normalized or percentage error. This is a deliberate choice rather than a simplification, for two reasons. First, a relative percentage error, $|\Delta|/\text{Real} \times 100\%$, is unstable or meaningless whenever the real value sits near zero, exactly the regime Scenario 1 falls in (Section 3.5.2): a real loss rate of 0.10% turns a practically negligible 0.22-percentage-point gap into a nominal "220% error," which would misrepresent an excellent result as a poor one. Second, RSSI is measured in dBm, a logarithmic, interval-scale quantity with no natural zero; a percentage difference on a dB value has no consistent physical meaning,

unlike a percentage difference in throughput or delay. Reporting Δ in each KPI's own unit sidesteps both problems and matches standard practice in the wireless networking literature for comparing measured and simulated link quality, and it allows both the direction and magnitude of any residual gap to be inspected KPI by KPI, rather than collapsed into a single aggregate score that could hide a large miss on one KPI behind good agreement on the others.

3.5 Results

The results are presented in the same order the pipeline itself builds them. Section 3.5.1 starts at the narrowest question, how accurately g_ϕ recovers the calibration parameters it was trained to predict, in parameter space alone. Section 3.5.2 then asks the question that actually matters for this thesis's goal: whether that parameter-space accuracy translates into simulated behavior that matches reality, across three real FIT IoT-LAB deployments. Sections 3.5.3 and 3.5.4 close the loop by asking whether the adaptive extension can keep that match holding as conditions change, first under controlled synthetic transitions with known ground truth, then on a real, noisy interference experiment.

3.5.1 ML Model Results

Table 3.3 reports MAE, RMSE, and R^2 per calibrated parameter for each of the three topologies of Table 3.1, at the best-validation-loss checkpoint (Section 3.3.2).

Scenario	**Target**	**MAE**	**RMSE**	R^2
1 (light)	RSSI Inflection Point	11.71	16.32	0.660
	Path Loss Exponent	0.077	0.121	0.990
	AWGN sigma	0.410	0.705	0.968
2 (heavy)	RSSI Inflection Point	5.57	8.41	0.906
	Path Loss Exponent	0.098	0.170	0.979
	AWGN sigma	0.490	0.836	0.959
3 (intermediate)	RSSI Inflection Point	9.53	13.50	0.757
	Path Loss Exponent	0.102	0.200	0.971
	AWGN sigma	0.431	0.898	0.953

Table 3.3: Held-out regression performance per parameter for each topology's trained g_ϕ.

A consistent pattern holds across all three topologies: Path Loss Exponent and AWGN sigma are both recovered with high accuracy ($R^2 \geq 0.95$ in every case), while RSSI Inflection Point is noticeably harder to pin down, with R^2 ranging from 0.660 (Scenario 1) to 0.906 (Scenario 2). This mirrors the parameter selection analysis of Section 2.3.2: PLE was identified as the dominant parameter by a wide margin (Spearman $\rho = 0.900$ with mean RSSI), and that dominance

carries through to how reliably g_ϕ can recover it from KPIs alone. RSSI Inflection Point, the second-strongest parameter in that analysis but well behind PLE, is correspondingly the harder of the three targets to regress precisely, though still learned well enough to be practically useful, as the behavior-space validation of Section 3.5.2 confirms.

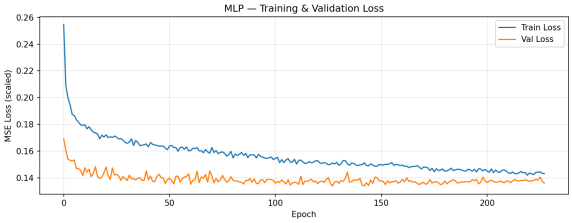
To check whether the MLP's added complexity is actually justified, g_ϕ's regression performance was additionally benchmarked against three standard baselines, Linear Regression, Random Forest, and XGBoost, trained and evaluated identically (same 11 features, same 80/20 split, same scaling) across all three topologies. Table 3.4 reports the resulting R^2 per parameter for every method.

Scenario	Target	Lin. Reg.	Rand. Forest	XGBoost	MLP
1 (light)	RSSI Inflection Point	0.552	0.635	0.631	**0.660**
	Path Loss Exponent	0.987	0.991	**0.994**	0.990
	AWGN sigma	0.923	0.965	0.967	**0.968**
2 (heavy)	RSSI Inflection Point	0.826	0.895	0.895	**0.906**
	Path Loss Exponent	0.958	0.981	**0.983**	0.979
	AWGN sigma	0.858	0.957	0.956	**0.959**
3 (intermediate)	RSSI Inflection Point	0.697	0.714	0.743	**0.757**
	Path Loss Exponent	0.965	0.972	**0.974**	0.971
	AWGN sigma	0.928	0.954	**0.957**	0.953

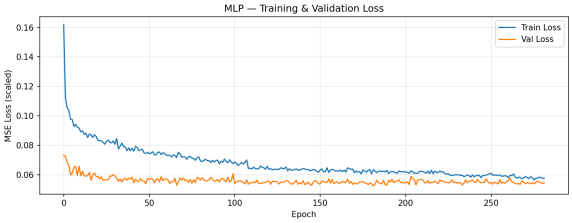
Table 3.4: R^2 comparison against four standard baselines, evaluated identically to g_ϕ across all three topologies. Bold marks the best-performing method per row.

No single method dominates uniformly. XGBoost achieves the highest R^2 for Path Loss Exponent in every scenario, by a small margin over the MLP, and the two are closely matched on AWGN sigma. The MLP's advantage is concentrated on RSSI Inflection Point, where it outperforms every baseline, including XGBoost, in all three scenarios, consistent with this parameter already being the hardest of the three to regress precisely (Section 3.5.1 above). Linear regression trails every nonlinear method throughout, most visibly on RSSI Inflection Point, indicating that parameter's mapping from KPIs is not well captured by a purely linear model.

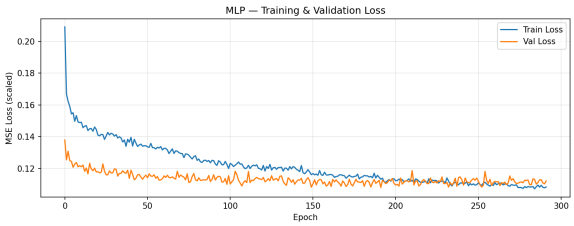
Training and validation loss curves for all three topologies are shown in Figure 3.2. All three show the expected pattern: validation loss drops sharply within the first 10–20 epochs, then continues to improve gradually as the learning rate schedule (Section 3.3.2) reduces the step size, with early stopping triggering once no further improvement is observed for 100 epochs. Scenario 2 reaches the lowest absolute validation loss (0.052), consistent with it having the largest, most balanced dataset of the three (its loss-rate distribution is shown in Figure 3.4); Scenario 1's validation loss (0.134) is noticeably higher, consistent with its weaker RSSI Inflection Point R^2 above.



(a) Scenario 1 (light)



(b) Scenario 2 (heavy)



(c) Scenario 3 (intermediate)

Figure 3.2: Training and validation MSE loss per epoch for all three topologies. Early stopping and best validation loss: Scenario 1, epoch 228 (0.134 at epoch 220); Scenario 2, epoch 282 (0.052 at epoch 276); Scenario 3, epoch 292 (0.108 at epoch 285).

The distribution each of these three datasets was trained on deliberately left unbalanced across loss-rate regimes (Section 2.3.4), is shown in Figures 3.3–3.5, one per topology. Each figure is a 3×3 grid of histograms, one per base KPI, with a dashed red vertical line marking that KPI's mean across the full parameter sweep. Rather than smooth, unimodal shapes, several KPIs show pronounced spikes at one or both extremes of their range: loss rate concentrates heavily near 0% and near 100%, with comparatively little density in between, and mean and minimum RSSI each show a sharp spike at exactly -100 dBm, the sentinel value assigned to total-loss rows where no packet was received to measure RSSI from (Section 2.3.4). Throughput mirrors the loss-rate pattern, clustering near its maximum for low-loss configurations and near zero for high-loss ones. This shape is a direct visual signature of the dataset's natural, unbalanced distribution rather than a binning artifact, and is precisely what the "no rebalancing" decision of Section 2.3.4 leaves intact.

