Automatic Protocol Configuration for Dependable Internet of Things Applications

Felix Jonathan Oppermann*, Carlo Alberto Boano*, Marco Antonio Zúñiga†, and Kay Römer*
*Institute for Technical Informatics, Graz University of Technology, Austria
†Embedded System Group, TU Delft, The Netherlands
Email: {oppermann,cboano,roemer}@tugraz.at, m.zuniga@tudelft.nl

***Abstract*—To meet strict dependability requirements in hostile and highly-varying environments, IoT communication protocols need to be carefully tuned in relation to the expected environmental changes. However, this is difficult to attain, as every application has unique properties and requirements. Tuning communication protocols correctly requires indeed significant expertise as well as a clear understanding on how hardware and software components are affected by environmental changes.**

In this paper, we propose a novel framework to automate the parametrization of IoT communication protocols. The framework uses models of the environment as well as the employed hardware and protocols to predict the effects of environmental changes on network performance and to automatically select a configuration that meets user-specified dependability requirements.

We demonstrate how to use this framework to configure a state-of-the-art MAC protocol for an IoT application deployed in a challenging outdoor environment and evaluate its accuracy in predicting how environmental changes affect network performance. We further evaluate the performance with different optimization strategies and show that the average run-time necessary to find a solution is sufficiently low to enable the use of our system in a typical IoT design process.

***Index Terms*—Dependability, Environmental Impact, Communication Protocols, Optimization, Internet of Things, Wireless Sensor Networks.**

I. Introduction

An increasing number of Internet of Things (IoT) and wireless sensor network (WSN) systems has been installed in real-world settings during the last years [1]. These systems are becoming an integral part of our daily lives, as they are used in application areas such as civil infrastructure monitoring, home automation, smart cities, smart grid, and smart healthcare. Several of these application domains are safety-critical, and rely on the dependable and predictable operation of sensors and actuators that are wirelessly networked. For example, systems employed to monitor patients, to control traffic, and to inspect the structural health of buildings impose *strict dependability requirements* on communication performance, as their failure can have severe consequences.

Fulfilling these dependability requirements can be very difficult, as WSN and IoT systems are often deployed in hostile environments that significantly affect their performance. For instance, systems deployed outdoors are affected by temperature fluctuations and changing weather conditions [2], [3]. To cope with these challenges and meet strict dependability requirements also in hostile environments, *communication protocols need to be carefully tuned* in relation to the expected environmental changes [4]. This is, however, difficult to attain, as every application has unique properties and requirements, and there is no "one-size-fits-all" solution. Tuning communication protocols correctly can indeed be a tedious task that requires significant expertise, as well as a clear understanding of how the environment affects the hardware in use and the different communication protocols [5], [6]. Furthermore, even for experts in the field, it may be difficult to find the right trade-off that satisfies multiple requirements for the application at hand (e.g., achieving both a high reliability and a low energy consumption). Therefore, there is a need for a simpler configuration of IoT communication protocols that does not overwhelm the intended users of the technology.

In this paper, we propose a novel framework to support the deployment of IoT and WSN applications by *automating the configuration of communication protocols* such that specific dependability requirements can be met. The framework uses models of the environment as well as of the employed IoT hardware and communication protocols to predict the effect of environmental changes on network performance. Given a set of user requirements, these models are used in combination with mathematical optimization techniques to select a protocol configuration that provides the required performance.

We demonstrate how our framework can be used to find a suitable protocol configuration that meets user-defined dependability requirements using a building façade monitoring application as a case study. We show that our framework can help to predict and avoid the adverse effects of temperature variations on communication performance found in this type of application and evaluate its accuracy and performance. We further *evaluate the performance* of different optimization strategies within the framework and show that the average run-time necessary to find a solution is sufficiently low to enable the use of our system in a typical IoT design process.

The contributions of this paper are three-fold:

1) We present the architecture of a new framework automating the configuration of IoT communication protocols;
2) We demonstrate how to apply this framework to configure a state-of-the-art MAC protocol for an IoT application deployed in a challenging outdoor environment;
3) We evaluate the accuracy of the framework in predicting the behavior of the environment and its effect on network performance.

