APEX: Automated Parameter Exploration for Low-Power Wireless Protocols

MOHAMED HASSAAN M. HYDHER, Graz University of Technology, Austria
MARKUS SCHUSS, Graz University of Technology, Austria
OLGA SAUKH, Graz University of Technology & Complexity Science Hub, Austria
KAY RÖMER, Graz University of Technology, Austria
CARLO ALBERTO BOANO, Graz University of Technology, Austria

Careful parametrization of networking protocols is crucial to maximize the performance of low-power wireless systems and ensure that stringent application requirements can be met. This is a non-trivial task involving thorough characterization on testbeds and requiring expert knowledge. Unfortunately, the community still lacks a tool to facilitate parameter exploration while minimizing the necessary experimentation time on testbeds. Such a tool would be invaluable, as exhaustive parameter searches can be time-prohibitive or unfeasible given limited testbed availability, whereas non-exhaustive unguided searches rarely deliver satisfactory results. In this paper, we present APEX, a framework enabling an automated and informed parameter exploration for low-power wireless protocols and allowing convergence to the best parameter set within a limited number of testbed trials. We design APEX using Gaussian processes to effectively handle noisy experimental data and estimate the optimality of a parameter combination. After developing a prototype of APEX, we demonstrate its effectiveness by parametrizing two IEEE 802.15.4 protocols across a wide range of application requirements. Our results show that APEX can return the best parameter set with up to 10.6x, 4.5x, 4.3x and 3.25x less testbed trials than traditional solutions based on exhaustive search, greedy approaches, support vector regression and reinforcement learning, respectively.

CCS Concepts: • **Computer systems organization** → *Embedded and cyber-physical systems*; *Sensor networks*; • **Networks** → *Network protocols*; *Network performance evaluation*; *Network performance modeling*; *Network performance analysis*; • **Mathematics of computing** → *Stochastic processes*; *Nonparametric statistics*; *Multivariate statistics.*

Additional Key Words and Phrases: Constraints, Crystal, D-Cube, Energy-efficient operation, Framework, Gaussian processes, Internet of Things, MAC protocols, Parametrization, Performance, PRR, Optimization, Reliability, RPL, Testbeds, Wireless sensor networks.

ACM Reference Format:
Mohamed Hassaan M. Hydher, Markus Schuß, Olga Saukh, Kay Römer, and Carlo Alberto Boano. 2025. APEX: Automated Parameter Exploration for Low-Power Wireless Protocols. 1, 1 (September 2025), 34 pages. https://doi.org/XXXXXXX.XXXXXXX

Authors' Contact Information: Mohamed Hassaan M. Hydher, m.h.mohamedhyder@tugraz.at, Graz University of Technology, Graz, Austria; Markus Schuß, markus.schuss@tugraz.at, Graz University of Technology, Graz, Austria; Olga Saukh, saukh@tugraz.at, Graz University of Technology & Complexity Science Hub, Graz/Vienna, Austria; Kay Römer, roemer@tugraz.at, Graz University of Technology, Graz, Austria; Carlo Alberto Boano, cboano@tugraz.at, Graz University of Technology, Graz, Austria.

1 Introduction

Low-power wireless (LPW) systems have become an integral part of the Internet of Things (IoT) and lay the foundations for a wide spectrum of applications that are of utmost importance for our society. Such applications include smart and efficient buildings, precision agriculture, smart health, asset tracking, and smart manufacturing – all very different in their nature and performance requirements. For example, in smart farming systems, it is often important to maximize energy efficiency, so to minimize the need to replace batteries and increase user acceptance. On the contrary, in industrial IoT systems, high reliability and short delays are pivotal to ensure correct operation and quickly detect anomalies. As there is no "one-size-fits-all" solution in the IoT realm, a large number of LPW communication technologies and protocols have been proposed to satisfy largely different application requirements. These LPW solutions provide support for highly-diverse network topologies and medium access control (MAC) strategies – thereby giving developers the chance to customize to the fullest extent their system to the application at hand in order to maximize performance [20].

Challenges. Customizing a system for a specific application involves picking the right networking stack and configuring its protocols to maximize performance: unfortunately, this is not a trivial task, as several challenges need to be tackled.

Expert knowledge required. LPW protocols often rely on a set of parameters that must be chosen meticulously, and their choice is not straightforward. For instance, consider the radio's transmission power parameter (*tx_power*) in a LPW protocol. This parameter directly influences both energy consumption and packet reception ratio (PRR). The relationship between *tx_power* and PRR can be complex. While increasing *tx_power* may initially improve PRR, it can also lead to excessive interference in dense networks, ultimately reducing PRR in certain conditions. Similarly, one might intuitively expect the overall energy consumption to increase with higher *tx_power*; however, this may not necessarily be the case (as we will illustrate in Fig. 2). This complexity makes it hard for users to identify the best parameters for a given application without a comprehensive understanding of these intricate dynamics. Moreover, the performance of MAC protocols can vary drastically as a function of the employed parameters, as it was shown in the context of several short-range and long-range LPW technologies [8, 13, 25, 44, 45, 47, 49]. While default settings for each parameter exist, they are commonly meant for the general case, and are often sub-optimal for a given application [48]. A careful configuration and customization of LPW protocols to the application at hand is hence a must, but it represents a complex and tedious task that requires specialized expertise and that strongly depends on aspects such as the employed platform, network topology and stack, as well as the type and amount of transmitted data. Therefore, in order to properly understand the role of each parameter, to quantitatively assess whether certain application requirements can be met, and to compare the performance of different configurations, one commonly needs to also gather experimental data. However, this is also not trivial, as discussed next.

Experimentation is expensive. Parameter optimization ideally takes place directly in the actual deployment scenario. However, testing and optimizing a solution at the deployment site is typically not viable, due to the high costs and labor involved, as well as due to the inability to perform repeatable and controlled experiments over a prolonged time. As a result, simulation platforms and testbed facilities are often used to test and configure the developed solutions in more convenient settings and in environments that mimic the conditions of real-world deployments. Simulation platforms for LPW systems such as COOJA [42], OMNeT++ [53], and *ns-3* [11] allow fully-controllable and repeatable experiments, thus enabling parameter exploration without the need for expensive field trials. However, these simulators often fail to capture key environmental factors, such as temperature fluctuations, RF interference, and hardware variability, which can significantly impact real-world performance[1]. In contrast, LPW testbeds, such as FIT IoT-LAB [1], Indriya [3], and D-Cube [47], enable the evaluation of protocol performance on actual hardware (HW) in real-world settings, and may even allow to control environmental conditions such as RF interference [46] and temperature variations [6]. For this reason, experimentation on LPW testbeds is typically preferred [7]. However, testbeds – especially public facilities – are often shared by several users, and their availability (i.e., the usable experimentation time) may hence be limited, e.g., one might only get one hour of experimentation time per day. This makes an exhaustive exploration of all possible parameters – or an exploration giving a sufficient statistical significance – impractical; especially considering that testbed runs may need to have a minimum duration to allow the network to setup and stabilize [5], or that certain metrics (such as the PRR) may require the exchange of a large number of packets to be computed with sufficient granularity. For instance, when a node transmits only 20 packets within a testbed run, a receiver can compute the corresponding PRR only with a granularity of 5%.

Lack of adequate tools. Unfortunately, to date, the community still lacks a simple tool to automatically parametrize LPW protocols based on the requirements of a given application. Existing work has proposed frameworks requiring extensive testbed usage (e.g., by exhaustively testing all parameter combinations [21]), demanding a deep understanding of the protocol internals [55], or requiring the user to specify complex models of the environment and the employed hardware, which can be daunting and error-prone for non-experts [41].

Our contributions. In this work, we introduce APEX, a modular framework enabling an automated and efficient parameter exploration for LPW protocols based on data collected using real-world testbeds. Specifically, APEX employs an iterative approach to automatically model a protocol's performance as a function of the exposed parameters based on available experimental data obtained through testbed trials. Using an experimentally-derived model, APEX intelligently selects the next parameter set to be tested using algorithms that minimize the overall number of testbed trials needed to find a solution that meets or is optimal w.r.t. given application requirements[2].

APEX's parameter exploration is fully automatic (i.e., users do not need to be actively involved in the process and do not require in-depth understanding of the protocol under study), and the interaction with the testbed infrastructure is seamless (i.e., APEX can leverage a testbed API to autonomously schedule new tests and refine the models that capture the protocol's performance as a function of certain parameters). To the best of our knowledge, APEX is the first framework of this kind ever proposed to the LPW community.

After presenting the high-level architecture of the proposed framework (§ 3), we discuss in detail the design choices of its inner components, navigating through the nuanced trade-offs (§ 4). Among others, we compare the use of greedy approaches (often preferred due to their simplicity and low computational demands) to the use of approaches based on Gaussian processes, showing that the latter can effectively handle the inherent noise in experimental data and quickly converge to the best possible parameter set. We also enrich APEX with a way to assess the confidence in the results and estimate their proximity to an optimal solution.

We implement a concrete instance of the APEX framework in Python and integrate it with D-Cube [24], one of the most recent and feature-rich public LPW testbeds. We also *open-source* our implementation and traces[3] to enable the

[1]Notably, simulation results can deviate by up to 50% compared to real-world tests performed on testbeds [14].
[2]Note that in black-box optimization problems, it is generally impossible to provide a formal proof of global optimality due to the unknown behavior of the objective function. Instead, APEX provides a structured approach balancing exploration and exploitation, which significantly reduces the probability of getting trapped in local minima. By effectively avoiding local minima, the framework increases the likelihood of finding a solution that meets or is optimal for given application requirements with fewer experimental trials.
[3]The APEX code is available at: http://iti.tugraz.at/APEX.

Protocol	Parameter	Possible values
Crystal	*tx_power*	[-5, -3, -1, 0] dBm
	n_tx	[1, 2, 3, 4]
RPL	*max_link_metric*	[16, 32, 64]
	DIO_interval	[$2^4, 2^8, 2^{12}, 2^{16}$] ms
	Rank_threshold	[4, 8, 12, 16]

Table 1. LPW protocols and parameters considered in this work, and their respective range of possible values.

creation of better performing IoT applications and to foster follow-up research on the topic. We then demonstrate the effectiveness of APEX experimentally by parametrizing two state-of-the-art multi-hop LPW protocols that have distinct design philosophies for a wide range of application requirements (§ 5). The first protocol, Crystal [28], utilizes concurrent transmissions to achieve high network throughput (and has a more deterministic behaviour), while the second protocol, the routing protocol for low-power and lossy networks (RPL) [54], adopts a classical routing approach (and is less deterministic). Our results indicate that APEX can return a parameter set that is optimal w.r.t. given application requirements using up to 10.6x less testbed trials compared to an exhaustive search of all possible parameter sets, thereby largely minimizing the necessary experimentation time. We also show the superiority of APEX over greedy algorithms and approaches based on reinforcement learning, which require, respectively, 4.5x and 3.25x more testbed trials than APEX to identify the optimal solution.

2 Motivation & Key Challenges

Our goal is to build a framework aiding the parametrization of LPW protocols based on the performance observed on a limited number of experimental trials[4]. Such framework should operate automatically (i.e., with minimal user intervention), and should *confidently* return the best parameter set for a certain protocol that satisfies given application requirements with as few testbed trials as possible (or within a given number of trials).

To exemplify the challenges in designing such a framework, consider an illustrative scenario where we are given a protocol such as Crystal [28][5], and are tasked to find the *best parameter set* (i.e., the optimal set of parameters that statistically provides the best performance) that allows to minimize the average energy consumption (E_c) of all nodes in the network while sustaining an average packet reception ratio (PRR) of at least 65%. For simplicity, we focus on two parameters of Crystal, as summarized in Tab. 1: the transmission (TX) power used by the radio to send packets (*tx_power*), and a node's maximum number of transmissions (*n_tx*) during a Glossy flood. Each of these two parameters can take one out of four possible values[6].

To design a framework that autonomously solves this task (i.e., that parametrizes Crystal such that energy consumption is minimized while satisfying the given constraint on PRR), we need to answer the following questions (Qi).

[4]In this paper, we explicitly focus on *testbed* experiments, as this is often the preferred way to test and debug the performance of LPW protocols using real hardware [7]. However, as discussed in § 6, the experimental data could also be collected through simulation or in a real-world deployment. APEX's functionality is, in fact, not bound to the use of LPW testbed facilities.

[5]Crystal is a well-known example of LPW protocol based on concurrent transmissions. Crystal works by exploiting Glossy [19] as a flooding primitive and by splitting floods into transmission and acknowledgment pairs. We refer the reader to the work by Istomin et al. [28] for further details.

[6]This results in a total of 16 parameter sets, which may suggest the feasibility of an exhaustive search. However, as we show later in this section, an exhaustive search where each parameter set is only tested *once* may not be sufficient to identify the optimal parameter set.

(Q1): What input does the framework require?

Firstly, the framework needs to receive input about the protocol under consideration. This includes knowledge of the parameters to be optimized, their range of possible values[7], how they are exposed within the protocol's code-base, and potential inter-dependencies. Secondly, the framework should know how to interface with the testbed where the protocol should be tested, in order to autonomously schedule runs that can shed light on the protocol performance for various parameter sets. In this context, any limitation about the testbed's availability (e.g., the maximum number of testbed trials that can be performed) should also be provided.

Next, the framework needs to be informed about the application requirement(s) for which the protocol shall be optimized. In the context of LPW systems, application requirements are typically translated into three key metrics related to communication performance: *reliability* (in terms of achieved PRR), *latency*, and *energy consumption* [20, 48][8].

Commonly, such requirements are expressed in the form:

$$\textit{maximize/minimize [metric(s)]} \quad \text{s.t.} \quad \textit{constraint(s) [metric].} \tag{1}$$

i.e., one defines an optimization goal specifying which metric(s) should be maximized or minimized, subject to (s.t.) given constraint(s). In our illustrative scenario, we have considered the following requirement:

$$\textit{minimize } E_c \quad \text{s.t.} \quad \textit{PRR} \geq 65\%. \tag{2}$$

(Q2): How to model protocol performance? Once provided with the necessary input, the framework shall (i) execute testbed trials to quantitatively assess protocol performance, and (ii) construct different *models* based on the available experimental observations. Such models are instrumental to predict protocol performance for unexplored parameter sets and guide the optimization process efficiently. For example, a model can predicts Crystal's performance (in terms of energy consumption, PRR, and/or latency) for specific values of the parameters *tx_power* and *n_tx*: this information is then used to identify promising parameter sets (i.e., parameter sets that are likely to provide the best performance) to be evaluated next, instead of relying on a random selection.

A model is built by fitting the *goal values* derived from previous experimental observations on a subset of parameter sets, where the goal values are numerical values reflecting the metric(s) that should be maximized or minimized. Such model embodies the *goal function*, i.e., it captures the performance of every parameter set in terms of the metric(s) specified in the defined optimization goal. Additional models derived by the framework represent the performance of every parameter set for each of the metric(s) given as constraint(s), i.e., for each of the *constraint metric(s)*. In our illustrative example, the framework would build two models capturing Crystal's energy consumption and PRR as a function of the employed parameter sets: the former embodies the goal function; the latter models protocol performance for the given constraint metric.

For instance, the sought framework can initially pick six parameter sets for Crystal[9], instruct the testbed to run one trial each, and collect the corresponding performance results in terms of PRR and energy consumption. Fig. 1(a) shows an example of the performance measured in these initial testbed trials: Crystal always sustains a PRR higher than 65%, and consumes between 182.0 and 209.5 J of energy. Based on these results, the framework can approximate the protocol's

[7]Expert users can also (but do not have to) offer recommendations on parameter values that are likely to be effective or ineffective.
[8]Note that these requirements are not independent: for example, improving reliability and latency typically entails a higher energy expenditure.
[9]We assume the user has no hint on good parameter values: the framework then picks 6 random sets (*tx_power,n_tx*): (0,4); (0,3); (-1,2); (-1;3); (-5,4); (-5,2).

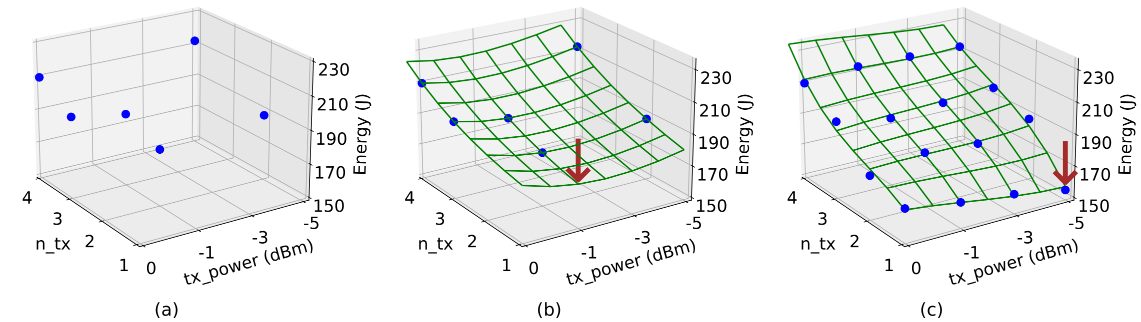


Fig. 1. Example of parameter exploration and modeling of protocol performance using Crystal. Energy consumption experimentally measured through testbed trials using six initial random parameter sets (a); polynomial regression model fitted to these initial experimental results (b); energy consumption and respective fit after an exhaustive search of all sixteen parameter sets (c). The brown arrow shows the best parameter set according to a model fitted after an exhaustive search of all parameter combinations.