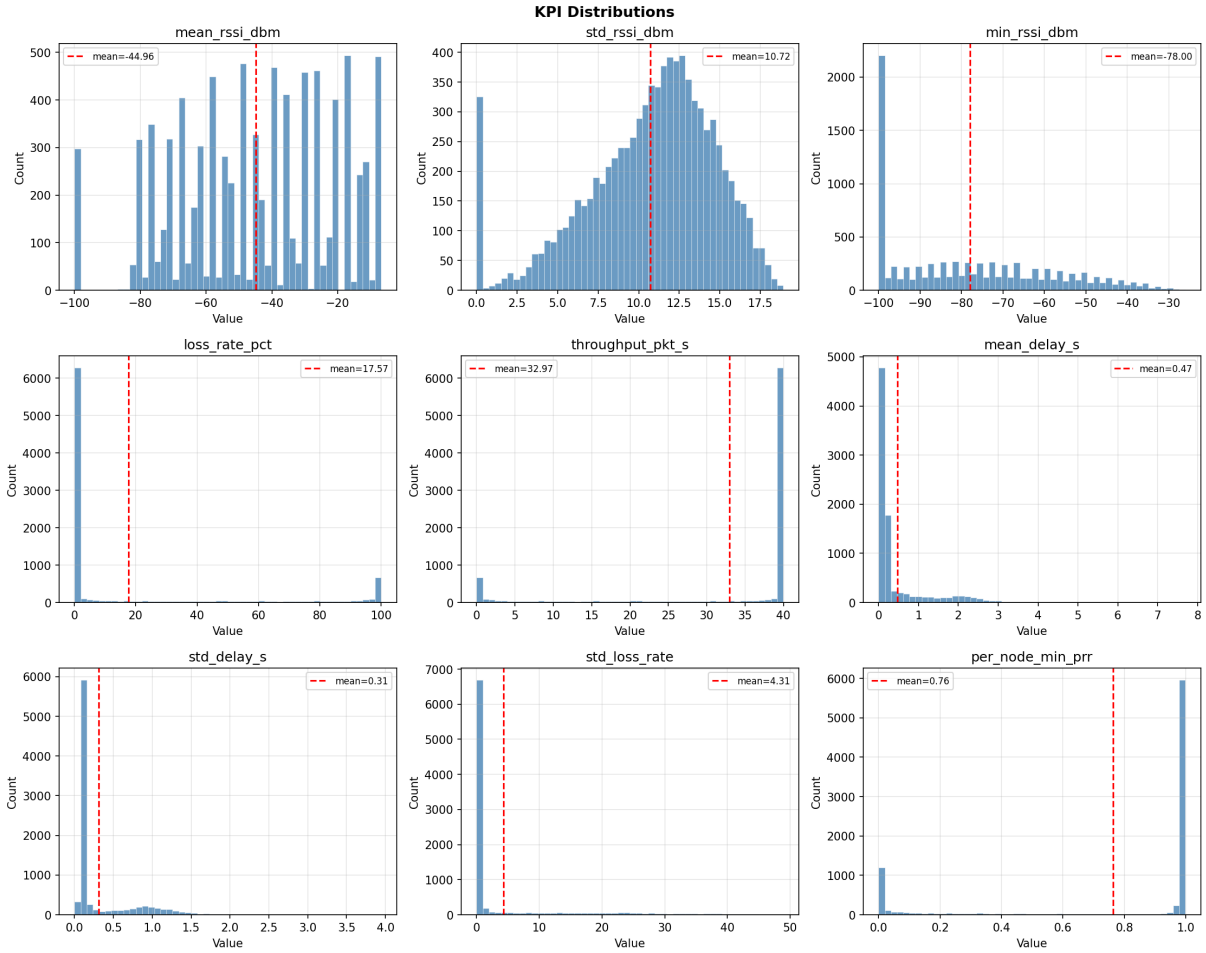


Figure 3.3: Scenario 1 (light traffic) dataset: distribution of all nine base KPIs across the full parameter sweep.

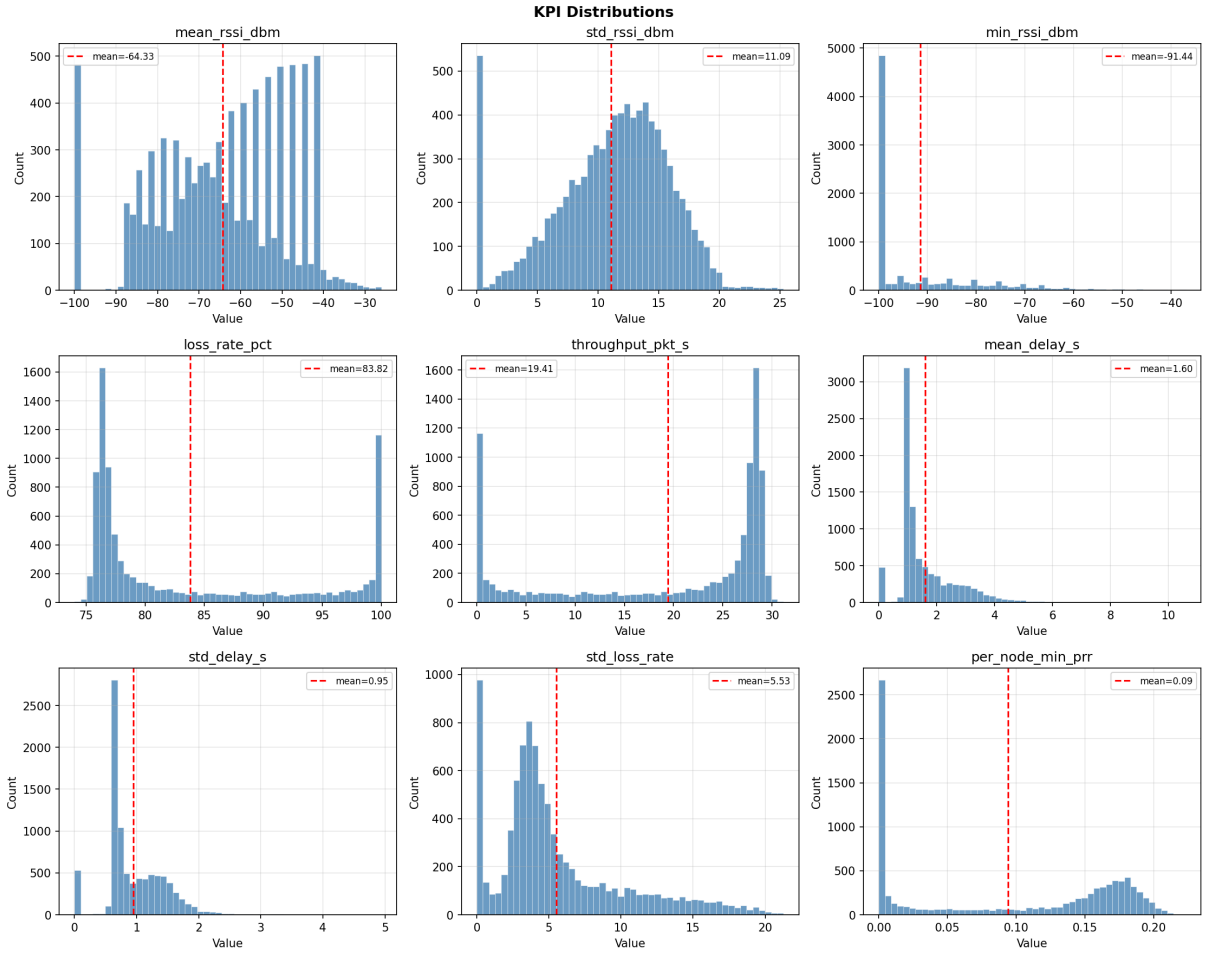


Figure 3.4: Scenario 2 (heavy traffic) dataset: distribution of all nine base KPIs across the full parameter sweep.