The remainder of the paper is structured as follows. The next section introduces an exemplary application that will serve as running example and as basis for the evaluation of our system. We illustrate our approach in Sect. III and detail on the architecture and implementation of our framework in Sect. IV. Thereafter, in Sect. V, we demonstrate how to use the framework to find a suitable configuration that meets user-defined dependability requirements. In Sect. VI we evaluate the overall performance of the framework. After describing related work in Sect. VII, we conclude our paper in Sect. VIII.

II. Case Study: Reliable Monitoring of Building Façades

A large number of IoT and WSN systems are deployed *outdoors* and are expected to operate dependably over extended periods of time, e.g., wildfire detection systems in forests [7] or wireless networks monitoring structural damage in civil infrastructures and buildings [8], [9]. Unfortunately, the performance of wireless systems deployed outdoors is typically affected by time-varying environmental conditions such as meteorological changes and variations in humidity [2], [3], which makes it difficult to satisfy strict dependability requirements. For example, large temperature variations can have a severe impact on network performance, as they reduce the efficiency of radio transceivers [4].

We have experienced these problems in the context of the RELYonIT project [10], during a pilot deployment of a WSN on the different façades of a building in Madrid, Spain. The purpose of our outdoor deployment is to promptly detect structural damage as well as to measure the energy efficiency of the construction by analyzing how the employed insulating materials reduce heat transfer. This type of application requires *a continuous reliable collection* of sensor data, such as temperature, humidity, and vibration. On the one hand, achieving a high packet delivery rate across the network is necessary to have a complete picture of the integrity of the building and to avoid severe issues, such as the detachment of loose parts from the building façade. On the other hand, to draw conclusions about the effectiveness of a constructing material or HVAC system, engineers rely on tiny changes in the measured variables, and any gap in the collected data may lead to false conclusions.

The major obstacle towards a reliable data collection is that nodes deployed on the building façades often experience high temperature fluctuations, especially if they are placed inside IR-transparent enclosures exposed to direct sun radiation. During our pilot deployment we have indeed observed daily temperature variations as high as $50\,^{\circ}$C.

These large temperature variations can have a severe impact on the operation of CSMA protocols, because they can reduce the effectiveness of clear channel assessment (CCA) methods and compromise the ability of a node to avoid collisions and to successfully wake-up from low-power mode [11]. Indeed, at high temperature the efficiency of low-power radio transceivers may reduce significantly: as a result, the signal strength between two wireless sensor nodes A and B decreases when the on-board temperature of one of the two nodes (or of both nodes) increases [4].

This problem is exacerbated by the fact that most state-of-the-art CSMA MAC protocols *rely on default system settings*, e.g., on the default CCA threshold of the employed radio device, hence neglecting the impact of the specific environmental properties of the target deployment site. As we show in Sect. V, protocols should instead be carefully parametrized in relation to the network configuration and to the properties of the environment. For example, an inaccurate selection of the CCA threshold may lead to a situation in which receiver nodes constantly remain in low-power mode at high temperatures, causing the disruption of links and a drastic reduction in network performance that may violate the dependability requirements of the application [11].

However, configuring protocols correctly is a very complex task. For example, to select the optimal CCA threshold for a MAC protocol employed outdoors, engineers need to consider the expected temperature variations at the deployment site, their impact on the hardware platform in use and on protocol operations, as well as the overall implications on a network level. This is not only time-consuming, but also non-trivial, given that the parameter value needs to be computed in relation to the specific application requirements defined by the user and to the actual placement of nodes.

To correctly predict the effects of environmental changes on network performance and select an optimal configuration of the system, it is hence highly desirable to employ a tool that automates the parametrization of communication protocols so that user-specified dependability requirements can be met. We describe next our attempt to build such an automatic framework by first illustrating the approach we have followed.

III. Approach

In order to fine-tune communication protocols such that user-specified requirements can be met, our framework needs a set of *user requirements* and a number of formal *models* to make reliable predictions about the system and its surrounding environment. The framework uses then mathematical *optimization techniques* to find a (near-)optimal configuration that meets the desired performance.

We distinguish between three categories of interacting *models*: environmental models, platform models, and protocol models.