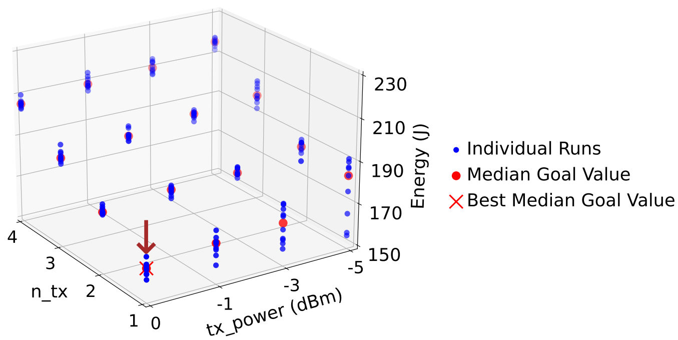


Fig. 2. Crystal's performance when executing ten experiments per parameter set. The best parameter set (brown arrow) is different from that observed in Fig. 1(c). Blue dots show the energy consumption observed in individual runs; red dots indicate the median energy consumption per parameter set (since energy consumption is the optimization goal, we refer to it as the median goal value). The red cross marks the lowest among all median goal values for configurations satisfying the specified constraint(s); we refer to this value as the *best median goal value*.

behaviour in terms of energy consumption (goal value) as a function of *n_tx* and *tx_power*, e.g., using polynomial regression, resulting in the model shown in Fig. 1(b), where the green surface represents the goal function[10].

In general, a key trade-off needs to be faced w.r.t. the development of such a model: the framework can adopt simple, pre-defined models (e.g., linear or polynomial regression) or more flexible, complex models (e.g., Gaussian processes). The trade-off here involves computational demands and information depth: complex models entail higher computational costs in favor of a greater information depth. Although the sought framework does not need to run on an LPW device, it is crucial to achieve a balance between model depth and computational efficiency, especially with a large number of parameters/values. Following the principle of Occam's razor, it is preferable to use simpler models with fewer assumptions, as they are often more flexible and better at capturing complex relationships. This ensures that the framework can adapt to the specific requirements of the optimization process.

[10]Note that the framework would also derive the model capturing the PRR as a function of *n_tx* and *tx_power*, but we omit this plot for brevity.

(Q3): How to select the next test-point?

Once fitted models are available, a key challenge is to make an informed selection of the next test-point (i.e., the parameter set whose performance should be measured next): this is essential to guide the optimization process effectively.

A straightforward approach might involve the exploitation of the current knowledge to select a test-point that is likely to align best with the application requirements. For example, following the model shown in Fig. 1(b), we could choose as next test-point the one marked by the brown arrow (i.e., *n_tx* = 1 and *tx_power* = -1 dBm). However, this approach may inadvertently hinder exploration of other potentially better regions within the parameter space[11]. In fact, exploring values with lower TX power would have more quickly led to a better parameter set, as shown in Fig. 1(c). The latter shows the model fitted after an exhaustive search of all sixteen parameter combinations[12]: the best parameter set, marked with the brown arrow, corresponds to *n_tx* = 1 and *tx_power* = -5 dBm. This example emphasizes the need to balance *exploration* and *exploitation* in surrogate-based optimization problems [10].

(Q4): How to handle real-world uncertainties?

Even though Fig. 1(c) suggests that *n_tx* = 1 and *tx_power* = -5 dBm (as marked with the brown arrow) is the best possible parameter set, this may not necessarily be the case: one needs in fact to also account for the *noisiness of experimental data.* More specifically, Fig. 1(c) is computed after testing each possible parameter combination *once.* Fig. 2, instead, shows the results of an exhaustive search where *10 testbed trials* were run for each parameter combination, with the red dots indicating the median energy consumption for each parameter set. When accounting for ten repetitions and using the median value as a reference, the best parameter set satisfying the application requirements is *n_tx* = 1 and *tx_power* = 0 dBm (marked with the brown arrow), which is very different from the one identified in Fig. 1(c) when testing each parameter set only once[13]. In fact, the blue dots in Fig. 2 highlight how the energy consumption measured across individual testbed runs varies by as much as 36.5 J for the same parameter set, thus underlining that *multiple experimental iterations are needed* to ensure robust parametrization.

In general, acknowledging and accounting for real-world uncertainties and variations is crucial to increase the statistical robustness of the results (i.e., tolerance to outliers): as outlined by Jacob et al. [30], ten repetitions are needed in this case to obtain results at the 60th percentile with 99% confidence[14]. Therefore, it is also imperative to consider ground truth data obtained by exhaustively testing all possible parameter sets *multiple times* when evaluating the performance of the sought framework. In the remainder of this paper, we will use the term *oracle*[15] to refer to such ground truth data (used exclusively to evaluate the sought framework's performance, and not available to the framework itself). Note that Fig. 2 essentially visualizes the knowledge of such *oracle*, and allows us to infer that – among all configurations that satisfy the specified constraint(s) – the best (i.e., lowest) median energy consumption is 164.38 J (marked with a red cross), which is obtained with *n_tx* = 1 and *tx_power* = 0 dBm. We can hence label the latter as the best parameter set (based on the oracle's knowledge), and 164.38 J as the best median goal value (again, based on the oracle's knowledge).

[11] Overexploitation may also trap the optimization process in local minima.
[12] In this simple example, an exhaustive search requires little effort, as there are only 16 possible sets of parameters. However, exhaustive search would be impractical when dealing with multiple parameters having several possible values each, as the number of experimental trials would rise exponentially.
[13] This may seem counter-intuitive, as the use of a higher *tx_power* (0 dBm vs. -5 dBm) should result in a higher energy consumption. In practice, at startup, Crystal continuously keeps a device's radio active in receiving mode (i.e., without duty cycling), awaiting synchronization with the initiator's first message. The use of a lower *tx_power* increases the chances to miss synchronization messages, thus resulting in a longer radio active time and a higher energy consumption.
[14] This entails a total of 160 experiments, underscoring the impracticality of exhaustive search even for a simple case with only 16 parameter sets.
[15] We assume that the oracle can be queried for any statistic, such as the parameter set with the lowest median goal value or the median goal value of a given configuration.

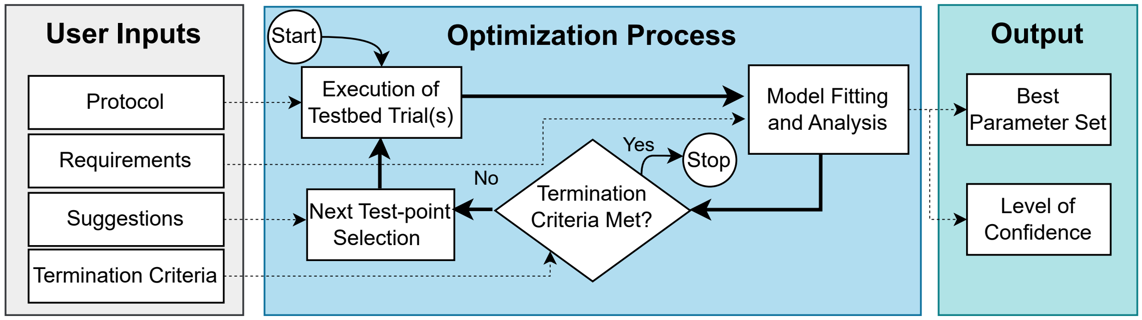


Fig. 3. High-level overview of the APEX framework.

(Q5): What and when should the framework return?

The selection of the next test-point based on the fitted models is an iterative process that can in principle run until an exhaustive search is completed. As this would imply a significant waste of time and resources (even more so when accounting for repetitions, as highlighted by **Q4**), the framework should establish a well-defined termination condition.

Determining when to terminate the optimization process is difficult: stopping the process prematurely may result in sub-optimal outcomes. In this regard, the ability to assess the *confidence* in the current solution would significantly help in defining the termination condition. However, measuring confidence is challenging in noisy scenarios with limited or no prior knowledge of the underlying protocol's behavior.

Once the decision to stop the optimization process is made, the framework should provide as output the best parameter set discovered during the exploration process. The framework should also give hints about the quality of the returned parameter set, e.g., in terms of robustness (confidence that the specified constraints are met) and optimality (confidence that no other parameter combination performs better).

3 APEX: Overview

This section gives an overview of APEX and its high-level building blocks, which are depicted in Fig. 3. As elaborated next, the design of each of these building blocks is guided by the need to address the challenges enumerated in § 2.

User inputs. To tackle the different aspects of **Q1**, the user needs to provide four types of inputs to APEX: the protocol under test, application requirements, suggestions for parameter selection (if any), and termination criteria. The first input refers to the specification of the protocol being tested (any protocol satisfying the assumptions stated in § 4.2), and the definition of the parameters to be optimized as well as their possible values. The user also needs to supply details about the experiments to be performed, such as the duration of individual testbed trials (to meaningfully capture performance), as well as traffic pattern, load, and network topology.

The second input are application requirements expressed as metrics related to communication performance. Each requirement entails one or more constraints and an optimization goal (i.e., which metric(s) to maximize or minimize), as described in **Q1**. Application requirements are referred to as AR: those used in this paper are listed in Tab. 3. The sample requirement presented in § 2 (listed as AR_1) is to minimize the average energy consumption (E_c) provided that a minimum PRR of 65% is sustained, with the considered PRR being the 50^{th} percentile PRR (median).

The third input are suggestions: users with expert knowledge about the protocol can define a favorable parameter space. APEX will account for these suggestions during the optimization.

The fourth input is the termination condition, which defines the criteria to stop the optimization process. Such criteria can be the maximum number of testbed trials available or the desired level of confidence in the returned parameter set.

Optimization process. APEX's parameter exploration starts with an *initial testing phase*, in which the framework *executes testbed trials* for a given number (n_{init}) of parameter sets to gain preliminary insights into the protocol's performance, as detailed in § 4.1. If user suggestions are provided in terms of favorable parameter sets, the initial tests prioritize these suggested sets. In the absence of user-suggested parameter sets, or if the number of suggested sets is insufficient, the initial parameter sets can be selected using methods such as Latin hypercube sampling, Sobol sequences[16], or random sampling.

After running the initial tests (n_{init} was set to 6 in the example shown in § 2), the framework proceeds to the *model fitting and analysis* step, which directly addresses **Q2** and plays a key role in understanding the underlying behavior of the protocol. In this phase, the framework fits models to the goal values and the values of the respective constraint metric(s), i.e., values of energy consumption and PRR in the case of AR_1. Specifically, as detailed in § 4.2, APEX leverages Gaussian processes (GPs) to efficiently model and analyze the limited data available, as they provide a flexible approach for capturing complex relationships without relying on a predefined functional form. Moreover, the use of non-parametric models such as GPs also allows APEX to address real-world uncertainties and thereby address **Q4**. Then, APEX identifies the *current-best parameter set*, i.e., the one with the highest or lowest goal value (depending on the context) that satisfies the specified constraint(s). Then, APEX assesses *how confidently* this parameter set satisfies the specified constraint(s), e.g., with 98% confidence, as well as the likelihood that this is the optimal solution. These two quantities express APEX's level of confidence in the current solution.

Afterwards, the framework checks whether the optimization process should continue or whether the given termination criteria have been met. APEX currently tackles **Q5** by allowing the user to specify as termination criteria the amount of available experimentation time on the testbed (from which the maximum number of testbed trials that can be executed is derived) and/or the level of confidence in the current solution. The latter is expressed either as confidence that the specified constraint(s) are met (robustness) or as the likelihood that no better parameter combination exists (optimality).

If the termination criteria are not met, the optimization process moves on to the *next test-point selection (NTS)*, which specifically addresses **Q3**, by determining the next parameter set to be tested in the next testbed trial. The algorithm selecting the next test-point uses the information gained from the fitted model and any user suggestion to select parameter sets that are likely to provide valuable insights into the protocol's behavior or that plausibly lead to better performance. § 4.3 will discuss in detail the NTS algorithms proposed for APEX.

Output. APEX's optimization process iterates until the termination criteria are met. Upon termination, APEX returns two outputs (fulfilling **Q5**). First, it provides the best parameter set found during the parameter exploration, i.e., the one that best aligns with the defined metrics and constraint(s). Second, APEX generates two confidence metrics: α quantifies the optimality of the returned solution (i.e., the confidence that no other parameter combination performs better than the returned one), whereas β expresses its robustness (i.e., the confidence that the given constraint(s) are met).

4 APEX: Key Components

Next, we delve into the APEX framework, exploring its key components. We aim to showcase how APEX tackles the challenging task of parameter exploration for LPW protocols.

[16]Latin hypercube sampling is a statistical sampling technique that ensures even coverage of the parameter space, while Sobol sequences provide a low-discrepancy sequence of points for a more uniform sampling distribution.

4.1 Execution of Testbed Trials

Testbed trials allow to quantify the performance of a protocol. A trial typically consists in running a given firmware implementing a network protocol on (a subset of) the testbed nodes, and in retrieving the relevant performance metrics upon the run's completion. To this end, one often needs to manually configure the tested firmware (so that it contains the intended parameter set), and to extract the sought performance metrics from raw testbed logs [3]. Note that a few modern tested facilities allow to conveniently compute certain metrics directly in HW (thereby minimizing the user's overhead and enabling a more objective performance evaluation [1, 37, 48]), or offer binary patching features to override certain parameter values without the need to regenerate the firmware [47].

The parameter set to be tested in a trial refers to a specific combination of values of the protocol parameters to be optimized. Let $D = \{\boldsymbol{x}_j \mid j = 1, 2, \ldots, N_p\}$ represent all the parameter sets considered, where $\boldsymbol{x}_j$ is the j-th parameter set, N_p is the number of total parameter sets. $\boldsymbol{x}_j = [v_1, v_2, .., v_b]$, where v_q is the value of the q-th parameter in the respective parameter set, and b is the total number of parameters considered. The performance metric returned by the testbed after the n-th testbed trial is denoted as $O_i(n, x_j)$, where x_j is the parameter set tested in the n-th testbed trial. $i \in \{1, 2, ..., N_m\}$ in O_i represents different performance metrics of interest, where N_m denotes the total number of performance metrics considered. For simplicity, we use $O(\cdot)$ to refer to the observation related to any performance metrics.

4.2 Model Fitting and Analysis

Before delving into the details of model fitting, we outline the key assumptions guiding our approach:

- We assume that parameter values can be represented in metric space, allowing for meaningful analysis;
- We assume that the underlying function representing the performance metric is continuous[17];
- Without loss of generality, we strictly consider the minimization problem (i.e., we minimize the goal value).

We now extend the mathematical notation to define our system model. Let $f_i : D \rightarrow \mathbb{R}$ represent an unknown continuous function from a compact metric space of parameter sets D to a set of real numbers $\mathbb{R}$. Also, $i \in \{1, 2, ..., N_m\}$ in f_i represents different performance metrics that are of interest in the respective optimization process. For instance, in AR_1, f_1 denotes the goal function (i.e., energy consumption) and f_2 represents the constraint metric (i.e., PRR). For simplicity, we consider a common function $f(\boldsymbol{x})$ to represent any f_i.

After the n-th testbed trial, we have n observations returned by the testbed. It is noteworthy that there could be multiple observations for a given parameter set. Then, the task of model fitting is to utilize the current observations and to derive a function $g(\boldsymbol{x})$ that approximates the unknown function $f(\boldsymbol{x})$ to guide the optimization process efficiently.

The models used to derive $g(\boldsymbol{x})$ can be categorized as parametric or non-parametric. Parametric models, such as linear regression and neural networks, assume specific forms and estimate fixed parameters. In contrast, non-parametric models, such as random forests and Gaussian processes (GPs), do not assume a fixed functional form like parametric models do. Instead, they directly learn from data, providing flexibility. The choice between parametric and non-parametric models depends on the nature of the problem, data availability, computational constraints, and flexibility.

[17] In most cases, performance metrics align with this assumption. Although some parameters may be defined by discrete values (e.g., transmit power levels of 0 dBm, -1 dBm, -3 dBm, and -5 dBm on specific platforms), their relationship with the performance metric can often be modeled as a continuous function. This is because the underlying behavior of the performance metric generally exhibits a smooth and predictable trend with respect to these parameter changes, allowing for meaningful modeling despite the discrete nature of the parameter values.

GPs are particularly well suited to our context [10, 23]. Because each wireless trial is costly, the model must learn quickly from only a few observations. A GP meets this need: every Bayesian update turns a handful of measurements into calibrated estimates of the mean and the variance [10]. The variance naturally grows in regions we have not tested, steering APEX toward informative trials and supporting the confidence in the termination criteria for optimality (α) described later on. Although GPs are computationally demanding, this is not a major concern for APEX, because all model updates run on a server rather than on resource-constrained devices, and the extra computation is negligible compared with the several minutes required for each testbed run. For these reasons, a GP is the natural surrogate for APEX's optimization.