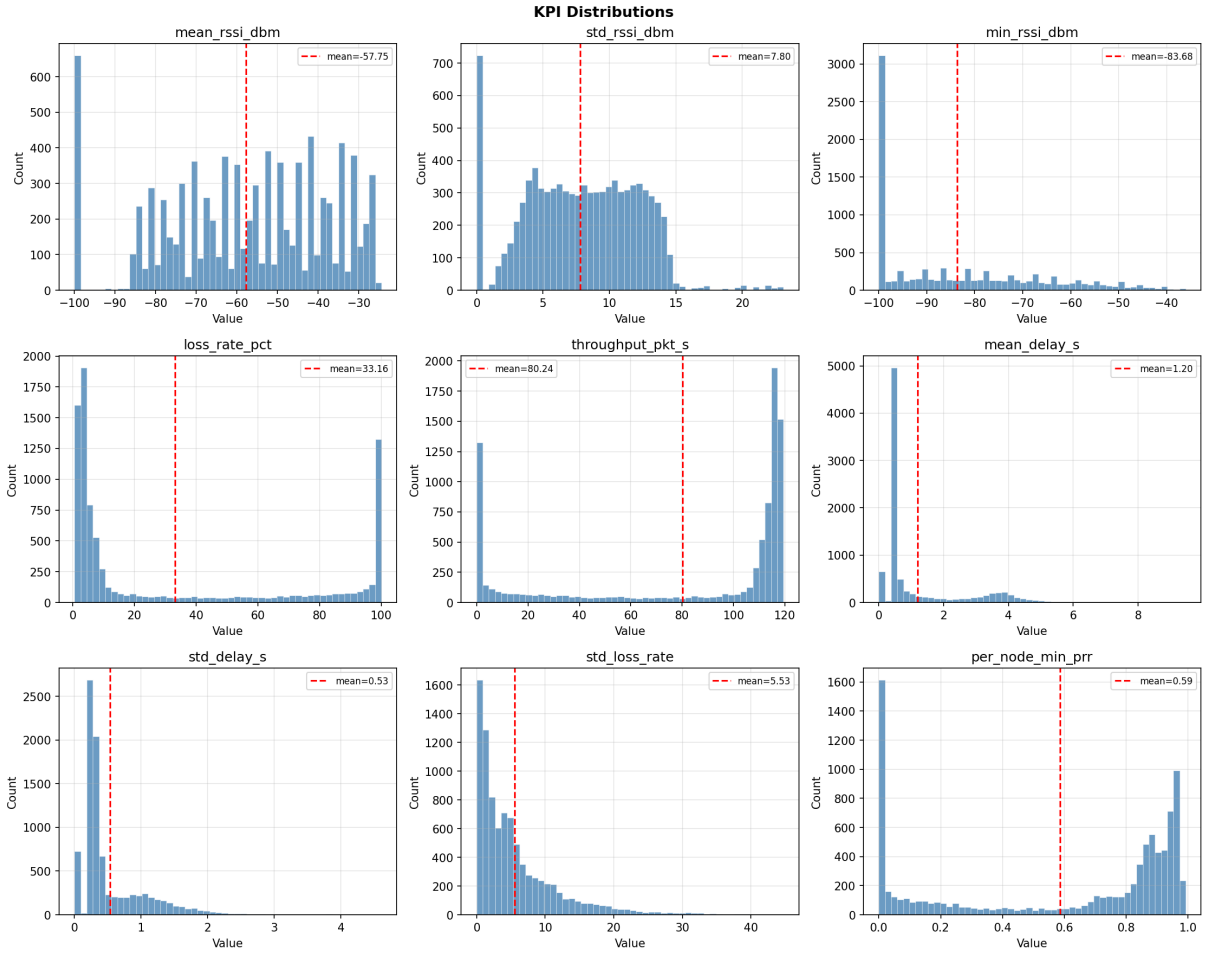


Figure 3.5: Scenario 3 (intermediate traffic) dataset: distribution of all nine base KPIs across the full parameter sweep.

3.5.2 Static Calibration Validation Against FIT IoT-LAB

The static calibration pipeline (Algorithm 1) was validated on the three real FIT IoT-LAB topologies of Table 3.1. For each, real KPIs collected on the physical deployment were passed through the topology's trained g_ϕ to obtain $\hat{\theta}$, which was then fed back into Cooja under the same topology to produce simulated KPIs for direct comparison.

Scenario 1: Light Traffic

The first scenario presents a near-zero-loss regime, with 10 senders each transmitting 5 packets per second. Figure 3.6 shows the physical deployment. From the real KPIs, the model predicted $\hat{\theta} = (\text{RSSI}_{50\%}, \text{PLE}, \sigma) = (-94.77 \text{ dBm}, 1.516, 18.735)$. Table 3.5 compares the resulting simulated KPIs against the real measurements.

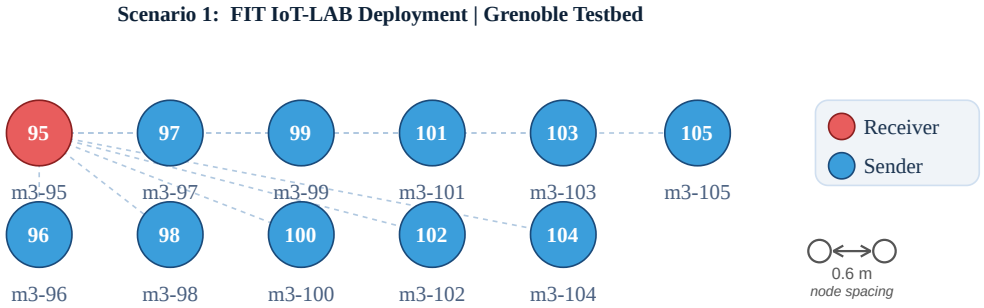


Figure 3.6: Scenario 1 physical deployment at the Grenoble testbed: one receiver and 10 sender nodes.

KPI	Real (FIT IoT-LAB)	Simulated (Cooja)	Δ
Loss rate	0.10%	0.32%	+0.22 pp
Throughput	39.87 pkt/s	39.87 pkt/s	0.00
Mean delay	0.278 s	0.229 s	−0.049 s
Mean RSSI	−77.24 dBm	−76.61 dBm	+0.63 dBm

Table 3.5: Scenario 1 (light traffic) behavior-space validation: real versus simulated KPIs after calibration with the unbalanced, naturally distributed dataset (Section 2.3.4).

Agreement is close across all four KPIs: throughput matches exactly to two decimal places, loss rate is reproduced within a fraction of a percentage point, and RSSI within a fraction of a decibel. This is the regime the raw dataset is most densely populated in (Section 2.3.4), so close agreement here is expected; the more demanding test is Scenario 2, where loss is high and the dataset's coverage of that regime is comparatively sparser.

Scenario 2: Heavy Traffic

The second scenario sits at the opposite end of the loss spectrum: 20 senders each transmitting 15 packets per second, the topology used as the worked example throughout Section 2.3.3, with real loss around 85%. Figure 3.7 shows the physical deployment. The model predicted $\hat{\theta} = (-54.23 \text{ dBm}, 3.027, 15.194)$, a substantially higher Path Loss Exponent than in Scenario 1, consistent with a more attenuated, congested environment. Table 3.6 compares the resulting KPIs.

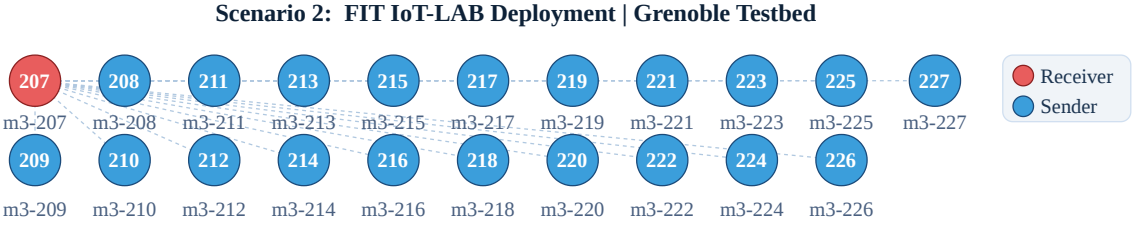


Figure 3.7: Scenario 2 physical deployment at the Grenoble testbed: one receiver and 20 sender nodes.

KPI	Real (FIT IoT-LAB)	Simulated (Cooja)	Δ
Loss rate	85.03%	85.54%	+0.51 pp
Throughput	17.5 pkt/s	17.35 pkt/s	−0.15 pkt/s
Mean delay	2.273 s	2.397 s	+0.124 s
Mean RSSI	−69.92 dBm	−61.35 dBm	+8.57 dBm

Table 3.6: Scenario 2 (heavy traffic) behavior-space validation: real versus simulated KPIs.

Loss rate, throughput, and delay are all reproduced closely, well within one percentage point or a fraction of a second. Mean RSSI is the exception: an 8.57 dBm gap, the simulated RSSI is substantially less negative (stronger) than the real measurement. The cause is a known limitation of the simulator rather than a failure of the calibration procedure: in this topology, 20 nodes transmit simultaneously, and the resulting collisions degrade the RSSI actually measured by the real receiver, an interference effect. Cooja's LogisticLoss model, by contrast, computes RSSI purely as a function of distance and the calibrated propagation parameters (Section 1.3); it has no mechanism to represent RSSI degradation caused by concurrent transmissions colliding on the channel. The calibration procedure cannot compensate for a physical effect the underlying simulator does not model at all, no choice of θ changes what LogisticLoss is capable of representing. Notably, this gap is isolated to RSSI: loss rate, throughput, and delay, the KPIs that most directly determine whether the digital twin is behaviorally useful, are all reproduced with high fidelity despite it.

Scenario 3: Intermediate Traffic

The third scenario, 5 senders each transmitting 15 packets per second, is the topology behind the real interference experiment used to validate the adaptive extension (Section 3.5.4); its Phase 0 baseline (main senders only, no interference) is itself a real deployment under this topology, and its trained g_ϕ's regression performance is reported in Table 3.3.

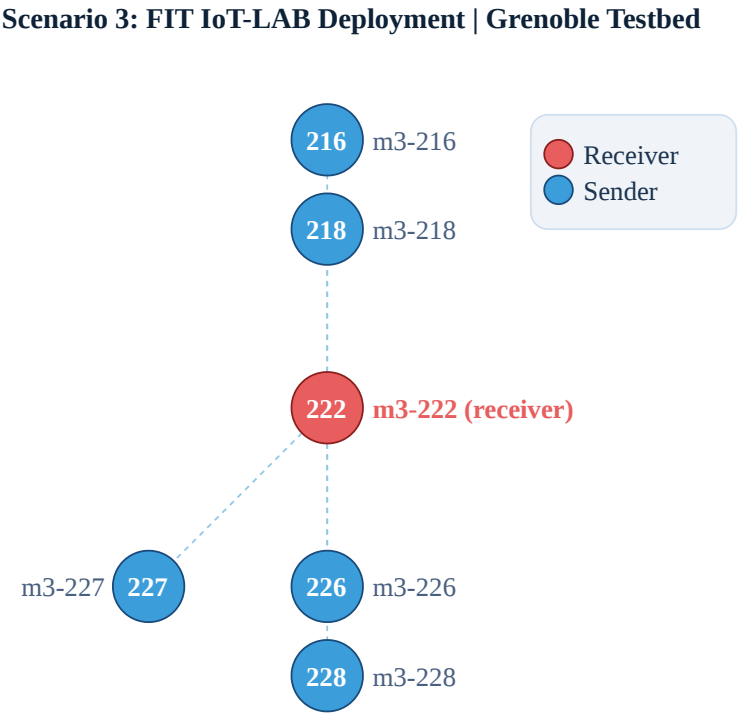


Figure 3.8: Scenario 3 physical deployment at the Grenoble testbed: one receiver and 5 sender nodes, the same topology later extended with interferer nodes for the real interference experiment (Section 3.5.4).