1) *Environment models.* These models capture relevant aspects of the environment and provide an abstract representation thereof. Individual model instances are created for each specific environment by setting key parameters. The latter are determined by running a data collection application prior to the actual deployment. In our façade monitoring application example, we need to employ a model capturing the evolution of temperature in the target deployment site. Such a model can consist of the expected minimal and maximal temperatures for specific time intervals of the day (e.g., dawn, morning, noon, afternoon, evening, and night). To instantiate the model,

temperature is monitored over an extended time period prior to the deployment to compute minimal and maximal temperatures for each time interval.

2) *Platform models.* Different brands and types of sensor nodes react differently to specific environmental conditions. This relationship is captured by platform models. The latter provide a mapping of environmental parameters to variables that are relevant for the operation of WSN software. In our case study, temperature affects the operation of low-power radio transceivers. Consequently, our platform model needs to capture the relationship between the on-board temperature of sender and receiver nodes and the attenuation of the received signal strength for the employed hardware platform [4].
3) *Protocol models.* These models characterize the operation of a protocol under certain environmental conditions and with a predefined hardware configuration, and are hence built upon environmental and platform models. To be able to assess the effects of performance changes, these models need to expose all the relevant parameters of the protocol that may suffer from environmental impact. In our façade monitoring application case study, we use a network of nodes running ContikiMAC [12], a CSMA-based MAC protocol that is vulnerable to the impact of temperature variations on clear channel assessment. We therefore need to derive a model for ContikiMAC that estimates how different CCA settings affect the expected packet reception rate (PRR) as a function of the expected temperature variations and hardware platform employed.

These models alone already allow predictions of the performance based on a specific configuration. To automatically identify a (near-)optimal configuration that meets the user's requirements based on these predictions we employ *mathematical optimization*. Optimization strategies provide a way to systematically evaluate configurations such that a (near-)optimal solution can be found in a relatively short time.

The configuration process is steered by *user-defined dependability requirements*, as an optimal selection of a parameter strongly depends on the actual application needs. The definition of application requirements is complicated by the fact that some applications support different states of operation, often with significantly different requirements. For example, a system to detect wildfires in forests would be typically optimized for a long system lifetime during normal operation. However, as soon as a forest fire is detected, lifetime is not a primary concern anymore, and the primary goal becomes instead the fast dissemination of the fire-front direction. Consequently, it is necessary to allow the user to define multiple performance states and their associated sets of requirements. For each state, an individual configuration is created, and at run-time the application can select the configuration that best meets its current state.

IV. Parameter Selection Framework

We now describe the parameter selection framework implementing the automatic protocol configuration approach introduced in Sect. III. After giving a coarse-grained description of the software architecture of the parametrization component, we then detail on the optimization techniques employed by the framework and illustrate the actual software implementation.

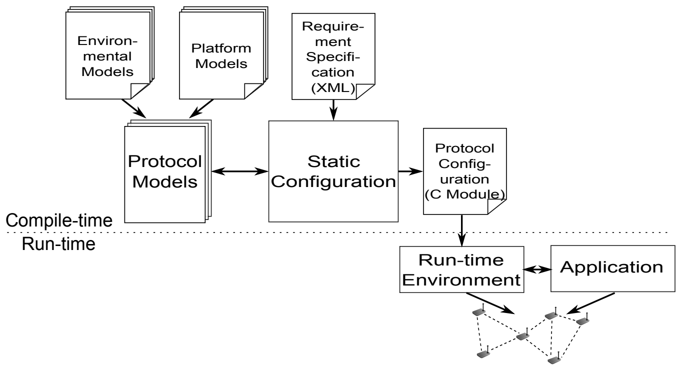

Fig. 1. Architecture of the parametrization framework.

A. Architecture

A bird's eye view of the architecture of the parametrization framework is presented in Fig. 1. The system architecture is organized around the *static configuration* component for protocol parametrization, which coordinates the automatic parameter selection process. It receives a user-provided *specification* of the dependability requirements as input.

The specification consists of a number of constraints on properties of the network behavior and a single property that should be either maximized or minimized (e.g., maximization of network lifetime). To increase the flexibility, our framework also supports probabilistic constraints that only need to hold at a specific point in time with a given probability. It is also possible to define more than one constraint for the same metric, typically with different probabilities. For example, a user could specify that the packet reception rate should stay above 0.8 with a probability of 0.9 and above 0.5 with a probability of 1. For applications that support different operation modes, the requirements for each mode are specified individually and, for each mode, an individual protocol configuration is derived.