Gaussian Processes (GPs). We model $g(\boldsymbol{x})$ as a stochastic process, so that for any finite set of inputs $\boldsymbol{x}$, the corresponding function values $g(\boldsymbol{x})$ follow a joint multivariate Gaussian distribution. Hence, for any parameter set $\boldsymbol{x}$, the possible corresponding function values $g(\boldsymbol{x})$ are represented as a Gaussian distribution. Thus, mean and variance can characterize the values for each parameter set $\boldsymbol{x}$.

A GP is fully characterized by its mean function $\mu(\boldsymbol{x})$ and covariance function (kernel function). For a given parameter set $\boldsymbol{x}$, mean and variance are calculated as in [10]:

$$\mu(\boldsymbol{x}) = \boldsymbol{k}^T \boldsymbol{K}^{-1} \boldsymbol{F^o} \quad ; \quad \sigma^2(\boldsymbol{x}) = c(\boldsymbol{x}, \boldsymbol{x}) - \boldsymbol{k}^T \boldsymbol{K}^{-1} \boldsymbol{k}, \tag{3}$$

where $\boldsymbol{k}$ is the vector of covariances between $\boldsymbol{x}$ and the observed parameter sets, $\boldsymbol{K}$ is the covariance (kernel) matrix of the observed parameter sets, $\boldsymbol{F^o}$ is the vector of observed values of the respective observed parameter sets, and $c(\cdot)$ is the covariance (kernel) function determining the similarity between two inputs. Common kernel functions include the radial basis function (RBF) kernel and the Matern kernel, each offering various degrees of smoothness and flexibility [9, 10].

As the GP model treats $g(\boldsymbol{x})$ as a stochastic process, once the GP is fitted to the given observations, it provides a range of possible functions that could approximate $f(\boldsymbol{x})$. This implies that there are multiple potential functions for $g(\boldsymbol{x})$. The mean of these functions is denoted by $\mu(\boldsymbol{x})$, which is considered the best estimate among these potential functions. Therefore, $\mu(\boldsymbol{x})$ is commonly regarded as the most representative approximation of $f(\boldsymbol{x})$, making it effectively $g(\boldsymbol{x})$.

In addition to $\mu(\boldsymbol{x})$ approximating the behavior of the $f(\boldsymbol{x})$, GPs also provide a tool to calculate the associated uncertainty. This aids in guiding the optimization process, specifically in selecting the next test-point (discussed in § 4.3) and also in approximating the achieved optimality (discussed later in this section). The uncertainty associated with a given parameter set $\boldsymbol{x}$ is characterized by the variance $\sigma^2(\boldsymbol{x})$. Because the GP couples all parameter sets through its kernel (covariance) matrix, an uncertainty estimate obtained at one configuration propagates to others according to their covariance. Hence, the predictive variance $\sigma^2(\boldsymbol{x}')$ at an unseen configuration $\boldsymbol{x}'$ is larger when it is correlated with noisy observations, compared to the variance one would obtain if the nearby points had low noise. This property allows APEX to anticipate real-world variability across the entire parameter space and to steer the optimization toward configurations that are both promising and robust.

Fig. 4 (top) shows an example of GP fit after $n = 6$ testbed trials. Here, $\mu(\boldsymbol{x})$ represent the $g(\boldsymbol{x})$. The uncertainty region around the $\mu(\boldsymbol{x})$, which is depicted as the shaded area around $\mu(\boldsymbol{x})$, is derived from $\sigma^2(\boldsymbol{x})$. After executing a new testbed trial ($n = 7$), we obtain a new observation at $x = 6$, which changes the overall structure of $g(\boldsymbol{x})$ and its related uncertainty, as shown in Fig. 4 (bottom). The uncertainty near the new observation will decrease, while the uncertainty related to points farther away may remain unchanged or increase. This depends on how the uncertainty region is defined, as discussed later in this section under "*optimality*". In Fig. 4, we can also see that the observation is not always

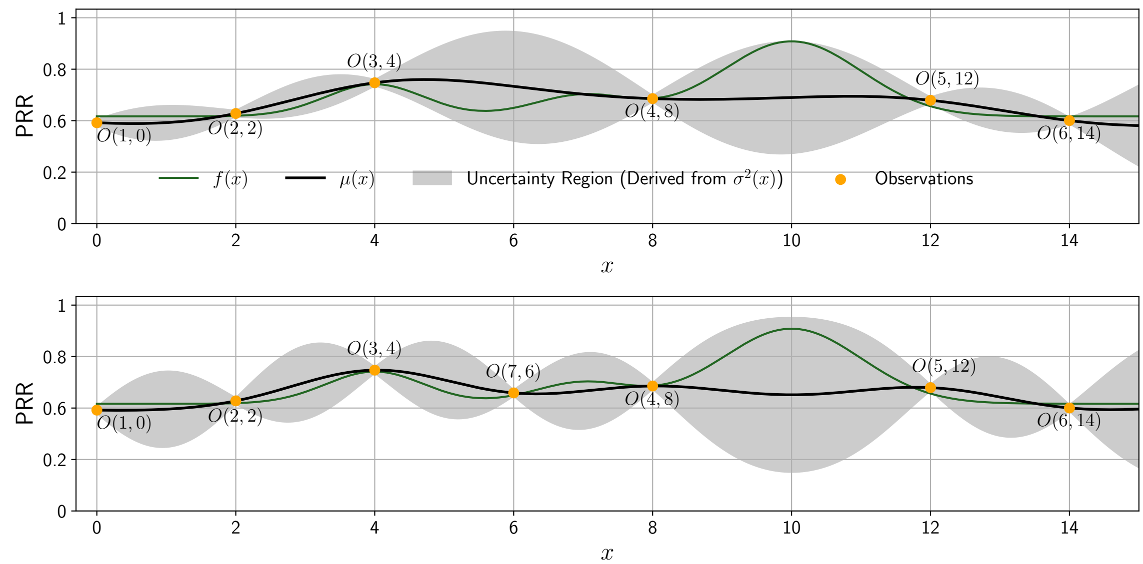


Fig. 4. Top: GP fit and regression after six testbed trials ($n = 6$) for a single parameter ($b = 1$), where x represents the values of that parameter. Bottom: updated GP fit and regression after a new trial ($n = 7$).

aligned with the underlying unknown function. This is because of *noise* in real-world experimental data, which arises from various factors (e.g., changes in environmental conditions, HW-related fluctuations, and RF interference). Hence, the observation of the performance in the n-th testbed trial can be denoted as:

$$O(n, \boldsymbol{x}) = f(\boldsymbol{x}) \pm \epsilon(n, \boldsymbol{x}), \tag{4}$$

where $\epsilon(n, \boldsymbol{x})$ is the *noise* at the n-th testbed trial for the given parameter set $\boldsymbol{x}$.

It is worth noting that we employ the GP in a simple Bayesian Optimisation (BO) loop, as it is more suitable to the given problem [10]. Although GPs have also been used in reinforcement learning (RL) as value or policy function approximators, the feedback available to APEX is fundamentally different. Each testbed run yields one or more terminal performance metrics (i.e., values collected only at the end of the run) with no intermediate state transitions and no dense reward signal. Therefore, in APEX, we embed the GP in a BO loop: the GP provides a global surrogate of the black box objective, and its posterior variance supplies a balance between exploration and exploitation. We will show in § 5 how APEX outperforms a GP-based RL baseline[18].

Current-best parameter set ($\boldsymbol{x}_n^+$). Once the fitted model is available, APEX moves to its analysis. The analysis begins by identifying the current-best parameter set based on the available results. The filtered parameter sets that satisfy the constraint(s) after the n-th testbed trial are $D_n \subseteq D$. From these sets, the parameter set with the best goal value, specifically the best *median* goal value[19], is selected as the current-best parameter set. The current-best parameter set after the n-th testbed trial is denoted as $\boldsymbol{x}_n^+$. Notably, within the framework's operation, the best parameter set that is returned as output is updated only if the new best set has an equal or greater number of test results compared to the

[18] Although BO can also be viewed as a model-based reinforcement learning method, we use the term RL here strictly for the classical formulation with explicit state transitions and dense step-wise rewards.
[19] The median is chosen as a robust measure against outliers and is used consistently in APEX to assess performance across testbed trials.

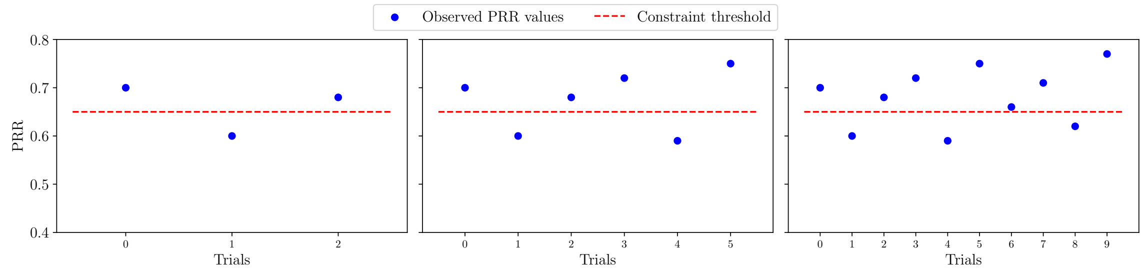


Fig. 5. Illustrative example of how the lower bound of the confidence metric β is computed for a given parameter set. Each subplot shows a set of PRR values obtained from multiple testbed trials using the same parameter configuration. The total number of trials (N) and the number of results satisfying the constraint (l) are used to calculate a lower bound on the non-parametric confidence β, as defined in Eq. (5). In this example, the constraint requires the median PRR to be no less than 0.65, corresponding to $p = 0.5$ in the confidence formulation. This lower bound is conservatively taken as the estimated β in our framework.

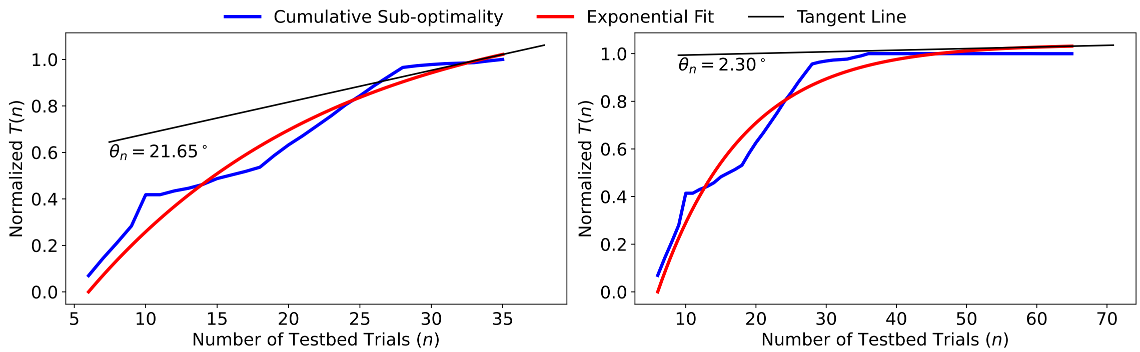
current one, thereby preventing frequent changes and decisions based on outliers. It is also important to distinguish between the best parameter set actually returned by APEX and the best parameter set that should be returned by APEX according to the oracle's knowledge. While $\boldsymbol{x}_n^+$ reflects the current-best parameter set identified by APEX up to trial n, it may or may not be the best parameter set according to the oracle. The latter is defined as the one that yields the *best median goal value*, as shown in Fig. 2, which corresponds to the lowest observed median goal value across all parameter sets satisfying the specified constraint(s). Also noteworthy is that the best parameter set according to the oracle is used solely for evaluating the performance of the framework, while APEX itself remains unaware of it during operation.

Robustness (β). Then, APEX will assess how confidently the parameter set satisfies the constraint(s). For example, according to AR_1 (see Tab. 3) the median of the PRR should not be smaller than 65%. The confidence metric β quantifies how confident APEX is that the median of PRR is higher than 65%. The robustness of a given parameter set $\boldsymbol{x}$ after the n-th testbed trial $\beta_n(\boldsymbol{x})$ can be represented through the following inequality using the non-parametric approach [16, 30]:

$$\beta_n(\boldsymbol{x}) \geq \mathbb{P}\left(s_l \geq P_p\right) = \sum_{k=0}^{l-1} \begin{pmatrix} N \\ k \end{pmatrix} p^k (1-p)^{N-k}, \tag{5}$$

where N is the total number of available test results for the parameter set $\boldsymbol{x}$ after the n-th testbed trial, l is the number of test results that satisfies the constraint, s_l is the nearest test result to the constraint that satisfies the constraint and P_p is the p-th percentile of the distribution where $p \in (0, 1)$ is the percentile. In AR_1, $p = 0.5$ represents the 50-th percentile (median). The expression $\mathbb{P}\left(s_l \geq P_p\right)$ provides the confidence that the p-th percentile of the respective constraint's metric is above the nearest test result satisfying the constraint for $\boldsymbol{x}$. Thus, we can conservatively take $\beta_n(\boldsymbol{x})$ as $\mathbb{P}\left(s_l \geq P_p\right)$ [20]. To clarify how the robustness metric β is computed, Fig. 5 presents a step-by-step illustrative example using synthetic data. We consider the AR_1 constraint on PRR (requiring the median PRR to be ≥ 0.65), which corresponds to the 50th percentile of the PRR distribution and results in setting $p = 0.5$ in the confidence formulation. The figure demonstrates how the total number of trials (N) and the number of observations satisfying the constraint (l) are used to compute a lower bound on the non-parametric confidence β for a given parameter configuration. In our framework, we conservatively take this lower bound as the estimated β. Starting with three trials in Fig. 5(a), we incrementally add more results in Figs. 5(b) and 5(c), showing how N, l, and the corresponding β evolve at each stage.

[20]With multiple constraints, the lowest confidence value is chosen as β.



(a) $n = 36$ (b) $n = 66$

Fig. 6. Illustration of how APEX derives optimality. We compute an exponential fit of the cumulative sub-optimality $T(n)$, and derive the angle θ_n of the fit's tangent after the n-th testbed trial. A higher θ_n (e.g., for n=36) indicates a low optimality. After additional testbed trials (e.g., n=66), a low θ_n indicates closeness to optimality.

Optimality (α). APEX computes a second confidence metric α quantifying the level of confidence in the optimality of $\boldsymbol{x}_n^+$. While this is hard in absence of prior knowledge, we can leverage the insights gained from the GPs to derive α. For that, we initially calculate the GP's lower confidence bound (LCB) for the parameter sets[21]. The LCB at a given parameter set $\boldsymbol{x}$ after the n-th testbed trial is given by:

$$\mathrm{LCB}_n(\boldsymbol{x}) = \mu_n(\boldsymbol{x}) - \kappa_n \sigma_n(\boldsymbol{x}), \tag{6}$$

where $\kappa_n > 0$ is a calibration parameter, $\mu_n(\boldsymbol{x})$ and $\sigma_n(\boldsymbol{x})$ represent the mean and the standard deviation at $\boldsymbol{x}$ after the n-th testbed trial. If the bounds are well-calibrated, they accurately reflect the uncertainty in the estimation of the minimum. For a finite parameter space D, a well-calibrated κ for the n-th testbed trial is given as $\kappa_n = \sqrt{2\log\left(|D|n^2\pi^2/6\delta\right)}$ [51], where $\delta \in (0, 1)$ represents the strictness of the asymptotic regret bound. The lowest LCB after the n-th testbed trial is $\mathrm{LCB}_n^*(\boldsymbol{x}) = \min_{\boldsymbol{x}\in D_n} \mathrm{LCB}_n(\boldsymbol{x})$. Given the bounds are well-calibrated, the instant suboptimality can be approximated as $\tau(n) = f(\boldsymbol{x}_n^+) - \mathrm{LCB}_n^*(\boldsymbol{x})$. To capture the relative trend, we calculate the cumulative suboptimality as:

$$T(n) = \sum_{t=1}^{n} \tau(t). \tag{7}$$

Analyzing $T(\cdot)$ trends aids in understanding optimality progression. Diminishing increments imply nearing optimality; constant or growing increments signal divergence[22]. Yet, quantifying this trend systematically is complex. We fit an exponential curve to the normalized trend, facilitating derivative calculation and tangent line angle (θ_n) determination at n. Fig. 6 presents an illustrative example of this fit.

A 45° angle implies a constant increase in cumulative suboptimality, considered as the worst scenario, thus capped at 45° with $\theta_n^{'} = \min(45^\circ, \theta_n)$. Then, the optimality confidence metric after the n-th testbed trial $\alpha(n)$ is calculated as:

$$\alpha(n) = 100(1 - \theta_n^{'}/45). \tag{8}$$

[21] In Fig. 4, the LCB is the lower limit of the shaded region.

[22] The phenomenon of diminishing returns implying progress toward optimality generally works even in cases with multiple local minima. However, it may not be effective in scenarios with a sharp loss landscape containing multiple local minima. APEX's use of simpler functions typically promotes a smoother loss landscape, ensuring that the metric remains effective in most practical cases.