From the real KPIs, the model predicted $\hat{\theta} = (\text{RSSI}_{50\%}, \text{PLE}, \sigma) = (-93.26\,\text{dBm}, 1.676, 25.209)$. Table 3.7 compares the resulting simulated KPIs against the real measurements.

KPI	**Real (FIT IoT-LAB)**	**Simulated (Cooja)**	Δ
Loss rate	15.93%	16.94%	-1.01 pp
Throughput	100.66 pkt/s	99.63 pkt/s	$+1.03$ pkt/s
Mean delay	0.269 s	0.738 s	-0.469 s
Mean RSSI	-80.54 dBm	-71.54 dBm	-9.00 dBm

Table 3.7: Scenario 3 (intermediate traffic) behavior-space validation: real versus simulated KPIs.

Loss rate and throughput are both reproduced closely, within roughly one point of their real values, consistent with the agreement seen in Scenario 1. Mean RSSI and mean delay, however, both show a substantial gap: simulated RSSI is 9 dB less negative (stronger) than the real measurement, in the same direction and of comparable magnitude to the 8.57 dB gap observed in Scenario 2, suggesting the same underlying cause, RSSI degradation from concurrent transmissions colliding on the channel, an effect LogisticLoss cannot represent regardless of calibration.

3.5.3 Adaptive Validation: Synthetic Data

Before trusting the adaptive loop on real, noisy data, it was first validated on a synthetic loss-rate trace, built on Scenario 2's trained model and deployment (Section 3.5.2), covering four distinct, deliberately engineered regimes, chosen so that the ground-truth timing and severity of every transition is known exactly: STABLE (85% loss), DEGRADED (92%), RECOVERY (78%), and CRITICAL (96%). Because the true regime boundaries are known in this synthetic setting, the detection loop's behavior, whether it fires at each transition, how many samples of delay it incurs, and whether it produces false positives while a regime is stable, can be checked directly against ground truth before being trusted on real, noisy FIT IoT-LAB data in Section 3.5.4. Figure 3.9 summarizes loss rate, delay, and throughput across the four 15-minute windows corresponding to the four regimes; the mean RSSI panel for this figure is still being finalized and is omitted here.

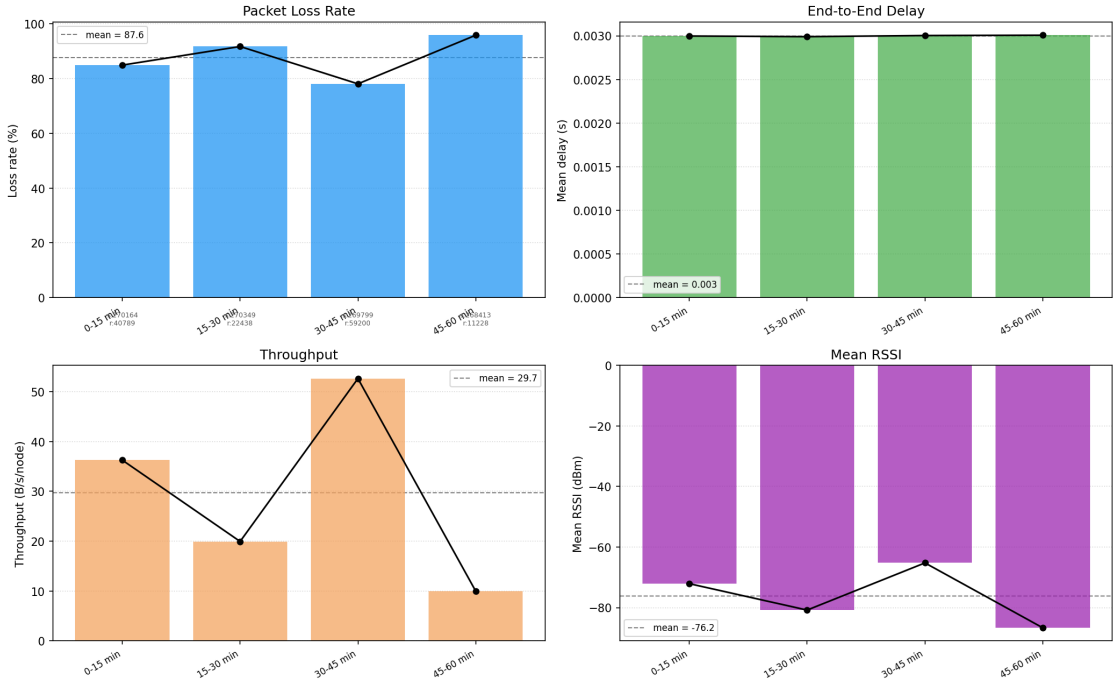


Figure 3.9: Metric evolution across the four synthetic regimes (Scenario 2 deployment), 15-minute windows: packet loss rate, end-to-end delay, and throughput.

Loss rate is the KPI monitored here, and throughout the adaptive loop in Section 3.5.4, since it is one of the four primary KPIs established in Section 2.3.2 and the one most directly affected by interference and congestion. Nothing in the detection-and-recalibration mechanism is specific to loss rate so the same pipeline would apply unchanged if throughput, delay, or mean RSSI were monitored instead.

Figure 3.10 shows the full loss-rate trace at finer time resolution than Figure 3.9's four 15-minute regime bins: the x-axis here steps through the sliding 30-second windows the detector actually observes one at a time, each advancing 10 seconds past the previous one, the same windowing the Page-Hinkley loop of Section 2.3.6 operates on. The solid teal line is the real

network's rolling mean; the shaded teal band around it is its $\pm 5\sigma$ envelope, five standard deviations of the loss-rate values across the windows contributing to that mean, a visual sense of how noisy the real signal is at each point, not a confidence interval on the mean itself. The dashed vertical lines mark every point the Page-Hinkley detector flagged a change, four of them: three matching the engineered transitions, and one secondary detection within the CRITICAL regime discussed alongside Table 3.8 below. The green square marks the one baseline simulation run before any recalibration, using the parameters obtained from the initial static calibration; the orange triangles mark the simulated loss rate immediately after each subsequent recalibration.

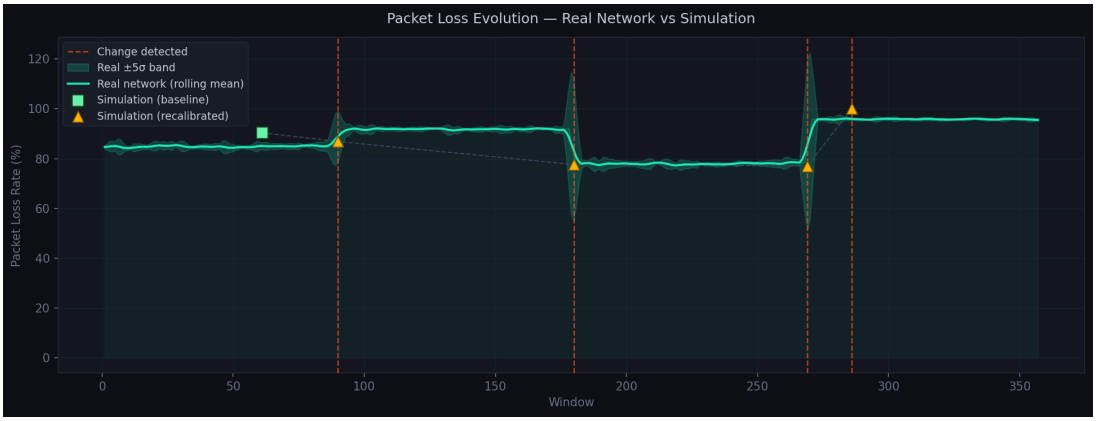


Figure 3.10: Packet loss evolution across the four synthetic regime transitions: real network rolling mean (teal line) with its $\pm 5\sigma$ band (shaded), Page-Hinkley detection points (dashed vertical lines), the initial baseline simulation (green square), and the simulated loss rate after each recalibration (orange triangles).

At each of the three transition-matching detection points, the orange triangle lands close to the real curve's level for that regime, STABLE, DEGRADED, RECOVERY, or CRITICAL in turn, confirming that the detection-and-recalibration loop tracks every engineered transition correctly under these controlled, known conditions, before it is asked to do the same on real, noisy data in Section 3.5.4. The exact detection delays and the one exception to this clean picture are detailed below.

Table 3.8 lists every calibration event across the run: the initial baseline calibration (window 61, before any regime change), the three detections that correspond to the three engineered transitions, and one further detection inside the CRITICAL regime once it had already settled. Detection delay is measured against the real trace itself, the window at which the real loss rate first departs from its previous plateau: for STABLE→DEGRADED, the real signal first moves at window 89 and the detector fires at window 90, a delay of one window (10 s); for DEGRADED→RECOVERY, the shift begins at window 179 and is caught at window 180, again one window; for RECOVERY→CRITICAL, the detector fires at window 269, the same window the real signal begins rising, a delay of essentially zero. Across the 353 windows belonging to the three stable plateaus (STABLE, DEGRADED, RECOVERY), not one produced a false detection, the detector stayed silent throughout every period where nothing had actually changed.