Based on these inputs a near-optimal configuration for the respective protocol is generated by the static configuration tool, employing mathematical optimization techniques. If the requirement specification defines different modes of operation, the process is executed individually for the requirements of each mode.

The final output of the parametrization tool is a *protocol configuration* for each employed protocol. These configurations are static and do not change at run-time. Nevertheless, it is possible to switch between configurations associated to the different performance modes. In the following sections we will look at individual aspects of the framework in more detail.

B. Underlying Mathematical Model

The static configuration component employs mathematical optimization to generate an optimal parameter configuration.

We employ stochastic optimization strategies that are not able to give absolute guarantees, but are usually more robust to noisy data and require less fine-tuning for a specific problem instance. Due to their inherent randomness, stochastic strategies tend indeed to be able to escape local minima and to approach a global optimum even in a non-convex search space.

Each dependability specification essentially defines a constrained optimization problem. The formulation is based on a number of metrics $m_i(\mathbf{c})$ that allow to evaluate a specific configuration $\mathbf{c}$ based on a protocol model. A single protocol model may support more than one metric, each representing a specific relevant performance measure of the underlying protocol implementation. During the evaluation, the protocol model resorts to a suitable environment and platform model to incorporate the exact properties of the the environment and platform at hand. In the current system, we only consider the single objective case where a single metric is optimized. In addition to this optimization goal, the user may specify a number of constraints that further determine the properties of the desired solution. This results in optimization problems of the following form[1]:

$$\begin{array}{ll} \text{Minimize} & m_0(\mathbf{c}) \\ \text{Subject to} & m_1(\mathbf{c}) \le t_1 \text{with prob. } p_1 \\ & m_2(\mathbf{c}) \le t_2 \text{with prob. } p_2 \end{array} \tag{1}$$

The goal is to find a set of protocol configuration parameters $\mathbf{c}$ that optimize a single metric m_0. In addition, a variable number of constraints need to be fulfilled by ensuring that $m_1(\mathbf{c})$ and $m_2(\mathbf{c})$ are below the respective thresholds t_1 and t_2 with a probability of at least p_1 or p_2. Note that m_0, m_1, and m_2 may refer to the same metric.

To be suitable for automatic optimization, this optimization problem needs to be transformed into a suitable goal function, as most optimization strategies cannot immediately handle probabilistic constraints.

First, the constraints are integrated with the optimization function in an approach resembling penalty functions [13]. In the unconstrained case, the cost $f(\mathbf{c})$ of a specific configuration $\mathbf{c}$ is determined only by the goal function $m_0(\mathbf{c})$. To integrate constraints, a measure of violation is computed for each constraint and is added to the total cost of the current configuration. To ensure that invalid solutions are unlikely to be selected, while still allowing the optimization algorithm to traverse infeasible regions of the search space in order to reach more promising regions, the influence of the constraint violation is given more weight. If we assume a weight of k, the total cost for a given configuration $\mathbf{c}$ and r constraints can now be calculated as:

$$f(\mathbf{c}) = \frac{f_{\text{goal}}(\mathbf{c}) + k\sum_{j=1}^{r} f_{\text{cons},i}(\mathbf{c})}{1 + kr} \tag{2}$$

[1]To simplify the presentation and without loss of generality, we assume that the metrics (m_0, m_1, m_2) and the thresholds (t_1, t_2) are normalized to the $[0, 1]$ range and that smaller values denote superior properties. In addition, only minimization and less-or-equal constraints are considered, as other goals and constraints can be easily converted into this form.

where $f_{\text{goal}}(\mathbf{c}) = m_0(\mathbf{c})$ depends only on the evaluation of the goal and $f_{\text{cons},i}(\mathbf{c}) = \max\{(t_i - m_i(\mathbf{c})), 0\}$ is determined by the degree of violation for the i^{th} constraint.