Here, $\theta_n^{'}$ is in degrees. The 45° angle suggests a high likelihood of being far from the optimal, resulting in a confidence level of 0%. Conversely, angles closer to 0° indicate proximity to the optimal, suggesting optimality close to 100%.

4.3 Next Test-point Selection (NTS)

The NTS is crucial because our optimization problem relies on surrogate models [43]. As highlighted by Brochu et al. [10], the core challenge in such problems is balancing exploration and exploitation – exactly the task of the NTS algorithm.

The next parameter set that is chosen after the n-th testbed trial is $x_{n+1} \in D$. Here, we exclusively employ $f(\boldsymbol{x})$ to represent the goal function. Thus, the goal is to find the $\boldsymbol{x}^*$ (best parameter set) such that $\boldsymbol{x}^* = \operatorname{argmin}_{\boldsymbol{x} \in D_c} f(\boldsymbol{x})$, where D_c is the actual parameter set that satisfies the constraint(s) to which the optimization goal is subject (see Eq. 1).

We explore two specific NTS algorithms incorporating the insights from GPs, and evaluate their performance in § 5.

GP-LCB. The first algorithm focuses on minimizing the lower confidence bound of the estimated model, providing a balanced approach for identifying promising regions likely to contain the optimal solution. The LCB of the parameter sets can be calculated as in Eq. (6), where κ_n helps to balance the exploration and exploitation. Then, in the GP-LCB approach, we choose the next test-point as $x_{n+1} = \operatorname{argmin}_{\boldsymbol{x} \in D_n} \mathrm{LCB}_n(\boldsymbol{x})$.

Expected Improvement (EI). Compared to GP-LCB, which focuses solely on the LCB of the estimated model, this NTS algorithm considers the probability of improvement and the uncertainty over the current-best observation. The EI at $\boldsymbol{x}$ after the n-th testbed trial is calculated as in [10]:

$$\mathrm{EI}_n(\boldsymbol{x}) = \begin{cases} \left(f(\boldsymbol{x}_n^+) - \mu_n(\boldsymbol{x})\right)\Phi(Z_n) + \sigma_n(\boldsymbol{x})\phi(Z_n), & \text{if } \sigma_n(\boldsymbol{x}) > 0, \\ 0, & \text{if } \sigma_n(\boldsymbol{x}) = 0 \end{cases} \qquad Z_n = \frac{f(\boldsymbol{x}_n^+) - \mu_n(\boldsymbol{x})}{\sigma_n(\boldsymbol{x})}. \quad (9)$$

where $f(\boldsymbol{x}_n^+)$ is the current-best observation of the goal value and Z_n is the standard improvement. $\Phi(Z_n)$ and $\phi(Z_n)$ are respectively the cumulative distribution function and probability density function of a standard Gaussian distribution. In Eq. (9), the term $\left(f(\boldsymbol{x}_n^+) - \mu_n(\boldsymbol{x})\right)\Phi(Z_n)$ quantifies the potential improvement over $f(\boldsymbol{x}_n^+)$ by leveraging the mean prediction of the model and the probability of improvement. Conversely, the term $\sigma_n(\boldsymbol{x})\phi(Z_n)$ captures the uncertainty in the prediction. Together, these components help balancing exploration and exploitation. The next test-point is selected as the one which maximizes the EI, i.e., $x_{n+1} = \operatorname{argmax}_{\boldsymbol{x} \in D_n} \mathrm{EI}_n(\boldsymbol{x})$.

Tackling outliers. Although GP-LCB and EI effectively handle uncertainty, they are still prone to local minima traps due to extreme outliers around global/local minima and challenges from noisy measurements in the constraint's metric(s). The likelihood of being trapped in a local minimum can be quantified by assessing the uncertainty associated with the selected next parameter set. Low uncertainty implies high confidence, indicating limited potential for new information [33]. Having the uncertainty falling below a predefined lower threshold signals the possibility of being stuck in a local minimum. In the EI approach, this likelihood can be quantified by EI itself; in GP-LCB, instead, with the coefficient of variation (CV). We hence set the thresholds EI_{min} and CV_{min} based on the EI and CV's maximal values[23].

To address the issue of extreme outliers around global/local minima, we discard the parameter sets with the highest statistical significance (i.e., with the highest number of test results). We then search for the parameter set that is more likely to offer substantial improvements according to the respective NTS approach. To address the challenge posed by

[23]Note that APEX allows expert users to override these values: this can be helpful when measurements are known to have a high/low uncertainty.

noise in the constraint's metric(s), we can use the GP model fitted to such metric(s). To generalize, we treat constraint(s) as maximal allowable values. Initially, we calculate the LCB values of the respective constraint's metric, $\text{LCB}_n^c(\boldsymbol{x})$.

The LCB_n^c values help to identify parameter sets that are very likely to satisfy the constraint. However, our goal is not only to satisfy the constraint, but to improve the goal value as well. The respective improvement in the goal value can be calculated as $I_n(\boldsymbol{x}) = f(\boldsymbol{x}_n^+) - \mu_n(\boldsymbol{x})$. Then, the combined metric $\Delta_n(x)$ is calculated as:

$$\Delta_n(x) = \frac{\text{LCB}_n^c(\boldsymbol{x})}{f_c^+} - \frac{I_n(\boldsymbol{x})}{f(\boldsymbol{x}_n^+)}, \tag{10}$$

where f_c^+ is the current-best observation of the constraint metric [24]. Both terms are divided by their respective current-best observation to reduce the biases. Then, the next parameter is selected as $\boldsymbol{x}_{n+1} = \text{argmin}_{\boldsymbol{x} \in D_n'} \Delta_n(x)$, where the complementary set D_n' represents the parameter sets that currently do not satisfy the constraint(s)[25], i.e., $D_n' = i \in D : i \notin D_n$. This approach efficiently explores parameter sets that are likely to satisfy the constraint(s) and provide improvements in the goal value.

APEX applies the above techniques to tackle outliers in the goal function and constraint metric(s) *consecutively in each run*, until the optimization process escapes the trap. It is important to note that, while we leverage Gaussian processes and well-established acquisition functions, we have tailored these techniques to address the specific challenges of LPW parametrization. In particular, our approach is designed to handle the presence of outliers in the goal function and constraint metric(s) effectively.

5 Evaluation

We implement an open-source[3] prototype of the APEX framework in Python, with each of the building blocks depicted in Fig. 3 having their own script. This modular approach allows users to easily select or replace various components. User input (such as protocol and parameters to be optimized, application requirements, and termination criteria) is incorporated into the APEX's Python scripts through YAML files.

We then evaluate APEX empirically, analyzing the performance of its optimization process (i.e., the number of testbed trials required to converge to what the oracle identifies as the best parameter set – § 5.2), the likelihood of returning the parameter set assessed to be the best by the oracle after a fixed number of testbed trials (§ 5.3), and how well α reflects this likelihood (§ 5.4). We further show APEX in action when exploring a large parameter space (§ 5.5), and when parametrizing a protocol using different network topologies and in environments rich of RF interference (§ 5.6).

5.1 Protocols, Metrics, and Methodology

For our evaluation, we use D-Cube [24], a public testbed providing a convenient testing environment with a total of 48 LPW devices deployed over two floors in a university building. We focus on a data collection application run over an IEEE 802.15.4 mesh network consisting of all devices available in D-Cube. One node acts as destination, whereas several nodes act as sources of data: their identity depends on the chosen *testbed layout.* We run most of our experiments using D-Cube's layout number 1 for data collection[26], but also show APEX in action on a different testbed layout and with different environmental conditions in § 5.6 to demonstrate its broader applicability.

[24] f_c^+ is bounded by the respective constraint's threshold value. For example, if the constraint is specified as "PRR ≥ 65%", the maximum value of f_c^+ is capped at 65, as only parameter sets that currently fail to meet the constraint are considered.
[25] If there are multiple constraints, the one with minimum $\Delta_n(x)$ is chosen.
[26] Details about the various layout in D-Cube is available at https://iti-testbed.tugraz.at/layout.

Protocols and parameters. We consider two well-known LPW protocols: Crystal [28] and RPL [54]. Their different nature (Crystal employs concurrent transmissions, whereas RPL adopts a classical routing approach) allows us to assess the versatility of APEX. For Crystal, we use the implementation based on Baloo [29] running on `TelosB` devices; for RPL, we employ Contiki-NG's RPL-Lite [15] running on `nRF52840-DK` devices. Each of the source nodes transmits packets periodically every 1 and 5 seconds for Crystal and RPL, respectively. This periodicity reflects the one found in common sensor network applications [34].

Tab. 1 shows the explored parameters for each of the two protocols. For Crystal, we consider the two parameters described in § 2: tx_power (transmission power) and n_tx (maximum number of transmissions). For RPL, we study the max_link_metric, a parameter used to reject parents with a higher link metric than a given ETX value; the $DIO\_interval$, which is the interval that controls how frequently DODAG Information Object (DIO) messages are sent to disseminate routing and topology information in the network; and the $rank\_threshold$, a parameter used to avoid network instability by only switching to parents having a link quality higher than a given ETX value.

Methodology. When evaluating our framework against other baseline approaches, we need to consider that any approach has the potential to select the best parameter set *by chance* after the first few testbed trials. However, such an outcome does not necessarily imply the inherent superiority of the approach. Understanding the true performance of an approach requires a statistically-significant assessment over *numerous iterations*. Therefore, in our evaluation, we repeat the optimization process 1000 times. However, performing 1000 iterations through testbed experiments is utterly impractical, as it would require several years[27]. Hence, we collect *ground truth data* on the D-Cube testbed by exhaustively testing all N_p parameter sets N_r times[28]; each individual trial lasts γ minutes, as summarized in Tab. 2.

This ground truth data will serve two key purposes. On the one hand, it represents the knowledge of an *oracle*, which is used to compute the best parameter set for certain application requirements (i.e., the one that should be returned by APEX once the optimization process terminates). On the other hand, the ground truth data is used to *emulate* 1000 iterations of APEX's optimization process as follows. At run time, we provide APEX with the results of one testbed trial at a time – exactly in the same way as if the framework would receive this data from the testbed API. Because APEX's optimization loop samples from these measured outcomes one at a time, the optimizer must contend with the same stochastic conditions it would face during an online run, giving a realistic measure of robustness. The optimization loop, therefore, preserves the necessity of balancing exploration and exploitation with the real-world variability that APEX must handle in practice. In each iteration, when the optimization process queries a parameter set, it randomly draws one of the N_r testbed trials recorded for that set and returns the corresponding result. Each result can only be used once, i.e., if all available results for a specific parameter set are used up, that parameter set is considered exhausted and is no longer a candidate for selection. In such cases, the process will move on to the subsequent parameter set as per the respective NTS criteria[29]. This choice ensures that we can limit the number of experiments to be performed for each parameter set.

[27]For example, a *single iteration* to derive the best parameter set for RPL would require 65 hours of testbed experiments. This estimate assumes 36 parameter sets, 6 repetitions per set, and 18 minute execution time per trial, as summarized in Tab. 2. It does not include queueing delays or the overhead of preparing and flashing each experiment on a shared platform; accounting for these factors, a full run would extend over several weeks.

[28]Note that the N_r independent repetitions for every parameter set are taken at different times of the day/week, and may hence experience variable environmental conditions due to uncontrollable interference and presence of people in the building. Fig. 2 already visualizes this effect for Crystal with $N_r = 10$. The vertical spread of the blue dots varies as much as 36.5 J for the same parameter set, confirming that the collected dataset already captures significant real-world variability.

[29]Note that if the optimization process requests a parameter for which no recorded result is available (e.g., $n\_tx = 1.5$, due to the use of a continuous domain – which may occur when employing greedy approaches), the optimization process will return the result of the closest available parameter set in the parameter space.

Parameter	Value	Parameter	Value	Parameter	Value
δ	0.1	EI_{min}	$EI_{max}/10$	η	0.5
Kernel for GP	RBF	CV_{min}	$CV_{max}/10$	n_{init}	6
RBF length scale	1	N_p (Crystal)	16	γ (Crystal)	10 minutes
N_r	6	N_p (RPL)	36	γ (RPL)	18 minutes
ϵ	0.05				

Table 2. Parameter values used in our evaluation.

Baselines. We compare APEX's performance against seven approaches. Specifically, we choose three greedy algorithms (for which the same mathematical notations introduced in § 4 are used to describe them), three approaches based on reinforcement learning (RL), and an SVM-based approach for the NTS. These seven baselines are summarized as follows (note that the fitted function by the chosen model after the n-th testbed trial is $g_n : D \rightarrow \mathbb{R}$):

- *Greedy for Exploitation (GEL)* focuses on exploitation. After the n-th testbed trial, it selects the next parameter set as $x_{n+1} = \text{argmin}_{\boldsymbol{x} \in D_n} g_n(\boldsymbol{x})$ or a random one if $D_n = \emptyset$.
- *Greedy for exploration (GER)* always explores the parameter space evenly, i.e., for each N_p testbed trials, all the parameters sets will be tested once.
- *Greedy for Uncertainty (GUC)* balances exploration and exploitation by identifying uncertain regions in the parameter space. After the n-th trial, it computes uncertainty metrics $\zeta_n(\boldsymbol{x})$ for $x \in D_n$. Considering only single-hop neighbors, it assigns -2 to $\zeta_n(\boldsymbol{x})$ for each test result at $\boldsymbol{x}$ and -1 for each test result among the neighbors. It selects a subset D_u with maximum uncertainty, where $D_u = \boldsymbol{x} \in D_n : \zeta_n(\boldsymbol{x}) = \max_{\boldsymbol{y} \in D_n} \zeta_n(\boldsymbol{y})$. From D_u, the next chosen parameter set is $x_{n+1} = \text{argmin}_{\boldsymbol{x} \in D_u} g_n(\boldsymbol{x})$. If $D_u = \emptyset$, a random parameter set is chosen.
- *Reinforcement Learning (RL-Step)* is inspired by the approach presented in [31]. In this method, the agent operates within a Q-learning framework, but with a more constrained approach where it can adjust only one parameter at a time by a single step. The states represent the current parameter combination, and actions involve incrementing, decrementing, or retaining the value of a single parameter. This approach models a more systematic and localized adaptation of the parameter space, with an ϵ-greedy policy used for action selection to balance exploration and exploitation. A penalty is applied for unsatisfied constraints, encouraging the agent to avoid infeasible configurations.
- *Reinforcement Learning (RL-Any)* builds upon *RL-Step* by relaxing the restriction of adjusting only one parameter at a time. In RL-Any, the agent can adjust multiple parameters simultaneously, making it a more flexible and generalized version than *RL-Step*. This flexibility is designed to align with other approaches, where the agent can move from any parameter set to any other parameter set, enabling broader exploration of the parameter space. As in *RL-Step*, states correspond to the current parameter combination, and actions represent transitions to new parameter configurations. The Q-table maps state-action pairs to expected performance and dynamically expands to accommodate transitions to any parameter combination. This extended flexibility enables the agent to explore the parameter space more efficiently. An ϵ-greedy policy, along with a penalty for unsatisfied constraints, is used in *RL-Any*, similar to *RL-Step*.
- *Reinforcement Learning with GP (RL-GP)* combines reinforcement learning with Gaussian process regression to guide parameter selection. For each visited parameter configuration (state), a separate GP model is trained to estimate the performance of potential next configurations (actions) from that state. This allows the agent to generalize from previously-observed transitions, and to make informed decisions without exhaustively evaluating all actions. From each state, the corresponding GP is used to select the next action using the GP-Lower Confidence Bound (GP-LCB) criterion, aiming to minimize the GP-LCB.

- *Support Vector Regression (SVR)* leverages Support Vector Machines (SVMs), a class of supervised learning models known for their effectiveness in high-dimensional spaces and robustness to overfitting. SVR is used to approximate the goal value across the parameter space. It is trained on previously-observed results to predict the performance of all untested configurations. At each iteration, the configuration with the lowest predicted goal value is selected for testing, guiding the exploration process based on regression estimates.

Note that GEL and GUC use ordinary *least squares regression* for model fitting due to its simplicity, ease of interpretation, and proven suitability for surrogate-based optimization problems [4]. In contrast, the reinforcement learning approaches use different mechanisms: RL-Step and RL-Any rely on a Q-table to learn state-action values, RL-GP leverages Gaussian processes to model transitions locally from each state, whereas the SVR baseline (BL-SVM) employs support vector regression to globally approximate the goal value across the parameter space.