Window	Time (s)	Event	Real loss (%)	PLE	AWGN	$\text{RSSI}_{50\%}$	Δloss (pp)
90	890–920	STABLE → DEGRADED	89.23	3.076	5.047	−67.84	2.38
180	1790–1820	DEGRADED → RECOVERY	81.71	3.326	6.075	−81.12	4.11
269	2680–2710	RECOVERY → CRITICAL	84.49	3.511	7.180	−85.73	7.49
286	2850–2880	Secondary detection, within CRITICAL	96.03	1.998	4.108	−43.68	3.97

Table 3.8: Every calibration event across the synthetic 4-regime run: the initial baseline, the three detections matching the three engineered transitions, and one secondary detection inside the already-settled CRITICAL regime.

The fourth detection, at window 286, does not correspond to a real regime transition, the real loss rate at that point (96.03%) is already sitting on the CRITICAL plateau, essentially unchanged from the surrounding windows. It reflects the detector firing on continued fine-grained variability within an already-stable regime rather than a missed or spurious transition in the usual sense, and it stands out from the other three events in its predicted parameters as well: a markedly lower PLE (1.998 against 3.5 at window 269) and an RSSI Inflection Point (-43.68 dBm) far above the others, which pushed the resulting simulation to a degenerate near-total-loss configuration (100% simulated loss, 0 throughput). This single event is the one place the synthetic validation does not read as cleanly as the rest, and is worth keeping in mind alongside the otherwise clean detection-delay and false-positive results above.

3.5.4 Adaptive Validation: Real Interference Scenario

The adaptive loop was then validated on a real FIT IoT-LAB interference experiment, built on Scenario 3's trained model and deployment (Section 3.5.2). The topology consists of one receiver (node m3-222), 5 main sender nodes, and two interferer groups, all transmitting on the same channel once activated. The experiment runs in four phases, controlled by on-device timers rather than by the detector itself, so the ground-truth timing of each transition is known independently of the Page-Hinkley detector being validated: Phase 0 is a stable baseline with only the main senders active; at 10 minutes, interferer group 1 joins; at 20 minutes, interferer group 2 also activates, with both groups transmitting simultaneously alongside the main senders; at 30 minutes, all interferers stop and the network returns to the baseline regime for the remaining 30 minutes. Table 3.9 summarizes the experimental setup, and Table 3.10 details the four phases.

Before subjecting the captured logs to the adaptive pipeline, it was first necessary to confirm that the introduced interferers actually produced the disruption the experiment design assumed, rather than, say, being too weak or too far from the receiver to have a measurable effect. Fig-

Field	Value
Experiment	`interference_20260619_120409` (ID 440826)
Date	Fri Jun 19 2026, 13:10:11 CEST
Site	Grenoble
Receiver	m3-222
Main senders	m3-218, 216, 226, 228, 227
Interferer group 1	m3-220, 223, 214, 230
Interferer group 2	m3-224, 225, 221, 212
Stable / interference phase duration	10 min each
Total duration	60 min

Table 3.9: Experimental setup for the real FIT IoT-LAB interference scenario.

Phase	Interval	Active senders
STABLE	[0, 10) min	Main senders only
INTERFERE-1	[10, 20) min	Main senders + interferer group 1
INTERFERE-2	[20, 30) min	Main senders + both interferer groups
RECOVERY	[30, 60) min	Interferers stopped, back to baseline

Table 3.10: The four phases of the interference experiment, controlled by on-device timers.

ure 3.11 is that check: it summarizes loss rate, delay, throughput, and mean RSSI across the six 10-minute windows spanning the experiment, and all four peak or trough exactly where the phase design predicts, most sharply once interferer group 2 joins at the 20-minute mark, the point at which both interferer groups transmit simultaneously alongside the main senders, before recovering back to baseline once all interferers stop at 30 minutes. Only once this was confirmed, that the logs genuinely contained the assumed disturbances and not just noise, were they handed to the adaptive pipeline to test whether it could detect and correctly react to changes now known to be real.

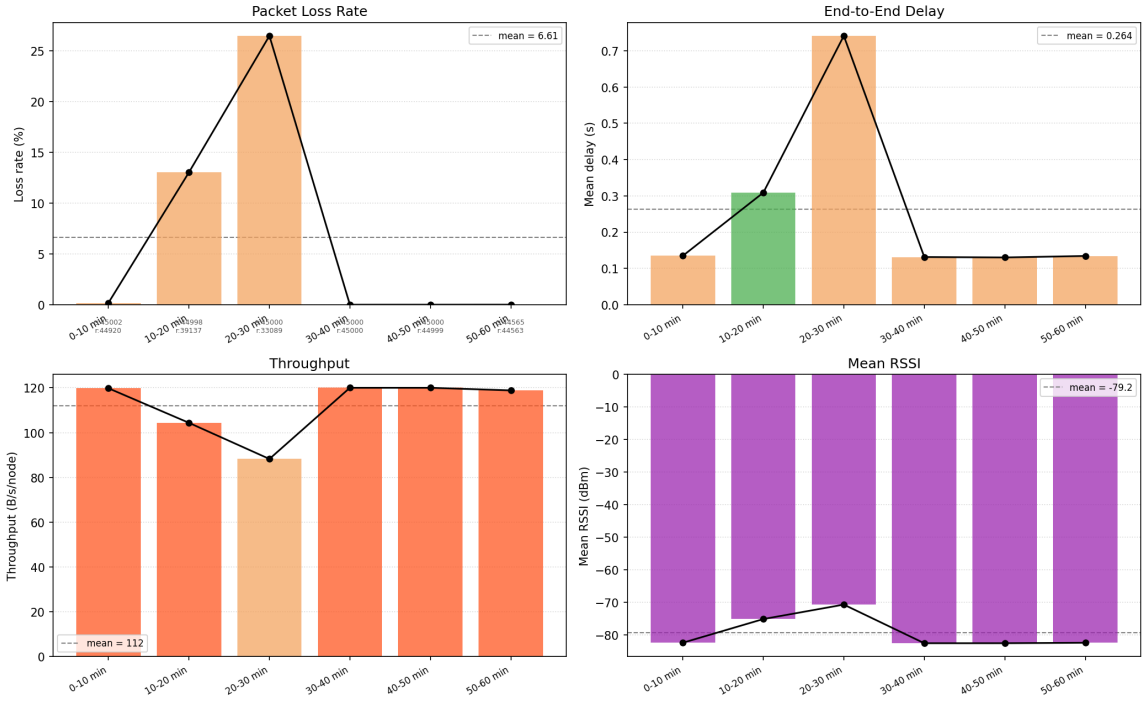


Figure 3.11: Metric evolution across the interference experiment, 10-minute windows: packet loss rate, end-to-end delay, throughput, and mean RSSI, all four peaking or troughing during the [20,30) min window when both interferer groups are active simultaneously, confirming the interferers produced the intended disruption before these logs were fed to the adaptive pipeline.

Figure 3.12 shows the physical deployment behind these logs. Critically, the simulator has no explicit knowledge of the interferers at any point during the adaptive pipeline's operation, it only ever observes the resulting KPIs, so successfully recalibrating through each transition demonstrates the adaptive extension responding to interference it was never told about, inferring new radio parameters purely from its effect on observed network behavior.

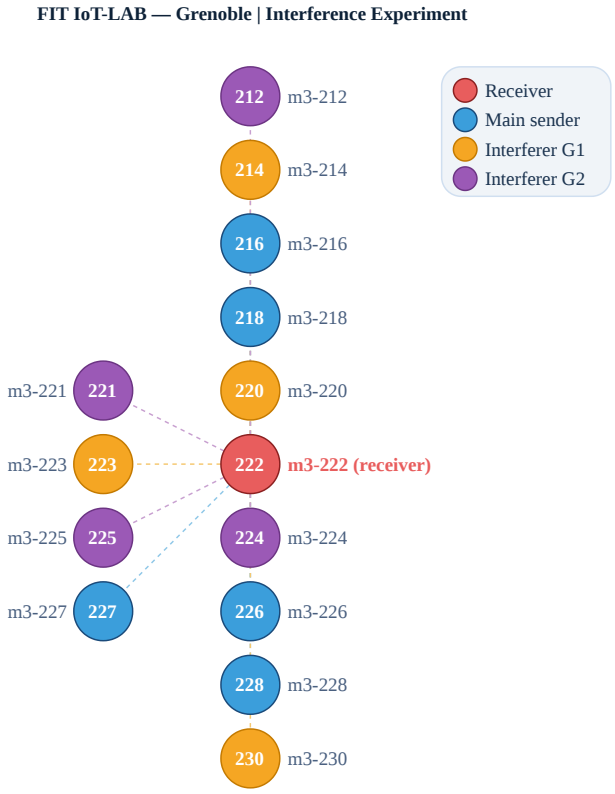


Figure 3.12: Physical node topology for the interference experiment: receiver (m3-222), main senders, and the two interferer groups at the Grenoble site.