Second, the non-standard feature of probabilistic constraints that is not supported by the employed optimization techniques needs to be handled. To support this, we exploit the fact that most environmental parameters exhibit a periodic behavior, e.g., a day and night cycle. This allows us to divide the period into a number of intervals with individual environmental properties. If we assume that communication events are evenly distributed over time, we can associate a probability q_j with each interval based on its relative length. This indicates how likely it is that a communication event is affected by the properties of this specific interval. Instead of a single function $m_i(\mathbf{c})$ per metric, we now employ a set of functions $m_{i,j}(\mathbf{c})$, each corresponding to one of the n intervals. Each of these functions uses a different instance of the environmental model that represents the distinct interval. To support probabilistic constraints, we now need to adapt the definition of the functions $f_{\text{goal}}(\mathbf{c})$ and $f_{\text{cons},i}(\mathbf{c})$ in Eq. 2. The cost of the goal $f_{\text{goal}}(\mathbf{c})$ is simply defined as the average of the cost for each individual interval:

$$f_{\text{goal}}(\mathbf{c}) = \sum_{j=1}^{n} q_j m_{0,j}(\mathbf{c}) \tag{3}$$

The calculation of the cost of the individual constraints now takes the probabilities in account by employing the definition:

$$f_{\text{cons},i}(\mathbf{c}) = \max\left\{\left(p_i - \sum_{j=1}^{n} \tau(m_{i,j}(\mathbf{c}), t_i, q_j)\right), 0\right\} \tag{4}$$

where n is the number of intervals and the function

$$\tau(v, t, q) = \begin{cases} q, & v > t \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

implements the aforementioned check for violation of the constraint. The resulting definition of $f(\mathbf{c})$ can now be employed as goal function of an unconstrained optimization problem that is well supported by the employed optimization techniques.

C. Implementation

The protocol parametrization tool is implemented as a standalone Python application. Its primary input is a user-provided requirement specification employing a custom XML-based specification language. This file contains an encoded specification of the user's requirements and defines an optimization problem as detailed in Sect. IV-B. If the application supports different states of operation, an independent requirements specification document is provided for eachstate.

In addition to the specification, the parametrization component has access to a collection of protocol model implementations. Available protocol model implementations are located via a search path and are automatically loaded by the framework as needed. To enable dynamic loading and the easy addition of additional models without the need to modify the framework itself, protocol models employ a plug-in interface.

This interface provides methods for initialization and model evaluation. The latter is used during the optimization process to evaluate the quality of individual protocol configurations. In addition, the interface allows to query which metrics are supported by the model and can be used in a related requirement specification. Most models only cover a subset of the available performance and reliability metrics. When evaluating a configuration, the protocol model implementations make use of platform and environmental model instances. For simpler models, their implementation is usually directly integrated with the implementation of the protocol models. For more complex platform and environmental models, they are implemented as separate modules which are accessed via method calls. Both environmental and platform models depend on application-specific empirical data that is usually loaded from a file at initialization. This interface is defined by an abstract `Model` class.

The protocol parameterization component can utilize different optimization strategies to solve the optimization problem, allowing the user to choose a strategy that is most appropriate for the specification and models at hand. The current prototype implements two stochastic optimization strategies. Stochastic optimization strategies cannot guarantee that an optimal solution is found in each run, but they are usually more robust to noisy data and require less fine tuning for a specific problem instance. Suitable stochastic optimization strategies still possess a high probability of convergence and are usually able to find a near-optimal solution. Due to their inherent randomness, stochastic strategies tend to be able to escape local minima and to approach a global optimum even in a non-convex search space. The current prototype supports simulated annealing and evolutions strategies. Both strategies use custom implementations that support configurations with integer, floating point, nominal, and Boolean values. The implementation of evolution strategies builds on ideas from Reehuis and Bäck [14].

The final output of the configuration tool is a protocol configuration encoded in a C source file. It contains individual configuration values for the configurable parameters of the involved protocols. The C representation is later compiled and linked with a run-time environment to enable the correct configuration of the communication protocols during operation. To give the protocol implementations access to the derived configuration values, a unified configuration interface is provided by a run-time environment, which enables them to receive their configuration parameters at initialization. In addition, to support different performance states for the application, the static optimization tool generates an individual parameter configuration for each performance state. The run-time environment provides methods for the user-application to switch between different performance states and to notify protocols of a state change.