For the confidence metric approximating optimality, we employ two baseline approaches. The first tracks variance in the best performance after each trial, quantifying confidence:

$$\alpha_{B_1}(n) = \left(1 - \frac{f(\boldsymbol{x}_n^+) - f(\boldsymbol{x}_{n-1}^+)}{R_g(n))}\right) \cdot 100, \tag{11}$$

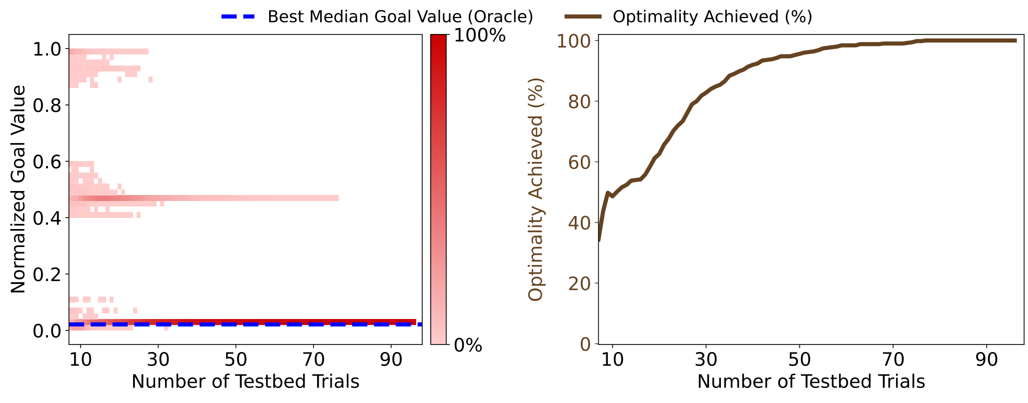
where $R_g(n)$ is the current range of the goal values. It calculates the subsequent change in the goal function and normalizes it to the current observed range. This approach assumes that the lesser the variance, the closer the solution is to optimal. However, it is not sensitive to changes in the parameter space (D). For instance, a minimal change in the goal value may obscure a significant parameter shift, potentially indicating that the solution is trapped in a local minimum. To address this, we adjust it as follows:

$$\alpha_{B_2}(n) = \left[\eta\alpha_{B_1}(n) + (1-\eta)\left(1 - \frac{d(\boldsymbol{x}_n^+ - \boldsymbol{x}_{n-1}^+)}{d_{\max}}\right)\right] \cdot 100, \tag{12}$$

where $d(\cdot)$ calculates the normalized Euclidean distance between two points, d_{max} is the maximum Euclidean distance of the parameter space, and $\eta \in (0, 1)$ is the balancing factor between both domains.

Evaluation metrics. Fig. 7 illustrates how we evaluate APEX's performance, and helps understanding how we derive the evaluation metrics. In Fig. 7(a), we show a heatmap of the median goal values (normalized energy consumption) obtained over 1000 independent runs of the optimization process when using GEL to parametrize Crystal for application requirement "*Minimize* E_c | *PRR* $\geq$ *96%*" (later referred to as AR_3, see Tab. 3). Such a heatmap can be used to visualize the points in which the optimization process could be trapped, and to illustrate how often this occurs before converging to the *best median goal value* (shown as blue dashed line). The latter is computed based on the oracle's knowledge. Fig. 7(b) complements this view by plotting the percentage of runs that successfully identify what the oracle deems as the best parameter set – referred to as *optimality achieved* (%) – as a function of the number of testbed trials. For example, we can infer from Fig. 7(b) that, in 80% of the cases (i.e., in 800 out of the 1000 runs of the optimization process), only 27 testbed trials were needed to converge to what the oracle defines as the best parameter set.

For clarity of interpretation, we define our evaluation metrics based on the combined version of the two plots in Fig. 7 (shown in Fig. 8). The green dashed lines depict the evaluation metrics. The first evaluation metric (EM_1) is defined as the number of testbed trials needed to achieve an *optimality of 99%*. The second evaluation metric (EM_2) is defined as the *optimality* achieved after performing N_p trials. Similarly, the third evaluation metric (EM_3) is defined as the *optimality achieved* after performing $2 \cdot N_p$ trials.



(a) (b)

Fig. 7. Evaluation process illustrated when using GEL to parametrize Crystal for AR_3 ("*Minimize* E_c | $PRR \geq 96\%$"). (a) Heatmap of normalized median goal values from 1000 GEL runs, with the best median goal value as computed by the oracle marked as a dashed blue line. (b) Optimality achieved as a function of the number of testbed trials, representing the percentage of runs that successfully identify what the oracle deems as best parameter set.

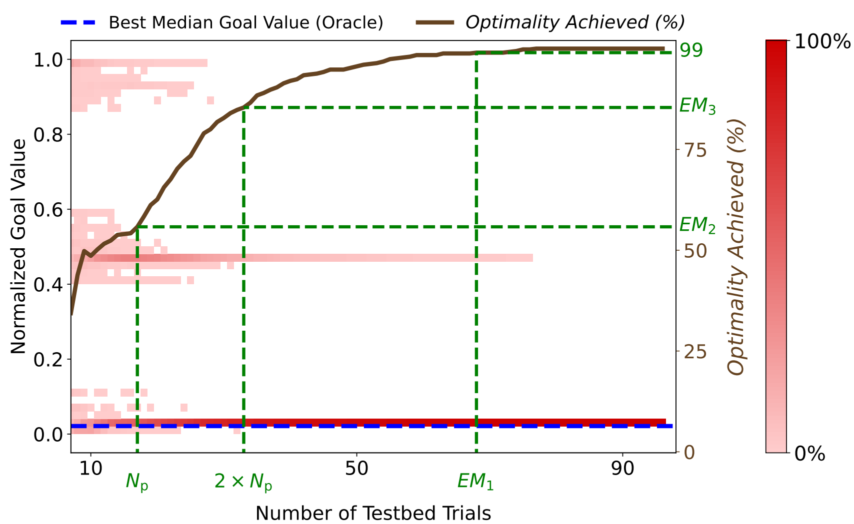


Fig. 8. Combined view of the evaluation outcome for the scenario shown in Fig. 7, highlighting the evaluation metrics (*EM*s) used in subsequent analyses.

Please note that the the oracle's knowledge used to derive the best median goal value and the achieved optimality (i.e., the number of runs that successfully identified what the oracle deems as the best parameter) is *not* available to APEX and to the optimization process in general; it is exclusively used for evaluation purposes.

Parameters and application requirements. Tab. 2 lists the relevant parameters used in our evaluation[30]. A single experiment runs for 10 minutes for Crystal and 18 minutes for RPL, with 1 and 3 minutes of settling time, respectively[31].

[30]Values such as δ and the RBF length are selected based on recommendations from the literature, and may be overwritten by expert users.
[31]We have verified with prior experiments that these durations ensure meaningful results.

Ref.	Goal	Constraint	Ref.	Goal	Constraint
AR_1	Minimize E_c	PRR ≥ 65%	AR_9	Minimize E_c	PRR ≥ 93%
AR_2	Minimize E_c	PRR ≥ 92%	AR_{10}	Maximize PRR	$E_c \leq 2940$ J
AR_3	Minimize E_c	PRR ≥ 96%	AR_{11}	Maximize PRR	$E_c \leq 2885$ J
AR_4	Maximize PRR	$E_c \leq 210$ J	AR_{12}	Maximize PRR	$E_c \leq 2879$ J
AR_5	Maximize PRR	$E_c \leq 190$ J	AR_{13}	Minimize E_c	PRR ≥ 0.975
AR_6	Maximize PRR	$E_c \leq 170$ J	AR_{14}	Maximize PRR	$E_c \leq 168$ J
AR_7	Minimize E_c	PRR ≥ 65.5%	AR_{15}	Minimize E_c	PRR ≥ 0.947
AR_8	Minimize E_c	PRR ≥ 88%	AR_{16}	Maximize PRR	$E_c \leq 2872$ J

Table 3. Considered application requirements (ARs). Goal: metric to be improved within the given constraint. Min.: minimize; Max.: maximize; E_c: energy consumption; PRR: packet reception ratio.

The settling time, which allows the network to stabilize before measurements begin, is particularly necessary for RPL as it is a routing-based protocol, and is therefore allocated more time. Note that users can define the test duration based on prior experience or in alignment with the literature. In the absence of these, the experimental setup can be derived using statistical tools, such as TriScale [30].

Tab. 3 lists the application requirements considered in our evaluation. We consider 16 different application requirements (labeled from AR_1 to AR_{16}) to test how different approaches behave with looser or tighter constraints (e.g., the ability to sustain a minimum PRR as low as 65% or as high as 97.5%). When selecting the application requirements, we focus on PRR and energy consumption (E_c) as performance metrics, and switch their role (as goal value or as constraint metric). Specifically, we pick some application requirements focusing on minimizing E_c while satisfying a given constraint on PRR, and other requirements focusing on maximizing the PRR while satisfying a given constraint on E_c.

We specifically use AR_1 to AR_6 to test Crystal, and AR_7 to AR_{12} to test RPL, as we aim to maintain a similar range of tightness in the selected constraints, and the two protocols sustain a largely different performance. For example, for Crystal, AR_1 to AR_3 are designed with the goal of reducing E_c with a progressively stricter constraint on PRR. AR_1 has the loosest constraint, with only one out of sixteen parameter sets failing to meet the required PRR. AR_2 introduces a tighter constraint, where half of the parameter sets fail to meet the PRR requirement. AR_3 introduces an even stricter constraint, with only three out of sixteen parameter sets satisfying the requirement. AR_4 to AR_6 follow the same concept, but with the goal of maximizing PRR given a constraint on E_c. Also in this case, out of sixteen parameter sets, 15, 8, and 3 satisfy the given constraint, respectively (this can also be observed in Fig. 2, which shows the performance of different parameter sets as energy drops from 230 to 150 J). We follow the same approach for RPL: AR_7 to AR_9 focus on reducing E_c while sustaining a minimum PRR, whilst AR_{10} to AR_{12} aim to maximize PRR while not exceeding E_c. Also for these application requirements, we strive to maintain a similar level of tightness in the constraints[32].

For each of the testbed protocols, we also select two application requirements with very tight constraints, such that only two parameter sets would provide a valid solution: these requirements are labeled AR_{13} to AR_{14} for Crystal, and AR_{15} to AR_{16} for RPL, respectively. These application requirements will be used to demonstrate that APEX outperforms other approaches also when a very limited number of solutions exists.

Results are expected at the 50th percentile with a maximum confidence of 98%, which necessitates at least six repetitions ($\rightarrow N_r = 6$) for each parameter set [30]. We set the number of initial trials n_{init} to 6 for APEX and the baseline approaches.

[32] For RPL, 33, 17 and 7 parameter sets out of 36 satisfy the constraints for AR_7 to AR_9. Likewise, 32, 18 and 8 parameter sets out of 36 satisfy the constraints for AR_{10} to AR_{12}.

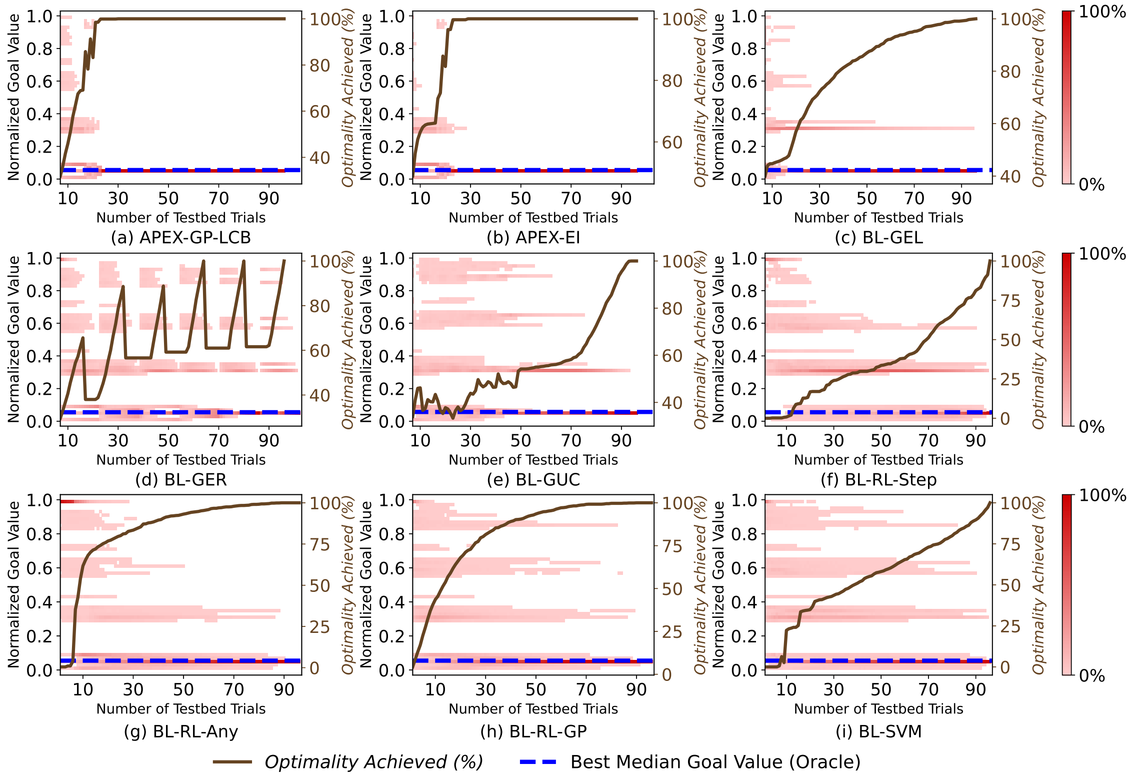


Fig. 9. Performance of various NTS algorithms evaluated when parametrizing the Crystal protocol for AR_2. APEX's algorithms are not trapped in local minima and are superior to other baseline approaches.

5.2 Performance of the Optimization Process

Analyzing the behaviour of NTS approaches. We start by evaluating the effectiveness of the next test-point selection (NTS) approaches used by APEX, showing that they are superior compared to the baseline strategies. Fig. 9 shows the performance of various NTS algorithms when finding Crystal's best parameter set for AR_2. Whilst APEX's GP-LCB and EI algorithms are able to quickly escape local minima, this is not the case for the baseline approaches following a greedy strategy (BL-GEL, BL-GER, BL-GUC), those employing RL (BL-RL-Any, BL-RL-Step, BL-RL-GP), and SVR (BL-SVM). It is to be noted that BL-RL-Any does perform well for a low number of testbed trials; however, its performance becomes similar to that of greedy approaches when it comes to achieving 99% optimality (EM_1).

EM_1 **(Crystal).** We analyze next the number of testbed trials needed to achieve an optimality of 99% when parametrizing Crystal for different application requirements. Note that the tightness of constraints varies across different application requirements, with constraints becoming tighter from AR_1 to AR_3 and from AR_4 to AR_6. Fig. 10 shows that, overall, APEX consistently outperforms or performs on par with baseline approaches, regardless of the application requirement and the tightness of its constraint. For example, for AR_2, APEX's GP-LCB and EI need only 20 and 22 trials such that 99% of the iterations find the best parameter set. In contrast, BL-GEL, BL-GER, BL-GUC, BL-RL-Step, BL-RL-Any, BL-RL-GP and BL-SVM need 85, 63, 91, 89, 64, 69, and 95 trials, respectively. Hence, APEX reaches optimality up to 4.75x faster

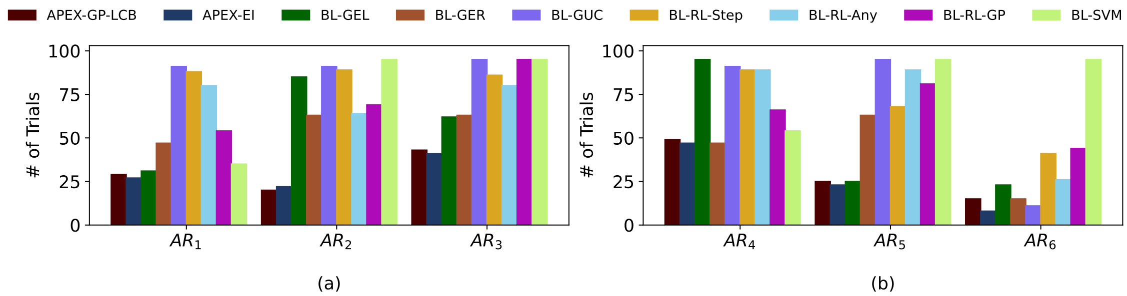


Fig. 10. Number of trials needed to achieve an *optimality* of 99% when evaluating Crystal for different application requirements with varying tightness in their constraint. We report two constellations: E_c as the goal value and the constraint given as the minimum required PRR (a); PRR as the goal value and the constraint given as the maximum allowed E_c (b).