Figure 3.13 makes that blind spot visible directly: a screenshot of Cooja's own simulator GUI, captured mid-experiment, at the moment the adaptive pipeline triggered a recalibration and generated new parameters. **Only the main senders and the receiver appear in the simulation**, no interferer nodes exist in Cooja at all, since the simulator has no mechanism to represent them, yet the recalibrated parameters still produce simulated behavior aligned with what the real, interference-degraded network was doing at that moment. The alignment comes entirely from the adaptive loop reading the effect of the interference in the KPIs and adjusting $\hat{\theta}$ accordingly, not from the simulator modeling the interferers themselves.

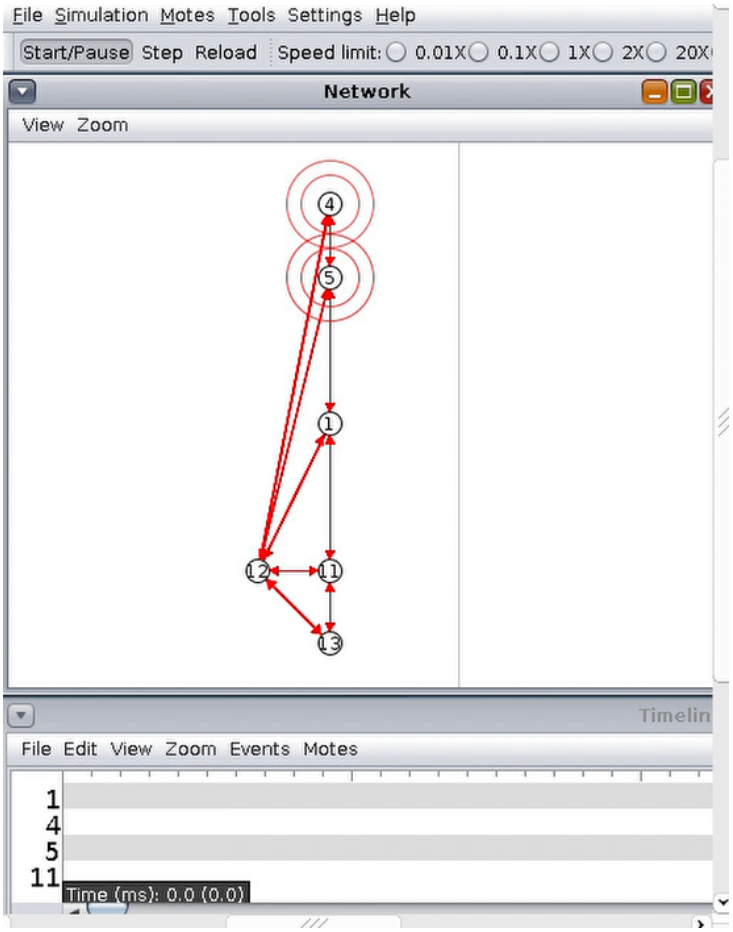


Figure 3.13: Cooja simulator GUI screenshot captured during the adaptive pipeline, at the moment a recalibration was triggered: only the 5 main senders and the receiver exist in the simulation, no interferer nodes are present, since the simulator has no way to represent them; the recalibrated parameters nonetheless track the real, interference-degraded network.

Figure 3.14 shows the same kind of trace as Figure 3.10 above, but built from the real, noisy experiment rather than a synthetic one: the solid teal line is the real network's rolling mean loss rate, the shaded band around it its $\pm 5\sigma$ envelope, the dashed vertical lines the Page-Hinkley detection points, and the orange triangles the simulated loss rate immediately after each recalibration.

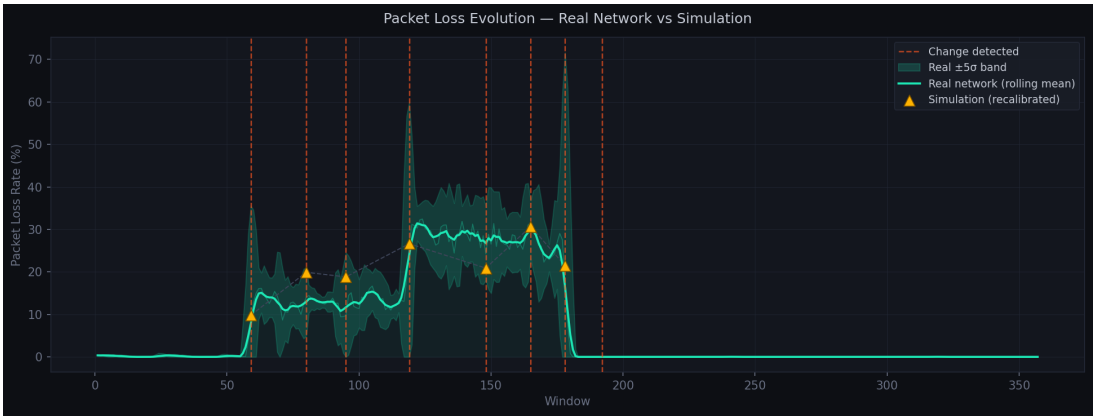


Figure 3.14: Packet loss evolution across the experiment: the real measurement (with its rolling $\pm 5\sigma$ band), the Page-Hinkley detection points (dashed vertical lines), and the simulated loss rate after each recalibration (triangles).

Monitoring Interface

The adaptive loop is also exposed through a web interface, introduced in Section 2.3.6 and shown here in action during this experiment; before describing its components it is worth being clear that what follows is a tour of that interface's layout, not the underlying results themselves. Figure 3.15 shows snapshot taken well into the run, capturing the interface's full behavior in one frame. It is organized into a configuration sidebar on the left, controlling the log file being replayed, the sliding window and step size, and playback speed; a pipeline-state indicator tracking the current stage (reading the log, extracting KPIs, change detection, NN inference, Cooja simulation, reading results); a live packet-loss chart at the top of the main panel; four summary metrics (windows processed, changes detected, average ΔRSSI, average Δloss); and a scrolling window-by-window feed showing each window's status, time range, and real KPIs, updated with predicted parameters and fidelity once a recalibration completes. By this point in the run, all of this is visible at once: the chart already shows all four detection markers reached so far, the feed includes both ordinary stable windows and one highlighted, recalibrated window with its predicted parameters and resulting fidelity, and the console log at the bottom records the run's history, an earlier recalibration, a subsequent change detection, and the recalibration that followed it.

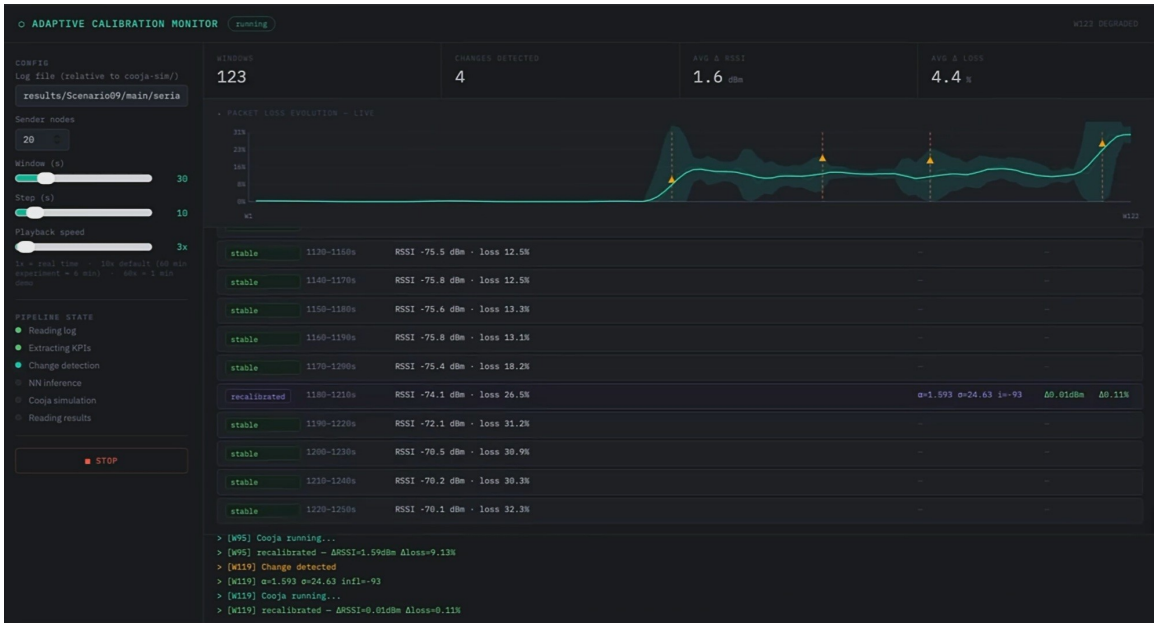


Figure 3.15: The monitoring interface at window 123 (DEGRADED phase): stable windows, the four detection markers reached so far, and a completed recalibration (window 119, ΔRSSI = 0.01 dBm, Δloss = 0.11%) are all visible together, alongside the console log recording the run's history.

Quantitative Results

Across the 357 monitoring windows of the full experiment, the Page-Hinkley detector flagged 8 changes. Table 3.11 lists each one, the predicted parameters, and the resulting fidelity against the real KPIs of that window.