V. Application of the Framework

We now demonstrate the applicability of our framework in a typical IoT application using the façade monitoring application introduced in Sect. II as a case study. In this scenario, nodes deployed on the building façades may experience high fluctuations of their on-board temperature. As we have discussed previously, these variations can have a severe impact on the operation of low-power CSMA MAC protocols such as ContikiMAC [12] due to the inefficiency of clear channel assessment at high temperatures. Indeed, the signal strength between two nodes A and B decreases in a linear fashion when the on-board temperature of one of the two nodes (or of both nodes) increases [4]. When low-power CSMA protocols perform an inexpensive clear channel assessment (CCA) check to determine if a node should remain awake to receive a packet or whether it should return to sleep mode, they essentially compare the current received signal strength with a CCA threshold ζ. The latter is typically chosen at compile-time and often set to the default value of the employed radio device (e.g., -77 dBm for the off-the-shelf CC2420 radio transceiver). When temperature increases, the received signal strength of a node may decrease to a point in which it becomes lower than ζ. When this happens, the receiver node remains constantly in low-power mode, causing disruption of the link [11]. Still, ζ should not be set to an arbitrarily low value, as this may lead to an increased number of false wake-ups due to interference and noise in the surroundings that would significantly increase the energy expenditure. To avoid the issues, we need to properly configure ζ such that any potential increase in the on-board temperature of the deployed nodes will not lead to a loss of connectivity, i.e., we need to find a suitable configuration of ζ such that the network can sustain the desired PRR despite temperature changes while still minimizing energy expenditure. Towards this goal, we need to (i) derive the necessary models, (ii) select the application requirements, and (iii) integrate them into the framework.

A. Deriving the Models

Our framework requires three different types of models: environmental, platform, and protocol models. The *protocol model* will describe how the operations of the employed MAC protocol, ContikiMAC, are affected by temperature changes. More specifically the protocol model captures the effect on the performance of ContikiMAC in terms of PRR. This model will build upon a *platform model* characterizing the signal strength attenuation of the radio transceiver embedded in the employed sensor nodes, as well as upon an *environmental model* capturing the possible variations of on-board temperatures in the specific deployment environment.

a) Environmental model: Our façade monitoring application is deployed in the city of Madrid, Spain, and we hence need to capture how its Mediterranean climate can affect the on-board temperature of sensor nodes. We devise an environmental model based on the maximum on-board temperatures recorded on the sensor nodes at specific times of the day. In particular, we sub-divide each day into six intervals of equal length, and run a specific data collection application prior deployment that records the on-board temperature variations over several days [10]. The model thus provides the maximal temperature as a function of the time of the day.

b) Platform model: In our deployment we employ MTM-CM5000-MSP motes embedding a CC2420 radio transceiver. In earlier research, we have shown the effects of temperature on the efficiency of this transceiver, and derived a linear model characterizing the decrease in signal strength as a function of temperature [4]. Denoting PL as the path loss between a transmitter-receiver pair, P_{t} as the transmission power, $P_{\mathrm{r}} = P_t - PL$ as the received power, and P_{n} as the noise floor at the receiver, this model describes the temperature effect on the SNR as follows:

$$\begin{aligned} SNR &= (P_{\mathrm{t}} - \alpha\Delta T_{\mathrm{t}}) - (PL + \beta\Delta T_{\mathrm{r}}) \\ &\quad -(P_{\mathrm{n}} - \gamma\Delta T_{\mathrm{r}} + 10\log_{10}(1 + \tfrac{\Delta T_{\mathrm{r}}}{T_{\mathrm{r}}})) \\ &= (P_{\mathrm{r}} - \alpha\Delta T_{\mathrm{t}} - \beta\Delta T_{\mathrm{r}}) \\ &\quad -(P_{\mathrm{n}} - \gamma\Delta T_{\mathrm{r}} + 10\log_{10}(1 + \tfrac{\Delta T_{\mathrm{r}}}{T_{\mathrm{r}}})) \end{aligned} \tag{6}$$

where the constants α, β, and γ with units dB/K denote respectively the effect on transmitted power, received power, and on the noise floor. These values are obtained by regression over data obtained from testbed experiments. The values T_{t} and T_{r} represent the reference temperature in Kelvin of transmitter and receiver; whereas ΔT_{t} and ΔT_{r} capture the difference of current temperature in Kelvin with respect to T_{t} and T_{r} [4].

c) Protocol model: The reception of a packet in CSMA-based protocols such as ContikiMAC can be estimated by analyzing how the signal strength with which the packet is received relates to the transitional phase of the radio response and to the selected CCA threshold.