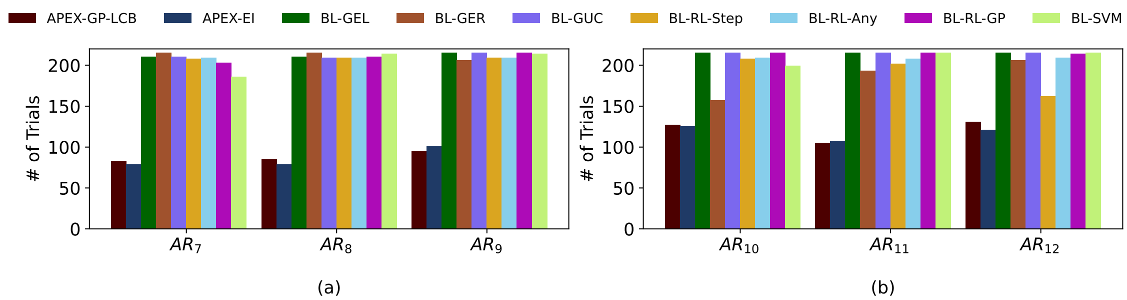
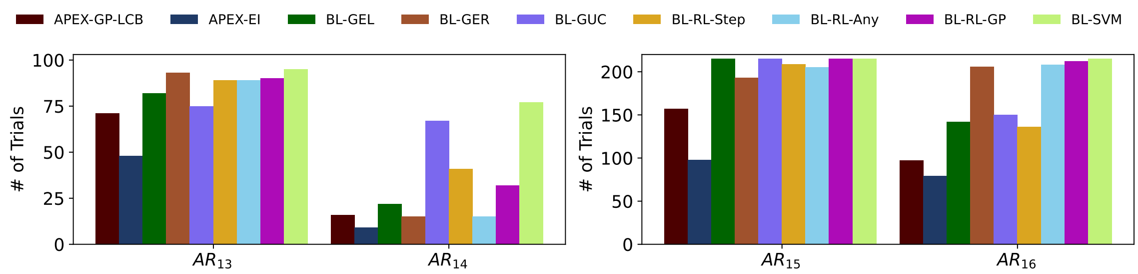


Fig. 11. Number of trials needed to achieve an *optimality* of 99% when evaluating RPL for different application requirements with varying tightness in their constraint. We report two constellations: E_c as the goal value and the constraint given as the minimum required PRR (a); PRR as the goal value and the constraint given as the maximum allowed E_c (b).

than BL approaches. For Crystal, the traditional exhaustive search approach would need 96 testbed trials to find the best parameter set, whereas APEX requires as little as nine: a 10.6x improvement in the best case. It is worth noting that the effectiveness of different approaches may vary based on the optimization task's specific requirements. If a scenario favors exploitation, BL-GEL would perform better, but in exploration-favoring situations, it may lag. Hence, judging an approach's superiority should consider its consistent performance across various situations. *APEX shows consistent performance regardless of the application requirement and the tightness of its constraint.*

EM_1 **(RPL).** We perform a similar evaluation for the RPL protocol, which has a different design philosophy and is less deterministic compared to Crystal. Constraints become tighter from AR_7 to AR_9 and from AR_{10} to AR_{12}. Fig. 11 shows that APEX outperforms baseline approaches by a significant margin. This superiority can be attributed to RPL being a less deterministic protocol than Crystal, which emphasizes APEX's ability to effectively handle noisy measurements.

Satisfying very tight constraints. When constraints are exceedingly tight, it may be enough to find at least one parameter set meeting the constraints. We hence evaluate how quickly such a parameter set can be found, focusing on extreme cases where only two parameter sets satisfy the constraint. Fig. 12 shows the number of trials required for 99% of iterations to discover a parameter set fulfilling given requirements for both Crystal and RPL. Observably, APEX's EI performs better compared to all other approaches, whereas APEX's GP-LCB performs better or on par with the baseline approaches. Specifically, EI requires up to 2.4, 2.6, 7.4, 4.5, 2.6, 4.0 and 8.4 times fewer testbed trials to find a



(a) Crystal (b) RPL

Fig. 12. Number of trials required to find the parameter set that satisfies the constraint (99th percentile) for Crystal (a) and RPL (b).

parameter set that satisfies the constraints, compared to BL-GEL, BL-GER, BL-GUC, BL-RL-Step, BL-RL-Any, BL-RL-GP, and BL-SVM, respectively.

We hence recommend, in general, the use of APEX's EI rather than GP-LCB, as the former performs better in scenarios with tighter constraint(s).

5.3 Performance with a Fixed Number of Trials

We select the requirements with the loosest to tightest constraints to study the optimality achieved after a fixed number of trials ($EM_2 \rightarrow N_p$ trials, $EM_3 \rightarrow 2 \cdot N_p$ trials).

Crystal. Fig. 13 shows the results for Crystal (with constraints becoming tighter from AR_1 to AR_3 and from AR_4 to AR_6). Overall, APEX outperforms the baseline approaches, especially with a large number of testbed trials (EM_3). Still, we can notice that for a few application requirements, some baseline approaches may offer a better or on-par performance. For example, BL-GEL outperforms APEX on AR_1 and AR_5 under EM_2, whereas BL-RL-GP outperforms it on AR_4 under EM_3. This is because APEX strikes a balance between exploration and exploitation, i.e., it may perform less optimally when very few testbed trials are available, but catches up achieving an optimality over 99%. Moreover, as mentioned earlier, one approach can perform better in a given scenario, but that does not mean it is superior; consistent performance across all situations is what makes an approach stand out in this kind of optimization.

RPL. Fig. 14 presents the results for RPL (with constraints becoming tighter from AR_7 to AR_9 and from AR_{10} to AR_{12}). Overall, APEX performs better or on-par for EM_3 (except for AR_{12} where BL-RL-Step performs better[33]). However, unlike Crystal, other baselines approaches outperform APEX for EM_2: this is expected, due to RPL's more stochastic nature. This overall behavior further reinforces our reasoning for Crystal's results: APEX may perform sub-optimally with fewer trials due to high noise, requiring more time to balance exploration and exploitation, whereas baseline approaches tend to act greedily, performing better initially, but failing in the long run. In fact, in most cases, APEX significantly outperforms these approaches with respect to EM_3, as well as to EM_1.

Summary. Overall, APEX approaches may perform sub-optimally with fewer trials, especially when there is higher stochasticity. However, they outperform baseline approaches in the long run. It is also noteworthy that APEX achieves 99% optimality in fewer testbed trials (EM_1) than the baselines, regardless of the application requirement or the tightness of the constraints, which is a key strength for surrogate optimization tasks. In such tasks, where a single approach may

[33] This result highlights a specific case in AR_{12}, where limited exploration around the current best, coupled with a penalty for not satisfying the constraint, proves beneficial. In this context, BL-RL-Step outperforms APEX. Nevertheless, when it comes to achieving 99% optimality (EM_1), BL-RL-Step performs worse compared to the APEX approaches, as shown in Fig. 11.

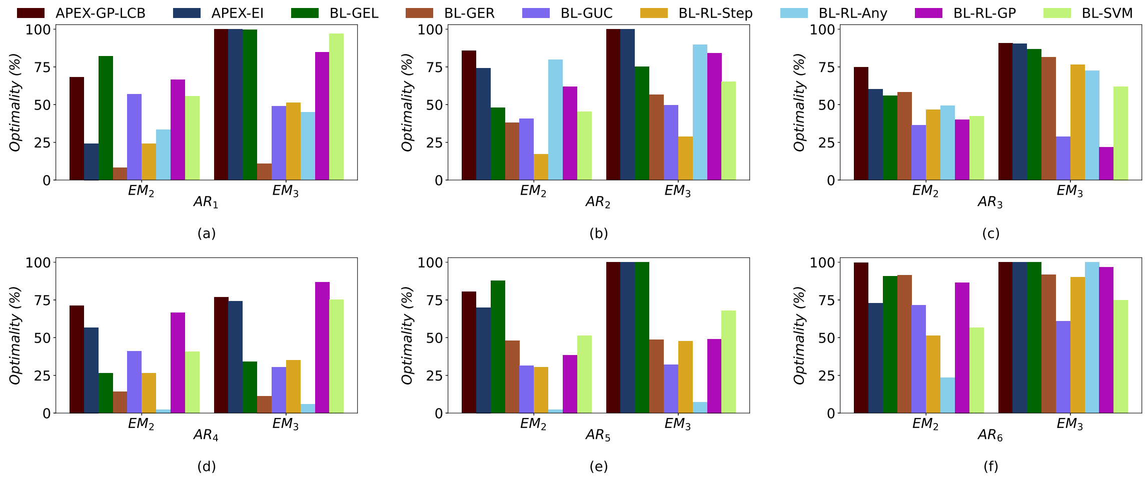


Fig. 13. Optimality achieved after a fixed number of trials when evaluating Crystal for application requirements with different tightness of their constraint. We report different constellations: E_c as the goal value with the constraint given as the minimum required PRR (a)–(c); PRR as the goal value with the constraint given as the maximum allowed E_c (d)–(f).

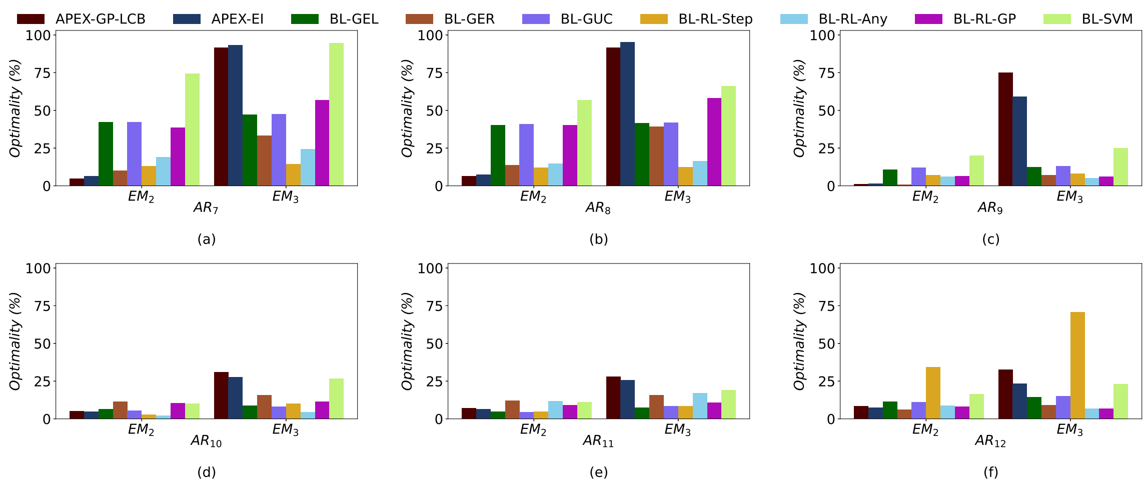


Fig. 14. Optimality achieved after a fixed number of trials when evaluating RPL for application requirements with different tightness of their constraint. We report different constellations: E_c as the goal value with the constraint given as the minimum required PRR (a)–(c); PRR as the goal value with the constraint given as the maximum allowed E_c (d)–(f).

excel in one scenario but fail in another, APEX's ability to maintain strong performance across diverse situations makes it particularly effective.

5.4 Suitability of α to Assess Optimality

We now evaluate how well the metric proposed to approximate the achieved optimality (α) performs compared to the actual optimality achieved. Fig. 15 illustrates the actual versus predicted optimality for AR_5, showing that α tracks the optimality trend better than baseline approaches (α_{B_1} and α_{B_2}).

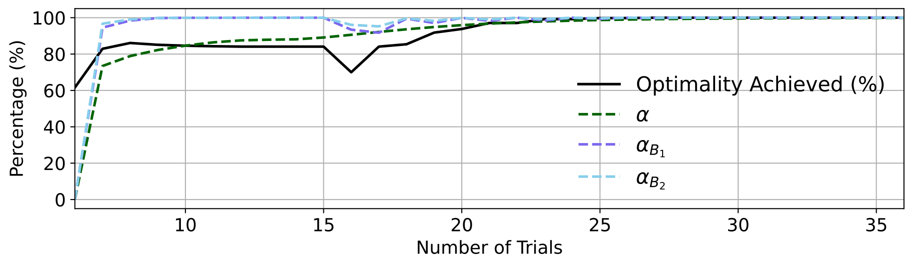


Fig. 15. Evaluation of various metrics approximating optimality for a given application requirement (AR_5).

To provide a quantitative evaluation, we compute the average difference in the number of testbed trials between when the process is terminated using estimated optimality and when the actual required level of optimality is achieved. For example, if a user specifies stopping the process at 80% confidence in optimality (as estimated by α), we calculate how many trials earlier or later the process stops compared to when the actual 80% optimality is achieved. The average of this difference in the number of testbed trials for α, α_{B_1}, and α_{B_2} under 80%, 90%, and 99% termination conditions for all application requirements listed in Tab. 3 except AR_{13} - AR_{16}[34] are 27.9, 52.4, and 52.6 trials, respectively. However, this does not assess how closely the predicted optimality matches the actual achieved optimality. Thus, we compute the root mean square deviation (RMSD) to assess how closely the predicted optimality aligns with the actual achieved optimality. After each testbed trial, we calculate the difference between the predicted and actual optimality, repeating this process for all ARs except AR_{13} to AR_{16}[34]. The RMSD is then computed based on these differences and averaged across all trials and ARs. The mean RMSD values for α, α_{B_1}, and α_{B_2} across all requirements are 23.21, 30.9, and 31.2, respectively. These results demonstrate the suitability of APEX's α as an effective metric for guiding the optimization process relative to the baseline approaches.

5.5 Evaluation in Higher Dimensional Settings

To further evaluate the versatility of our approach and understand its behavior in more complex optimization settings, we extend our investigation to higher dimensional parameter spaces. While the main experimental evaluation in previous sections focuses on parameter sets containing two to three parameters, we complement these evaluations with an additional evaluation designed to assess how APEX performs in more intricate parameter spaces, where a larger number of interacting configuration parameters must be optimized.

This analysis is motivated by the need to explore whether APEX can maintain its efficiency and accuracy as the dimensionality of the search space increases. This is an important property of any optimization framework intended for general applicability. In practice, such higher dimensional optimization scenarios are not only more computationally demanding, but also pose significant challenges due to the increased size of the search space and the presence of complex inter-dependencies among parameters.

As already discussed in § 5.1, executing exhaustive testbed trials is unfortunately impractical – especially when the goal is to establish a reliable ground truth for comparison. For example, increasing the number of parameters to five, each with three possible values, results in 243 unique parameter combinations. To evaluate the performance of different optimization approaches against the oracle's best configuration, it is necessary to exhaustively test all combinations

[34]We use AR_1 to AR_6 for Crystal and AR_7 to AR_{12} for RPL. AR_{13} to AR_{16} are excluded as they are only intended for identifying a parameter set that satisfies the constraints rather than finding the best parameter set.

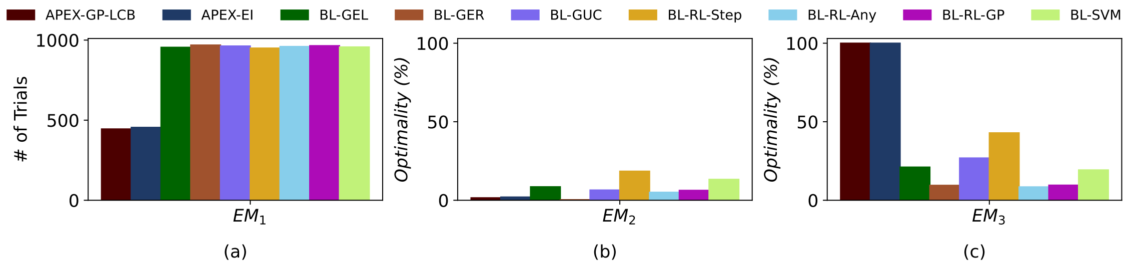


Fig. 16. Evaluation results for synthetic 5-parameter configuration space of RPL for application requirement AR_9. We compare APEX variants against all baselines using the following evaluation metrics. EM_1: number of testbed trials needed to achieve an *optimality* of 99% (a); EM_2: *optimality* achieved after performing N_p trials (b); EM_3: *optimality* achieved after performing $2 \cdot N_p$ trials (c).

multiple times. This is required to obtain statistically sound median values and to account for the stochastic nature of real-world wireless environments. Performing three repetitions per configuration demands 729 individual testbed runs, which would require several months of continuous experimentation on a shared testbed infrastructure. It is important to note that this effort is solely to establish the oracle reference for comparison. In practice, APEX would need only a fraction of these trials to identify the best performing parameter set, as it converges quickly using model-based exploration (we will further demonstrate its live performance in § 5.6). Hence, while the generation of the oracle requires exhaustive evaluation, running APEX in a real deployment setting would be significantly more time efficient.

To address this challenge while still gaining insight into APEX's performance in more complex settings, we construct a synthetic dataset that builds on the characteristics observed in real testbed measurements. In particular, we extend the RPL protocol by adding two additional parameters – RPL DAO delay and RPL DIS interval – resulting in a total of five parameters for this analysis. Each parameter is assigned three discrete values, leading to 243 unique parameter configurations. To mimic the natural variability of wireless systems, each configuration is evaluated four times, yielding a total of 972 synthetic observations. The dataset is constructed using rule-based models that reflect the behavioral patterns found in real testbed data, such as the impact of different parameter settings on reliability and energy consumption. Controlled noise is added using variance models learned from actual testbed results, allowing the synthetic data to capture non-linear interactions and realistic performance variability[35]. Hence, this synthetic evaluation enables a fair and scalable comparison of optimization strategies in a high-dimensional setting without incurring the prohibitive costs of exhaustive real-world experimentation[36].

We use this synthetic dataset to compare APEX against baseline approaches, including greedy selection, reinforcement learning with Q-table, reinforcement learning with Gaussian processes, and support vector regression. All methods are evaluated on application requirement AR_9. Fig. 16 shows the overall results using the same evaluation metrics (*EMs*) introduced earlier. As observed in previous sections, APEX consistently outperforms all baseline methods by a considerable margin in EM_1, requiring only half the number of testbed trials to achieve 99% *optimality*. For EM_2, baseline methods may perform better when only a limited number of trials are available, as observed in earlier evaluations. However, as more trials are allowed, APEX surpasses all baselines in EM_3. This reinforces the earlier observation that while baselines can show early gains, APEX excels in both consistency and long-term performance.