Window	Time (s)	Real loss (%)	PLE	AWGN	$\text{RSSI}_{50\%}$	Δloss (pp)
59	580–610	7.43	0.99	20.86	−104.96	2.37
80	790–820	14.06	1.73	24.03	−93.44	5.85
95	940–970	9.69	1.67	24.18	−95.76	9.13
119	1180–1210	26.53	1.59	24.63	−92.99	0.11
148	1470–1500	24.25	1.82	25.26	−92.13	3.36
165	1640–1670	31.56	1.78	25.33	−90.43	0.86
178	1770–1800	17.73	1.66	25.07	−94.45	3.76
192	1910–1940	0.00	*skipped: loss $<$ 2%, recalibration guard (Section 3.3)*			

Table 3.11: Every Page-Hinkley detection event across the interference experiment, with the resulting recalibration's predicted parameters and loss-rate fidelity, or the reason recalibration was skipped.

Of the 8 detected changes, 7 triggered a full recalibration; the eighth (window 192) coincides with the network having already fully recovered (real loss rate at exactly 0%) and was correctly skipped by the recalibration guard introduced in Chapter 2, avoiding an unnecessary Cooja run. The full recalibration itself is completing quickly enough to keep pace with live monitoring; MLP inference takes a fraction of a second (Section 2.3.1), and a representative Cooja run at

this topology's unlimited simulation speed completed a 3-minute simulated duration in 41.4 seconds of wall-clock time, a $4.35\times$ speedup over real time, for a window where the previous calibration was already adequate. Across the 7 completed recalibrations, mean absolute fidelity was $|\Delta\text{RSSI}| = 1.34$ dBm, $|\Delta\text{loss}| = 3.63$ percentage points, $|\Delta\text{throughput}| = 4.21$ pkt/s, and $|\Delta\text{delay}| = 0.311$ s, consistent with the close agreement already observed in the static calibration validation of Section 3.5.2. Window 119, immediately following the transition into the INTERFERE-2 phase (both interferer groups active), shows the tightest fidelity of all seven ($\Delta\text{loss} = 0.11$ pp), while windows 80 and 95, both inside the more volatile INTERFERE-1 phase, show the largest throughput and delay residuals, consistent with that phase's higher measurement variability visible in Figure 3.14's $\pm 5\sigma$ band.

3.5.5 Loss-Rate Distribution

As discussed in Section 2.3.4, no rebalancing was applied to any of the three training datasets; each retains its naturally generated distribution across loss-rate regimes. These distributions were already shown per topology in Figures 3.3–3.5 (Section 3.5.1) alongside the other eight base KPIs, rather than repeated here. Their relevance to the adaptive extension is direct: Scenario 2's dataset, whose `loss_rate_pct` distribution (Figure 3.4) is heavily weighted toward both very low and very high loss with comparatively little density in between, is exactly the model the synthetic 4-regime validation (Section 3.5.3) draws on when recalibrating between the STABLE, DEGRADED, RECOVERY, and CRITICAL regimes; Scenario 3's dataset similarly underlies every recalibration in the real interference scenario (Section 3.5.4, Table 3.11).

3.6 Discussion

Taken together, the results of this chapter answer the two questions Chapter 2 set out to ask: can g_ϕ be learned well enough to close the simulation-to-reality gap, and can the resulting calibration be kept valid as the network changes.

On the first question, the parameter-space results of Section 3.5.1 and the behavior-space results of Section 3.5.2 tell a consistent story once read together. PLE and AWGN sigma are recovered with $R^2 \geq 0.95$ in every topology tested, RSSI Inflection Point noticeably less so, yet the behavior-space validation shows that this asymmetry in parameter-space accuracy does not translate into an asymmetry in the KPIs that matter most: loss rate, throughput, and delay all land within a fraction of their real values across both Scenario 1 and Scenario 2, the two topologies at opposite ends of the loss spectrum. RSSI Inflection Point being the harder of the three targets to regress precisely, in other words, does not by itself prevent the digital twin from behaving like its real counterpart. The one place behavior-space agreement does break down, Scenario 2's 8.57 dBm RSSI gap, was traced not to a weakness in g_ϕ but to a physical effect, collision-induced RSSI degradation under simultaneous transmission, that LogisticLoss has no

mechanism to represent at all, a limitation of the simulator this thesis calibrates rather than of the calibration procedure itself. That distinction matters for how the result should be read: the gap marks a boundary of what this radio model can be made to reproduce, not a failure of the method to reach that boundary.

On the second question, the two-stage validation of Sections 3.5.3 and 3.5.4 shows the adaptive extension working end to end in both a controlled and an uncontrolled setting. The synthetic validation, where every transition's timing and severity is known in advance, confirms the detection-and-recalibration loop is wired correctly: all four engineered regime changes are caught and recalibrated toward. The real interference experiment then repeats that test under conditions the pipeline cannot control, transmission collisions it was never aware of, measurement noise visible in the $\pm 5\sigma$ bands of Figure 3.14, and it holds up: 7 of 8 detected changes were recalibrated with a mean loss-rate fidelity of 3.63 percentage points, and the eighth was correctly identified as not needing recalibration at all by the guard introduced in Chapter 2, itself a small but concrete piece of evidence that the pipeline's design choices behave as intended outside of the conditions they were designed under. Read alongside the static results, the adaptive extension is not compensating for a weak static calibration, it is preserving a strong one as the network drifts away from the conditions it was calibrated under.

The results are not without gaps. The exact detection delay, how many samples elapse between a true phase transition and the corresponding Page-Hinkley alarm, is visible qualitatively in Figures 3.10 and 3.14 but has not been quantified precisely in either setting. This gap does not change the overall picture established above, but would sharpen it.

Beyond reproducing observed behavior, a calibrated digital twin is also a testbed for questions that were previously unanswerable in simulation: whether a given configuration choice, rather than the radio channel's own idiosyncrasies, is responsible for an observed change in performance. Work already underway applies this to the MAC layer, using the calibrated channel to evaluate configuration changes such as queue sizing and backoff tuning under conditions the simulator can now be trusted to represent; the general conclusion develops this direction further.

3.7 Conclusion

This chapter set out to show that the method proposed in Chapter 2 actually works, and the results support that claim on both of its halves. The static calibration pipeline recovers PLE and AWGN sigma with high accuracy across every topology tested, and, more importantly, reproduces real network behavior closely enough in behavior space, on loss rate, throughput, and delay in particular, that its one identified shortfall, an RSSI gap under heavy simultaneous transmission, was traced to a documented limitation of the underlying radio model rather than to the calibration procedure itself. Built on top of that static calibration, the adaptive extension was shown to keep pace with a changing network in both a controlled synthetic setting and a real, noisy FIT IoT-LAB interference experiment, correctly recalibrating through every genuine tran-

sition it encountered and correctly declining to act on the one window where recalibration was not warranted.

Together, these results close the loop opened in Chapter 1: the simulation-to-reality gap this thesis set out to address is substantially narrowed by the static calibration pipeline, and kept narrowed over time by the adaptive extension built on top of it, within the bounds of what the underlying radio model is physically capable of representing. The general conclusion that follows closes this thesis by drawing these contributions together against its original objectives, and points to the specific gaps flagged in Section 3.6 as the most immediate directions for continuing this work.

Conclusion and Perspectives

Key Insights and Contributions

This thesis set out to address the simulation-to-reality gap in Cooja/Contiki-NG as a radio model fidelity problem, and to build a pipeline capable of both closing that gap and keeping it closed as real network conditions change. The results presented across the preceding chapters support that this was achieved, and three contributions stand out.

The first is methodological: rather than calibrating every parameter LogisticLoss exposes, a literature-guided elimination followed by a statistical screening combining Spearman correlation and Partial Dependence Plots narrowed the candidate set from five parameters down to the three actually worth calibrating. This screening did more than reduce dimensionality for its own sake, it caught a failure mode a correlation coefficient alone would have missed: transmitting range showed a correlation magnitude comparable to AWGN sigma, yet its Partial Dependence Plots revealed a clear instability, a reversal in its marginal effect partway through its range, that correlation strength alone could not distinguish from a genuinely useful signal. Excluding it on this basis, rather than on correlation strength alone, was later corroborated in Chapter 3 by a dedicated, unsuccessful attempt to include it as a fourth regression target.

The second contribution is the static calibration pipeline itself. A dataset of 8,721 labeled simulations was generated through a full factorial sweep of the three selected parameters, a computational effort brought down from an estimated 221 sequential days to roughly 9 days by parallelizing across a 24-core compute server, and used to train a Multilayer Perceptron approximating the inverse mapping from network KPIs to LogisticLoss parameters. Validated on three real FIT IoT-LAB deployments spanning light, heavy, and intermediate traffic regimes, this pipeline recovered the Path Loss Exponent and AWGN sigma with high accuracy in every topology tested ($R^2 \geq 0.95$), and, more importantly, reproduced real network behavior closely in the metrics that matter most for a usable digital twin: loss rate, throughput, and delay. The one place this agreement did not hold, an 8.57 dBm RSSI gap under a heavy-traffic, high-collision topology, was traced not to a weakness in the learned model but to a genuine limitation of LogisticLoss itself, which has no mechanism to represent RSSI degradation caused by concurrent transmissions colliding on the channel. Identifying this as a boundary of what the underlying radio model can represent, rather than a shortfall of the calibration procedure, is itself a useful outcome of this validation work.