Each node periodically wakes up from low-power mode and checks for incoming packets by verifying if the received signal strength s_{r} is above a fixed CCA threshold ζ. The PRR is hence firstly influenced by the relationship between s_{r} and ζ: if $s_{\mathrm{r}} \geq \zeta$, the node infers that an ongoing transmission is present and remains awake to receive the packet; if $s_{\mathrm{r}} < \zeta$, the node believes that there is no ongoing transmission and returns to sleep mode without receiving the packet.

Furthermore, PRR is also affected by the transitional phase of the radio response. When the signal strength of the received packet is too close to the sensitivity threshold of the employed radio, it becomes unlikely to successfully demodulate the packet. Early WSN research has shown that the decrease of PRR in the transitional region follows a sigmoid curve [15], in which the probability p of receiving a packet is

$$p = (1 - f(P_{\mathrm{r}} - P_{\mathrm{n}}))^b \tag{7}$$

with P_{r} being the received signal strength in dBm, P_{n} the sensitivity threshold of the radio in dBm, and b the number of bits in the packet.

Denoting $s_{0.99}$ as the signal strength that leads to a delivery rate of 0.99, and $s_{0.01}$ as the corresponding signal strength for a delivery rate of 0.01, we can define three reception regions, as shown in Fig. 2 in the black sigmoid curve: a connected region, where the received signal strength is above $s_{0.99}$; a disconnected region, where the signal strength is below $s_{0.01}$,

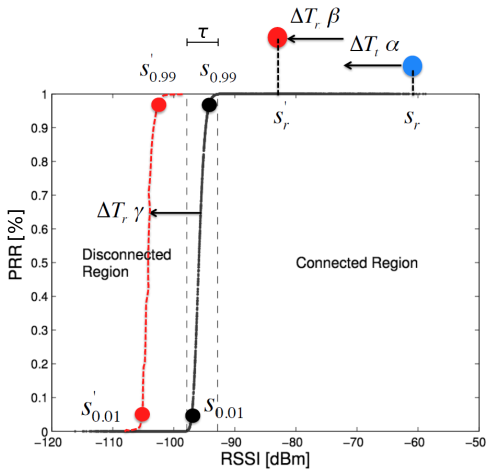

Fig. 2. The impact of temperature on the operation of a MAC protocol.

and a transitional region of length τ dB, where the delivery rate drops monotonically between 1 and 0.

Because of the dependency between signal strength and temperature, according to Eq. 6, a variation in the on-board temperature at the receiver or at the transmitter will cause the receiver to measure a signal strength

$$s_{\mathrm{r}}^{'} = (s_{\mathrm{r}} - \alpha\Delta T_{\mathrm{t}} - \beta\Delta T_{\mathrm{r}}) \tag{8}$$

i.e., an increase (decrease) in T_{t} and/or T_{r} will attenuate (strengthen) s_{r} into $s_{\mathrm{r}}^{'}$. Fig. 2 shows an example in which the received signal strength s_{r} decreases (i.e., is shifted to the left) due to an increase of temperature in both transmitter ($\alpha\Delta T_{\mathrm{t}}$ component) and receiver ($\beta\Delta T_{\mathrm{r}}$ component). To predict if a change in the on-board temperature at the transmitter and/or receiver node will affect packet reception, we need to verify if $s_{\mathrm{r}}^{'} < \zeta$. If this is the case, no packet will be received, as the node will return to sleep mode after having assumed no ongoing transmission.

Similarly, if the on-board temperature of the receiver changes, also the position of the sigmoid curve may change. For example, an increase in temperature would lower the noise floor (see Eq. 6), shifting this curve towards left (red sigmoid curve). To predict how $s_{0.01}$ and $s_{0.99}$ would change in relation to temperature variation we use Eq. 6 to derive $s'_{0.99} = s_{0.99} - \Delta T_{\mathrm{r}}\gamma$ and $s'_{0.01} = s_{0.01} - \Delta T_{\mathrm{t}}\gamma$.