[35]The script used to create the synthetic data is available in the APEX repository (http://iti.tugraz.at/APEX) under `results/Synthetic_Data`.
[36]We acknowledge that synthetic data cannot perfectly capture all aspects of real-world behavior. However, by incorporating empirical performance trends and noise characteristics derived from actual testbed results, the generated data provides a sufficiently realistic approximation for evaluating model performance in higher-dimensional scenarios.

5.6 APEX in Action

In the previous sections, we evaluated APEX using both real-world and synthetic traces. We showcase next the performance of APEX when parametrizing "live" both RPL and Crystal: we do so by employing different settings than those used previously, and by introducing dynamicity in the environment in order to also highlight the general applicability of our framework. We run our evaluations across different testbed layouts (i.e., diverse combinations of source and destination nodes[37]) and interference conditions. Specifically, we artificially generate interference (i.e., RF noise) using JamLab-NG [46] across all Wi-Fi channels, and distinguish between *static* and *dynamic* interference; the former is time-invariant[38], whereas the latter alternates 50-second spans of interference with equally long interference-free gaps. In all our live experiments shown next, we use APEX EI (which was previously shown to perform best in most scenarios) and run 120 testbed trials for each setting, marking the number of runs Γ after which APEX would have terminated (when reaching a level of confidence in optimality α of 0.9). This allows us to quantify the actual optimality of the parameter set returned by APEX after Γ runs by comparing it to the best median goal value derived over the entire 120 testbed trials.

RPL. We parametrize RPL using application requirement AR_{11} (maximize PRR while ensuring $E_c \leq 2885$ J), and execute six independent live runs in each of the four considered scenarios, which are summarized in Fig. 17. First, we run the protocol using testbed layout 1 in absence of interference (Fig. 17(a)). We then run the protocol on the same testbed layout with static (Fig. 17(b)) and dynamic (Fig. 17(c)) interference. Finally, we run RPL on a different testbed layout (layout 7) in absence of interference (Fig. 17(d)). Each live run lasts five minutes, which is sufficient to capture the protocol dynamics while minimizing the overall experimentation time.

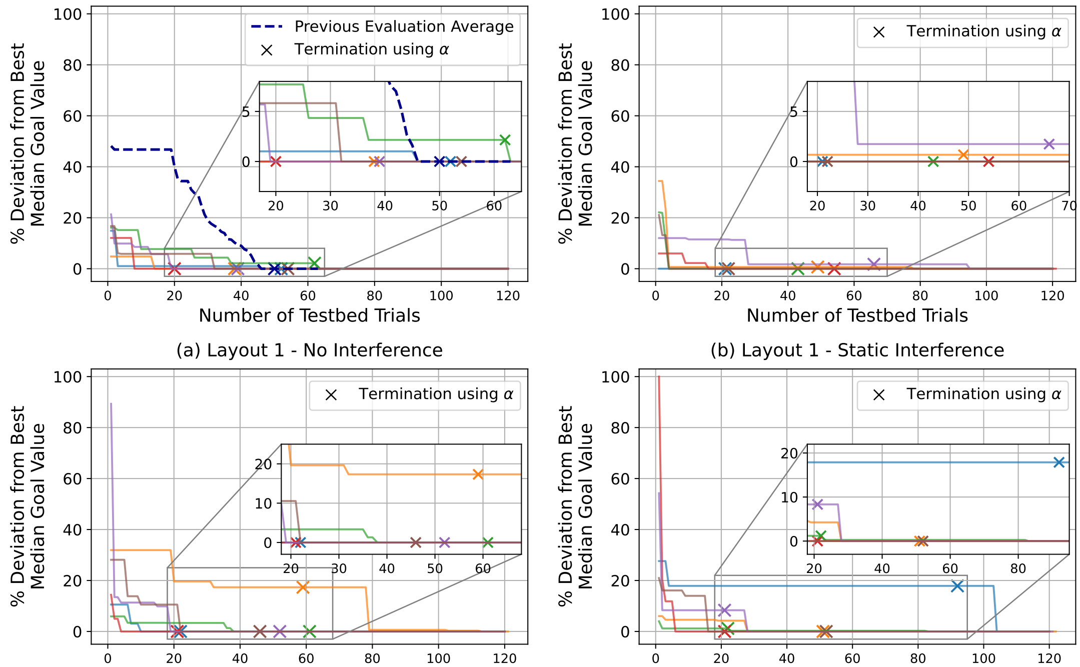
From the results depicted in Fig. 17, we can infer that APEX terminates after a median of 45, 46, 49, and 36 trials in each of the four scenarios (a)–(d), reducing the experimentation time by about 63%, 64%, 58%, and 64%, respectively, compared to an exhaustive search. These results indicate that APEX's performance is consistent despite different experimental settings. When analyzing the results in Fig. 17(a), we can also get a quantitative comparison between the results obtained in the previous evaluations (derived using real-world traces in Fig. 14) with those obtained with "live" runs. Specifically, the blue dashed line reports the average deviation from the best median goal value obtained earlier (Fig. 14): 48 trials are needed to terminate (blue cross). This value almost matches the 45 trials that were required from the "live" runs.

Crystal. For Crystal, we report the performance of APEX in two scenarios that complement the earlier evaluation, which was carried out on testbed layout 1 (Fig. 13). First, we parametrize Crystal for AR_1 (minimize E_c while maintaining PRR $\geq$ 65%) using testbed layout 4 (Fig. 18(a)). Second, we parametrize Crystal for AR_4 (maximize PRR while ensuring $E_c \leq 210$J) using the original testbed layout (layout 1), but introducing static interference (Fig. 18(b)). We execute six independent live runs per scenario, each lasting four minutes.

From the results depicted in Fig. 18, we can infer that APEX terminates after a median of 28 trials in scenario (a) and 31 trials in scenario (b), thereby reducing the experimentation time by about 83% and 73%, respectively, compared to an exhaustive search. The blue dashed lines report the average deviation from the best median goal value obtained earlier (Fig. 13): 30 trials are needed to terminate (blue cross). This value matches well the number of trials after which APEX returns during the "live" runs.

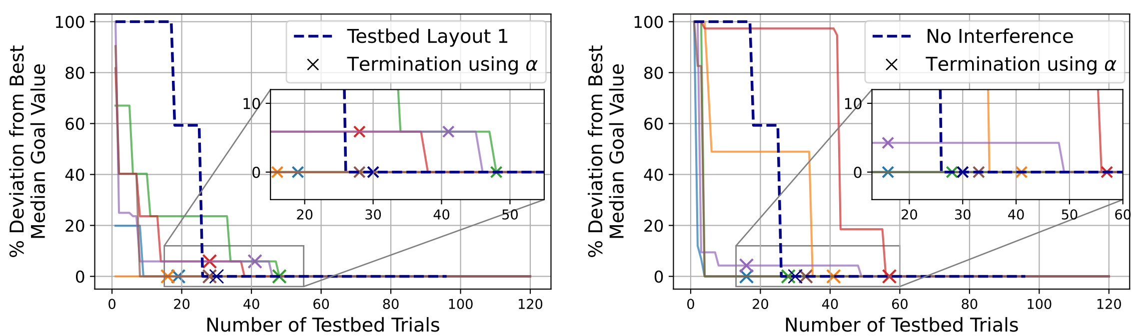
[37] All testbed layouts used in this work are data collection layouts (nRF52840 Timely Data Collection v1). Details about these layouts are available in D-Cube's Webpage: https://iti-testbed.tugraz.at/layout.

[38] We leverage a mild interference pattern provided by D-Cube, which generates repeatable intermittent bursts of Wi-Fi interference using Raspberry Pi 3B nodes placed in the surroundings of all testbed nodes for a duration of roughly 5 ms in periods of 13 ms using a transmission power of ≈ 30 mW.



(a) Layout 1 - No Interference
(b) Layout 1 - Static Interference
(c) Layout 1 - Dynamic Interference
(d) Layout 7 - No Interference

Fig. 17. Performance of APEX when parametrizing “live” RPL across four different scenarios. The plots show the deviation from the best median goal value as a function of the number of executed testbed trials: a value of 0% coincides with the best median goal value (obtained over all 120 testbed trials) and hence implies that APEX correctly identified the best parameter set. Each solid line is one live run; the "X" marks the first trial at which the confidence in optimality α (0.9 in this case) is reached, indicating when APEX would have terminated.



(a) Layout 4 - No Interference
(b) Layout 1 - Static Interference

Fig. 18. Performance of APEX when parametrizing “live” Crystal across two different scenarios. The two plots show the deviation from the best median goal value as a function of the number of executed testbed trials: a value of 0% coincides with the best median goal value and hence implies that APEX correctly identified the best parameter set. Each of the solid lines in the plot represents an independent optimization run; the “X” marks the trial in which APEX would have terminated (i.e., the point in which APEX’s confidence-in-optimality metric α reaches 90%).

6 Discussion & Future Work

Multi-objective goal value. In some applications, there may be multiple metrics that need to be optimized for. For instance, enhancing both PRR and E_c could be needed. In such cases, constructing multi-objective goal values requires determining priority weights for metrics and obtaining estimated ranges or expected statistical values for normalization, thus mitigating bias from absolute value differences. If these aspects are available, APEX can parameterize a multi-objective goal function; otherwise, an alternative approach must be explored to avoid bias favoring one metric over another, pointing to a potential area for future research in understanding Pareto optimality in multi-objective optimization.

Flexible constraint(s). Here, constraints are defined as single values that must be met with a specified confidence level. However, this approach may discard parameter sets that are close to meeting the constraints but offer better goal values. To address this, understanding how to model flexibility in constraint metrics and integrate it into the parameterization process is an interesting avenue for future investigations.

Latency. The time to compute the next test-point, depending on the model and available data, is typically minimal compared to a single trial. For instance, using GPs with 200 input samples, the NTS took about 1 second on average[39], a negligible duration compared to the testbed trials of 18 minutes. Even when the sample size is increased to 972 input samples, as in our high-dimensional evaluation, the average computation time increases to only 20 seconds, which still remains negligible in comparison. However, if latency is an issue, we can precalculate future test points while updating the current model with new results using NTS approaches. If the updated model suggests the same point as the one just conducted, we can test the second-best point from the updated model, and so forth.

General applicability & extensions. In this paper, we have explicitly focused on *testbed* experiments. This choice was dictated by the fact that experimentation on testbeds is often the preferred way to test and debug the performance of LPW protocols using real hardware [7]. In fact, testbeds provide a controlled yet realistic environment to explore parameter configurations while approximating field conditions. This is very convenient, given the practical challenges and costs of performing extensive parameter optimization directly in a real-world deployment. However, APEX is not bound to testbed experimentation: the experimental data could also be collected through simulation or in a real-world deployment, and the functionality of the framework's inner modules would be agnostic to this (only the way in which the experimental trials should be executed needs to be changed accordingly). Moreover, APEX is designed for LPW protocol optimization in this study. However, its approach of treating the cost function as a black box suggests the potential for extensions to tackle similar challenges in domains where evaluations are expensive and brute force methods with statistical significance are required but impractical.

Kernel selection. For our evaluation, we opted for the RBF kernel (with length scale = 1), a widely-used choice when correlation behavior is unknown. However, users can select different kernels based on preferences for smoothness or other properties. We also assessed the Matérn kernel, another common option, and found its performance comparable.

User in the loop. APEX is designed to function autonomously once the required user inputs are provided. However, users must make certain decisions to ensure they get the best results from the APEX framework. First, the duration of each experiment and the corresponding experimental settings should be long enough to produce meaningful results. These can be determined based on prior experience, literature, or statistical analysis [30]. Additionally, users should

[39]System specifications: Intel Core i5-10500 processor, 3.10 GHz; 32.0 GB of RAM (31.8 GB usable). The `GaussianProcessRegressor` library from sklearn was used for regression.

ensure that the selected parameters can be represented in a metric space, meaning they exhibit some form of correlation with changes in their values, even if the relationship is not strictly linear or convex. When dealing with protocols that have only a few parameters, it is possible to optimize all of them. However, when there are many parameters, optimizing all of them may become impractical [52]. In such cases, users can select parameters related to the performance metrics or identify the most influential parameters [2], which may require additional experiments. Once the key parameters are chosen, the experimental setup can be determined using tools like TriScale [30], which suggests appropriate experiment duration and the number of repetitions per parameter set based on specific requirements.

7 Related Work

Over the years, LPW applications have gained significant attention, leading to the development of various LPW protocols tailored to different applications and needs. Optimizing these protocol parameters has become crucial for maximizing performance, prompting extensive research efforts in this domain. Below, we summarize existing works in this field, highlighting important studies, methodologies, and findings.

Parametrization for specific settings. Early research focused on parameter exploration through measurement studies in specific settings [17, 18, 40]. For example, [17] analyzed the impact of duty cycling on average delay and packet loss rate in the IEEE 802.15.4 protocol stack. Similarly, [18] proposed a systematic methodology to investigate packet delivery performance in large-scale LPW networks, while [40] focused on optimizing packet size and error correction in LPW networks. Efforts also targeted specific protocols like Bluetooth [36, 50], Zigbee [26], and LoRa [32]. While these studies provided valuable insights, they often focused on individual parameters, limiting inherent trade-offs. Joint multi-parameter exploration began with the work by Fu et al. [21], which provided guidelines for multi-layer parameter optimization across various performance metrics and analyzed a specific protocol in a point-to-point communication system. Later, Mazloomi et al. [38] have utilized the measurement database from [21] to model networks using support vector regression and optimized performance with a genetic algorithm. Karunanayake et al. [31] have presented an adaptive approach based on reinforcement learning to parametrize the collection tree protocol (CTP [22]). Such adaptive approach runs directly on an IEEE 802.15.4 device, i.e., it optimizes protocol parameters on individual nodes without accounting for the actual performance across the entire network. Similarly, Li et al. [35] proposed a lightweight reinforcement learning method for LoRaWAN, where each device independently selects its spreading factor and channel based only on local acknowledgments. This approach focuses on continuous adaptation to improve performance, but remains limited to device-level optimization for two parameters in a single-hop setting. In contrast, APEX focuses on network-wide optimization. Even when reusing or extending the reinforcement learning method proposed in [31], APEX yields up to 3.25x better results. Moreover, with APEX, we provide a generic framework for protocol parametrization and test it on multiple protocols with a large variety of application requirements.

Modular parametrization. Multiple modular frameworks have been proposed to optimize protocol parameters. One of the pioneering works in this domain is pTUNES [55], a runtime optimization framework that dynamically adjusts MAC protocol parameters in LPW networks to meet application requirements such as network lifetime, reliability, and latency based on real-time network state data. Another framework that expands beyond the optimization of MAC protocol parameters is proposed in [41] to automate the configuration of IoT communication protocols. However, it requires user expertise in model definition, as it leverages environmental and hardware models for adaptation. The MakeSense project [12, 39] has the closest relation to our work, although its primary focus is on reducing costs. Specifically, the authors applied reinforcement learning on a simulation of the deployed network, with characteristics of the real network being automatically collected. Although impressive, this contribution is not directly applicable to our context, as it would require a new protocol developer to implement MakeSense's low-level functionalities within their protocol. In a prior poster abstract [27], we articulated the necessity for an automated parameter selection framework for LPW systems and outlined its potential design. This paper represents the concrete realization of this initial vision.

Among all these modular frameworks, none enable parametrization for non-experts nor focus on reducing the experimentation time for parametrization. With APEX, we are the first to propose a modular framework that can be used by non-experts while also focusing on reducing the experimentation time.

8 Conclusion

In this paper, we introduce APEX, a framework designed to streamline the parametrization process of LPW protocols. By automating parameter exploration based on real-world testbed data and by leveraging Gaussian processes, APEX offers an efficient solution to protocol parameterization that does not require users to be heavily involved in the optimization process and to possess in-depth protocol understanding.

Our empirical evaluations underscore the effectiveness of APEX in efficiently identifying optimal parameter sets, demonstrating a significant reduction in the number of necessary testbed trials compared to traditional approaches. Among others, we showcase APEX's ability to parameterize two state-of-the-art IEEE 802.15.4 protocols, to adapt to diverse application requirements, and to minimize the efforts and costs associated with the parameter tuning process. By making APEX open-source[3], we aim to empower researchers and practitioners in the field of LPW systems with the ability to develop dependable networking solutions that can meet stringent application requirements.

Acknowledgments

This research was funded in whole, or in part, by the Austrian Science Fund (FWF) [10.55776/DFH5]. For the purpose of open access, the author has applied a CC BY public copyright license to any Author Accepted Manuscript version arising from this submission.