The third contribution is the adaptive extension. Built around a two-sided Page-Hinkley detector using a true cumulative mean, chosen after an earlier exponential-moving-average de-

sign was found to produce false alarms following genuine phase transitions, this extension was validated in two stages: first on a synthetic trace with known ground truth, where it correctly detected and recalibrated through all three engineered regime transitions with detection delays of at most one monitoring window and zero false positives across the stable regimes, and then on a real, noisy FIT IoT-LAB interference experiment, where it correctly recalibrated through 7 of 8 detected changes with a mean loss-rate fidelity of 3.63 percentage points, and correctly declined to recalibrate the one window where the network had already recovered. Notably, the simulator has no explicit knowledge of the interferers at any point in this process, confirmed directly by inspecting Cooja's own simulation state during a live recalibration, so this result demonstrates the adaptive extension inferring the effect of a real physical disturbance purely from its signature in observed KPIs.

Limitations

Several limitations, most already flagged where they arose, are worth restating together here. The RSSI gap observed under heavy simultaneous transmission is the most significant: it is a genuine boundary of what LogisticLoss can be calibrated to represent, not something a different training procedure or a larger dataset would resolve. The RSSI Inflection Point was also consistently the hardest of the three parameters to regress precisely in parameter space, though this did not propagate into a corresponding weakness in behavior-space validation. On the evaluation side, the exact detection delay of the adaptive extension was quantified precisely for the synthetic scenario but only qualitatively for the real interference experiment. Finally, the static calibration pipeline is not topology-transferable: g_ϕ must be retrained, meaning a new dataset generated and a new model trained, for every topology and traffic pattern it is applied to, a real practical cost demonstrated directly by the three separate datasets and models this thesis required.

Future Work

Several directions follow naturally from where this work leaves off. The RSSI limitation identified under heavy traffic points toward extending or complementing LogisticLoss with an explicit collision-awareness term, or exploring a hybrid radio model capable of representing interference-induced degradation directly rather than leaving it entirely to the calibration procedure to approximate. The topology-transferability limitation suggests investigating whether a single model, perhaps conditioned on topology or traffic-pattern features rather than trained separately for each using transfer learning, could reduce the retraining cost demonstrated in this thesis without sacrificing accuracy. On the adaptive side, quantifying detection delay precisely across a wider range of real interference patterns, and understanding the origin of the one anomalous detection observed in the synthetic validation, would sharpen confidence in the extension's real-world

reliability.

More broadly, and in line with the motivation stated in the abstract, this thesis's calibration pipeline turns the simulator into a reliable testbed for what-if network configuration studies, evaluating a proposed change under a channel model shown in Chapter 3 to track real deployments closely. The most immediate application of this, already underway, targets Contiki-NG's MAC layer: preliminary investigation suggests its default configuration can silently drop packets under high traffic rates on RAM-constrained Z1 motes, well before packets reach the radio layer, which motivates testing several concrete MAC-layer parameters against the real deployment scenarios validated in this thesis: the CSMA/CA minimum and maximum backoff exponents, which govern how aggressively a node backs off after detecting a busy channel; the maximum number of backoff stages permitted before a frame is dropped; the maximum number of MAC-layer frame retransmissions attempted before giving up on a packet; and the MAC queue size itself. None of these questions could be answered with confidence in an uncalibrated simulator, since any observed improvement could equally be an artifact of the simulation-to-reality gap rather than a genuine effect of the configuration change; with that gap closed by the static calibration pipeline and kept closed by the adaptive extension, this MAC-layer exploration is a direct, ongoing continuation of the work presented in this thesis.

Bibliography

Adeogun, R. O. (2019). Calibration of stochastic radio propagation models using machine learning. *IEEE Antennas and Wireless Propagation Letters*, *18*(12), 2538–2542. https://doi.org/10.1109/LAWP.2019.2942221

Adjih, C., Baccelli, E., Fleury, E., Harter, G., Mitton, N., Noël, T., Pissard-Gibollet, R., Saint-Marcel, F., Schreiner, G., Vandaele, J., & Watteyne, T. (2015). FIT IoT-LAB: A large scale open experimental IoT testbed. *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 459–464. https://doi.org/10.1109/WF-IoT.2015.7389098

Baena-García, M., del Campo-Avila, J., Fidalgo, R., Bifet, A., Gavaldà, R., & Morales-Bueno, R. (2006). Early drift detection method. *Proceedings of the 4th International Workshop on Knowledge Discovery from Data Streams*, 77–86.

Bharti, A., Briol, F.-X., & Pedersen, T. (2022). A general method for calibrating stochastic radio channel models with kernels. *IEEE Transactions on Antennas and Propagation*, *70*(6), 3986–4001. https://doi.org/10.1109/TAP.2021.3083761

Bifet, A., & Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. *Proceedings of the 2007 SIAM International Conference on Data Mining*, 443–448. https://doi.org/10.1137/1.9781611972771.42

Dezfouli, B., Radi, M., Razak, S. A., Hwee-Pink, T., & Abu Bakar, K. (2015). Modeling low-power wireless communications. *Journal of Network and Computer Applications*. https://doi.org/10.1016/j.jnca.2014.02.009

Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. In *Advances in artificial intelligence—SBIA 2004* (pp. 286–295, Vol. 3171). Springer. https://doi.org/10.1007/978-3-540-28645-5_29

Ivanov, S., Herms, A., & Lukas, G. (2007). Experimental validation of the ns-2 wireless model using simulation, emulation, and real network. *Communication in Distributed Systems (KiVS), 2007 ITG-GI Conference*, 433–444. https://www.researchgate.net/publication/228619429

Kanaris, L., Sergiou, C., Kokkinis, A., Pafitis, A., Antoniou, N., & Stavrou, S. (2019). On the realistic radio and network planning of iot sensor networks. *Sensors*, *19*(15), 3264. https://doi.org/10.3390/s19153264

Khan, L. U., Han, Z., Saad, W., Hossain, E., Guizani, M., & Hong, C. S. (2022). Digital twin of wireless systems: Overview, taxonomy, challenges, and opportunities. *IEEE Communications Surveys & Tutorials*, *24*(4), 2230–2254. https://doi.org/10.1109/COMST.2022.3198273

Kushalnagar, N., Montenegro, G., & Schumacher, C. (2007, August). RFC 4919: IPv6 over low-power wireless personal area networks (6LoWPANs): Overview, assumptions, problem statement, and goals. https://www.rfc-editor.org/rfc/rfc4919

Miranda, J., Abrishambaf, R., Gomes, T., Gonçalves, P., Cabral, J., Tavares, A., & Monteiro, J. (2013). Path loss exponent analysis in wireless sensor networks: Experimental evaluation. *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, 54–58. https://doi.org/10.1109/INDIN.2013.6622857

Page, E. S. (1954). Continuous inspection schemes. *Biometrika*, *41*(1–2), 100–115. https://doi.org/10.1093/biomet/41.1-2.100

Si-Mohammed, S., & Theoleyre, F. (2025). Per link data-driven network replication towards self-adaptive digital twins. *27th International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*. https://doi.org/10.1109/MSWiM67937.2025.11308908

Singh, A. (2021). Radiowave propagation models for vhf and uhf band. *Scientific and Practical Cyber Security Journal*.

Tarafder, P., Oyekola, A. O., Mannil, J. A., Ahmed, I., Hassan, Z., Rawat, D. B., & Che, W. (2026). Calibrating the digital twin channel: Statistics-consistent sim-to-lab adaptation for w-band industrial ofdm links.

Winter, T., Thubert, P., Brandt, A., Clausen, T. H., Hui, J. W., Kelsey, R., Levis, P., Pister, K., Struempler, M., Vasseur, J.-P., & Alexander, R. K. (2012, March). RFC 6550: RPL: IPv6 routing protocol for low-power and lossy networks. https://www.rfc-editor.org/rfc/rfc6550

Zinonos, Z., & Vassiliou, V. (2012). Radio propagation in industrial wireless sensor network environments: From testbed to simulation evaluation. *Proceedings of the 7th ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks (PM2HW2N)*. https://doi.org/10.1145/2387191.2387209

Webography

Atmel Corporation. (2014). AT86RF231 datasheet [https://ww1.microchip.com/downloads/en/DeviceDoc/doc8111.pdf].

Contiki-NG. (2024). Cooja radio mediums source directory [https://github.com/contiki-ng/cooja/blob/master/java/org/contikios/cooja/radiomediums/].

Contiki-NG Cooja Simulator. (2024). Cooja: The contiki-ng network simulator [https://github.com/contiki-ng/cooja].

Contiki-NG Documentation. (2024). Contiki-ng documentation—platforms [https://docs.contiki-ng.org/].

Contiki-NG LogisticLoss Source. (2024). LogisticLoss.java [https://github.com/contiki-ng/cooja/blob/master/java/org/contikios/cooja/radiomediums/LogisticLoss.java].

FIT IoT-LAB Grenoble Deployment. (2024). Grenoble site deployment documentation [https://iot-lab.github.io/docs/deployment/grenoble/].

IEEE. (2020). *IEEE Standard for Low-Rate Wireless Networks*. Institute of Electrical and Electronics Engineers.