References

[1] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne. 2015. FIT IoT-LAB: A Large Scale Open Experimental IoT Testbed. In *Proc. of the 2nd World Forum on the Internet of Things (WF-IoT)* (Milan, Italy). IEEE, Piscataway, NJ, USA, 459–464. https://ieeexplore.ieee.org/document/7389098

[2] R. Adán-López, D. Fernández-Martínez, R.A. Rodríguez-Gómez, and J. Camacho. 2024. Coupled Design and Analysis of Experiments in Network Management. In *Proceedings of the 37th Network Operations and Management Symposium (NOMS)* (Seoul, Korea). IEEE, Piscataway, NJ, USA, 1–5. https://doi.org/10.1109/NOMS59830.2024.10575504

[3] P. Appavoo, E. K. William, M. C. Chan, and M. Mohammad. 2018. Indriya2: A Heterogeneous Wireless Sensor Network (WSN) Testbed. In *Proceedings of the 13th TridentCom Conference* (Shanghai, China). Springer, Berlin, Heidelberg, 3–19. https://link.springer.com/chapter/10.1007/978-3-030-12971-2_1

[4] S. Balduin, T. Westermann, and E. Puiutta. 2020. Evaluating Different Machine Learning Techniques as Surrogate for Low Voltage Grids. *Energy Informatics* 3 (Oct. 2020), 1–12. https://doi.org/10.1186/s42162-020-00127-3

[5] M. Banaszek, M. Schuß, C.A. Boano, and K. Iwanicki. 2024. RPL at Scale: Experiences from a Performance Evaluation on up to 700 IEEE 802.15.4 Devices. In *Proceedings of the IEEE Network Operations and Management Symposium (NOMS)*. IEEE, Piscataway, NJ, USA. https://doi.org/10.1109/NOMS59830.2024.10574969

[6] C.A. Boano et al. 2014. TempLab: A Testbed Infrastructure to Study the Impact of Temperature on Wireless Sensor Networks. In *Proceedings of the 13th International Conference on Information Processing in Sensor Networks (IPSN)* (Berlin, Germany). IEEE, 95–106. https://doi.org/10.1109/IPSN.2014.6846744

[7] C.A. Boano, S. Duquennoy, A. Förster, O. Gnawali, R. Jacob, H.-S. Kim, O. Landsiedel, R. Marfievici, L. Mottola, G.P. Picco, X. Vilajosana, T. Watteyne, and M. Zimmerling. 2018. IoTBench: Towards a Benchmark for Low-Power Wireless Networking. In *Proc. of the 1st CPSBench Workshop* (Porto, Portugal). IEEE, Piscataway, NJ, USA, 36–41. https://ieeexplore.ieee.org/document/8429500

[8] M. Bor and U. Roedig. 2017. LoRa Transmission Parameter Selection. In *Proc. of the 13th DCOSS Conference* (Ottawa, Canada). IEEE, Piscataway, NJ, USA, 27–34. https://ieeexplore.ieee.org/document/8271941

[9] V. Borovitskiy, I. Azangulov, A. Terenin, P. Mostowsky, M. Deisenroth, and N. Durrande. 2021. Matérn Gaussian Processes on Graphs. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics*, Vol. PMLR 130. MLResearchPress, 2593–2601. https://proceedings.mlr.press/v130/borovitskiy21a.html

[10] E. Brochu, V.M. Cora, and N. de Freitas. 2010. *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning.* Technical Report CORR – arXiv preprint 1012.2599. Cornell University. http://arxiv.org/abs/1012.2599

[11] L. Campanile, M. Gribaudo, M. Iacono, F. Marulli, and M. Mastroianni. 2020. Computer Network Simulation with ns-3: A Systematic Literature Review. *Electronics* 9, 2 (2020), 1–25. https://doi.org/10.3390/electronics9020272

[12] F. Casati, F. Daniel, G. Dantchev, J. Eriksson, N. Finne, S. Karnouskos, P.M. Montera, L. Mottola, F.J. Oppermann, G.P. Picco, A. Quartulli, K. Römer, P. Spiess, S. Tranquillini, and T. Voigt. 2012. Towards Business Processes Orchestrating the Physical Enterprise with Wireless Sensor Networks. In *Proceedings of the 34th International Conference on Software Engineering (ICSE).* IEEE, Piscataway, NJ, USA, 1357–1360. https://doi.org/10.1109/ICSE.2012.6227080

[13] M. Cattani, C.A. Boano, and K. Römer. 2017. An Experimental Evaluation of the Reliability of LoRa Long-Range Low-Power Wireless Communication. *Journal of Sensor and Actuator Networks (JSAN)* 6, 2 (June 2017), 1–19. https://doi.org/10.3390/jsan6020007

[14] U.M. Colesanti et al. 2007. On the Accuracy of OMNeT++ in the Wireless Sensor Networks Domain: Simulation vs. Testbed. In *Proceedings of the 4th International Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN)* (Chania, Crete Island, Greece). ACM, New York, NY, USA, 25–31. https://doi.org/10.1145/1298197.1298203

[15] Contiki-NG Contributors. 2024. RPL-Lite in Contiki-NG. https://docs.contiki-ng.org/en/develop/doc/programming/RPL.html#rpl-lite Accessed: December 2024.

[16] H. A. David and H. N. Nagaraja. 2005. Order Statistics in Nonparametric Inference. In *Wiley Series in Probability and Statistics.* John Wiley & Sons, Inc., Hoboken, NJ, USA, 159–170. https://doi.org/10.1002/0471722162.ch7

[17] F. Despaux, Y.-Q. Song, and A. Lahmadi. 2013. Measurement-based Analysis of the Effect of Duty Cycle in IEEE 802.15.4 MAC Performance. In *Proc. of the 2013 IEEE 10th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS).* IEEE, Piscataway, NJ, USA, 620–626. https://doi.org/10.1109/MASS.2013.102

[18] W. Dong, Y. Liu, Y. He, T. Zhu, and C. Chen. 2014. Measurement and Analysis on the Packet Delivery Performance in a Large-Scale Sensor Network. *IEEE/ACM Transactions on Networking* 22, 6 (Dec. 2014), 1952–1963. https://ieeexplore.ieee.org/document/6567076

[19] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. 2011. Efficient Network Flooding and Time Synchronization with Glossy. In *Proc. of the 10th IPSN Conference* (Chicago, IL, USA). IEEE, Piscataway, NJ, USA, 73–84. https://ieeexplore.ieee.org/document/5779066

[20] F. Foukalas, P. Pop, F. Théoleyre, C.A. Boano, and C. Buratti. 2019. Dependable Wireless Industrial IoT Networks: Recent Advances and Open Challenges. In *Proc. of the 24th European Test Symposium* (Baden Baden, Germany). IEEE, Piscataway, NJ, USA, 1–10. https://ieeexplore.ieee.org/document/8791551

[21] S. Fu, Y. Zhang, Y. Jiang, C. Hu, C.Y. Shih, and P.J. Marrón. 2015. Experimental Study for Multi-layer Parameter Configuration of WSN Links. In *Proc. of the 35th ICDCS Conference* (Columbus, OH, USA). IEEE, Piscataway, NJ, USA, 369–378. https://doi.org/10.1109/ICDCS.2015.45

[22] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. 2009. Collection Tree Protocol. In *Proceedings of the 7th Conference on Embedded Networked Sensor Systems (SenSys).* ACM, New York, NY, USA, 1–14. https://dl.acm.org/doi/10.1145/1644038.1644040

[23] R.B. Gramacy. 2020. *Surrogates: Gaussian Process Modeling, Design, and Optimization for the Applied Sciences* (1st ed.). CRC Press, Boca Raton, FL, USA. https://doi.org/10.1201/9780367815493

[24] Graz University of Technology. 2024. D-Cube: A Low-Power Wireless Networking Benchmark. [Online] https://iti-testbed.tugraz.at – Last access: 2024-06-30.

[25] B. Großwindhager, C.A. Boano, M. Rath, and K. Römer. 2018. Enabling Runtime Adaptation of Physical Layer Settings for Dependable UWB Communications. In *Proceedings of the 19th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)* (Chania, Greece). IEEE Computer Society, Washington DC, United States, 1–11. https://ieeexplore.ieee.org/document/8449776

[26] J. Han. 2009. Global Optimization of ZigBee Parameters for End-to-End Deadline Guarantee of Real-Time Data. *IEEE Sensors Journal* 9, 5 (March 2009), 512–514. https://ieeexplore.ieee.org/document/4806310

[27] H. Hydher, M. Schuß, O. Saukh, C.A. Boano, and K. Römer. 2023. Poster: Automatic Parameter Exploration for Low-Power Wireless Protocols. In *Proceedings of the 2023 International Conference on Embedded Wireless Systems and Networks* (Rende, Italy) *(EWSN '23).* Association for Computing Machinery, New York, NY, USA, 311–312. https://dl.acm.org/doi/10.5555/3639940.3639986

[28] T. Istomin, A.L. Murphy, G.P. Picco, and U. Raza. 2016. Data Prediction + Synchronous Transmissions = Ultra-low Power Wireless Sensor Networks. In *Proc. of the 14th SenSys Conference* (Stanford, CA, USA). ACM, New York, NY, USA, 83—-95. https://dl.acm.org/doi/10.1145/2994551.2994558

[29] R. Jacob, J. Baechli, R. Da Forno, and L. Thiele. 2019. Synchronous Transmissions Made Easy: Design Your Network Stack with Baloo. In *Proc. of the 16th EWSN Conference* (Beijing, China). Junction Publishing, USA, 106–117. https://dl.acm.org/doi/10.5555/3324320.3324334

[30] R. Jacob, M. Zimmerling, C.A. Boano, L. Vanbever, and L. Thiele. 2021. Designing Replicable Networking Experiments With Triscale. *Journal of Systems Research (JSys)* 1, 1 (Nov. 2021), 1–26. https://escholarship.org/uc/item/63n4s9w2

[31] P.N. Karunanayake, A. Könsgen, T. Weerawardane, and A. Förster. 2023. Q learning based adaptive protocol parameters for WSNs. *Journal of Communications and Networks* 25, 1 (2023), 76–87. https://doi.org/10.23919/JCN.2022.000035

[32] G. Kaur, S.H. Gupta, and H. Kaur. 2022. Optimizing the LoRa Network Performance for Industrial Scenario using a Machine Learning Approach. *Computers and Electrical Engineering* 100 (May 2022), 1–16. https://www.sciencedirect.com/science/article/pii/S0045790622002403

[33] J. Kirschner and A. Krause. 2018. Information Directed Sampling and Bandits with Heteroscedastic Noise. In *Proc. of the 31st Conference On Learning Theory* (Stockholm, Sweden). MLResearchPress, 358–384. http://proceedings.mlr.press/v75/kirschner18a/kirschner18a.pdf

[34] M. Lewandowski and B. Płaczek. 2021. Data Transmission Reduction in Wireless Sensor Network for Spatial Event Detection. *Sensors* 21, 21 (2021), 1–21. https://doi.org/10.3390/s21217256

[35] A. Li et al. 2022. A Lightweight Transmission Parameter Selection Scheme Using Reinforcement Learning for LoRaWAN. arXiv:2208.01824 [cs.LG] https://arxiv.org/abs/2208.01824

[36] A. Liendo, D. Morche, R. Guizzetti, and F. Rousseau. 2018. BLE Parameter Optimization for IoT Applications. In *Proc. of the International Conference on Communications (ICC)* (Kansas City, MO, USA). IEEE, Piscataway, NJ, USA, 1–7. https://ieeexplore.ieee.org/document/8422714

[37] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel. 2013. FlockLab: A Testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems. In *Proc. of the 12th IPSN Conference* (Philadelphia, PA, USA). IEEE, Piscataway, NJ, USA, 153–166. https://ieeexplore.ieee.org/document/6917582

[38] N. Mazloomi, M. Gholipour, and A. Zaretalab. 2022. Efficient Configuration for Multi-Objective QoS Optimization in Wireless Sensor Network. *Ad Hoc Networks* 125, C (Feb. 2022), 18 pages. https://dl.acm.org/doi/10.1016/j.adhoc.2021.102730

[39] L. Mottola, G. P. Picco, F. J. Oppermann, J. Eriksson, N. Finne, H. Fuchs, A. Gaglione, S. Karnouskos, P. Moreno Montero, N. Oertel, K. Römer, P. Spieß, S. Tranquillini, and T. Voigt. 2019. makeSense: Simplifying the Integration of Wireless Sensor Networks into Business Processes. *IEEE Transactions on Software Engineering* 45, 6 (June 2019), 576–596. https://ieeexplore.ieee.org/document/8240710

[40] C. Noda, S. Prabh, M. Alves, and T. Voigt. 2013. On Packet Size and Error Correction Optimisations in Low-power Wireless Networks. In *Proc. of the 10th SECON Conference* (New Orleans, LA, USA). IEEE Computer Society, Piscataway, NJ, USA, 212–220. https://ieeexplore.ieee.org/document/6644980

[41] F.J. Oppermann, C.A. Boano, M.A. Zúñiga, and K. Römer. 2015. Automatic Protocol Configuration for Dependable Internet of Things Applications. In *Proc. of the 10th SenseApp Workshop* (Clearwater Beach, FL, USA). IEEE, Piscataway, NJ, USA, 742–750. https://ieeexplore.ieee.org/document/7365923/

[42] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. 2006. Cross-Level Sensor Network Simulation with COOJA. In *Proceedings of the 31st Conference on Local Computer Networks (LCN)* (Tampa, FL, USA). IEEE, Piscataway, NJ, USA, 641–648. https://ieeexplore.ieee.org/document/4116633

[43] N.V. Queipo, R.T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P.K. Tucker. 2005. Surrogate-based Analysis and Optimization. *Progress in Aerospace Sciences* 41, 1 (Jan. 2005), 1–28. https://www.sciencedirect.com/science/article/pii/S0376042105000102

[44] E. Salomon and C.A. Boano. 2024. Adaptive Transmission Power Control in BLE: Unveiling and Overcoming the Limits of Current Solutions. In *Proceedings of the 20th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)* (Paris, France). IEEE, Piscataway, NJ, USA, 455–462. https://ieeexplore.ieee.org/document/10770502

[45] M. Schuh, M. Baddeley, K. Römer, and C.A. Boano. 2024. Understanding Concurrent Transmissions over Ultra-Wideband Complex Channels. In *Proceedings of the 22nd International Conference on Embedded Networked Sensor Systems (SenSys)* (Hangzhou, China). ACM, New York, NY, USA, 757–770. https://doi.org/10.1145/3666025.3699372

[46] M. Schuß et al. 2019. JamLab-NG: Benchmarking Low-Power Wireless Protocols under Controllable and Repeatable Wi-Fi Interference. In *Proceedings of the 16th International Conference on Embedded Wireless Systems and Networks (EWSN)* (Beijing, China). Junction Publishing, 83–94. https://doi.org/10.5555/3324320.3324331

[47] M. Schuß, C.A. Boano, and K. Römer. 2018. Moving Beyond Competitions: Extending D-Cube to Seamlessly Benchmark Low-Power Wireless Systems. In *Proc. of the 1st CPSBench Workshop* (Porto, Portugal). IEEE, Piscataway, NJ, USA, 30–35. https://ieeexplore.ieee.org/document/8429499

[48] M. Schuß, C.A. Boano, M. Weber, and K. Römer. 2017. A Competition to Push the Dependability of Low-Power Wireless Protocols to the Edge. In *Proc. of the 14th EWSN Conference* (Uppsala, Sweden). Junction Publishing, USA, 54–65. https://dl.acm.org/doi/10.5555/3108009.3108018

[49] M. Spörk, C.A. Boano, M. Zimmerling, and K. Römer. 2017. BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices. In *Proceedings of the 15th International Conference on Embedded Networked Sensor Systems (SenSys)* (Delft, The Netherlands). ACM, New York, NY, USA, 2:1–2:14. https://dl.acm.org/doi/10.1145/3131672.3131687

[50] M. Spörk, J. Classen, C.A. Boano, M. Hollick, and K. Römer. 2020. Improving the Reliability of Bluetooth Low Energy Connections. In *Proc. of the 17th International Conference on Embedded Wireless Systems and Networks (EWSN)* (Lyon, France). Junction Publishing, USA, 144–155. https://dl.acm.org/doi/10.5555/3400306.3400324

[51] N. Srinivas, A. Krause, S.M. Kakade, and M.W. Seeger. 2012. Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting. *IEEE Transactions on Information Theory* 58, 5 (May 2012), 3250–3265. http://ieeexplore.ieee.org/document/6138914/

[52] G.V. Trunk. 1979. A Problem of Dimensionality: A Simple Example. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1, 3 (July 1979), 306–307. https://doi.org/10.1109/TPAMI.1979.4766926

[53] A. Varga and R. Hornig. 2008. An Overview of the OMNeT++ Simulation Environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTOOLS)*. ICST, Brussels, Belgium, 1–10. https://dl.acm.org/doi/10.5555/1416222.1416290

[54] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. 2012. *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. RFC 6550. IETF. https://www.rfc-editor.org/info/rfc6550

[55] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele. 2012. pTUNES: Runtime Parameter Adaptation for Low-power MAC Protocols. In *Proc. of the 11th IPSN Conference* (Beijing, China). IEEE, Piscataway, NJ, USA, 173–184. https://ieeexplore.ieee.org/document/6920955