We can hence estimate the packet delivery rate PRR' given a specific ζ value for each link i in the network as:

$$PRR' = \begin{cases} 1, & \text{if } \max\{\zeta, s'_{0.99}\} < s'_{\mathrm{r}} \\ p, & \text{if } s'_{0.01} \leq \zeta < s'_{\mathrm{r}} \leq s'_{0.99} \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

This allows us to estimate the worst case delivery rate given a specific temperature variation/range.

B. Integration with the Framework

The aforementioned model has been realized as a plug-in implementing the interface defined by the framework. The

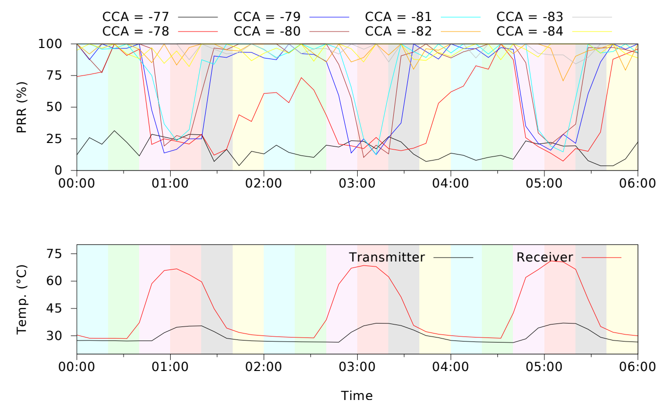

Fig. 3. Impact of CCA threshold on PRR for a single link. Background colors indicate different time intervals.

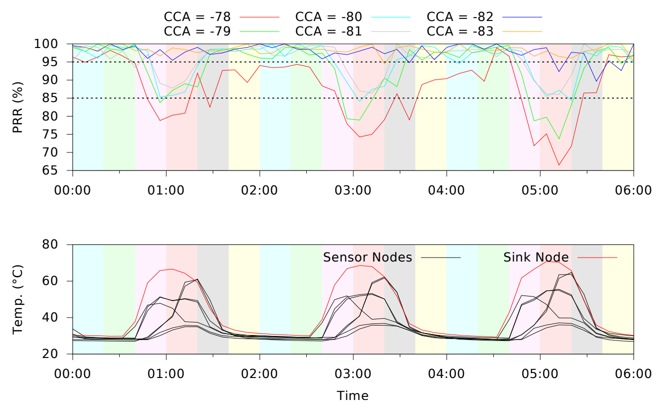

Fig. 4. Impact of CCA threshold on PRR on a network of nodes. The bottom plot shows the different temperature profiles of the nodes in the network.

implementation exposes one configuration parameter, ζ, and provides a number of metrics that can be used to define goals or constraints. The primary metric is the expected worst case PRR of a link averaged over all links in the network.

C. Selecting the Dependability Requirements

As discussed in Sect. II, the target application needs a highly reliable data collection. Therefore, the MAC layer needs to ensure a high PRR even under adverse environmental conditions. At the same time, it is also important to ensure that the system will have a long lifetime in order to keep maintenance at a manageable level. Hence, we need to find a CCA threshold setting that still ensures that the user requirements are met but leads to as little energy wastage as possible. Consequently, the CCA should be maximized while still not violating the constraints, as lower CCA values lead to a higher number of false wake-ups.

An industry partner running the deployment specified, that to enable useful insights about a building's energy-efficiency, at least 85% of the collected measurements should be successfully delivered to the sink at all times and with a probability of 0.9 at least 95 % of the packets should arrive at their destination.

Based on these requirements and the implemented models it is now possible to determine a protocol configuration that is able to support the expected performance. These requirements lead to the following optimization problem:

$$\begin{aligned} &\text{Maximize} && \mathrm{CCA}([\zeta]) \\ &\text{Subject to} && \mathrm{PPR}([\zeta]) \geq 0.85 \text{ with probability } 1.00 \\ & && \mathrm{PPR}([\zeta]) \geq 0.95 \text{ with probability } 0.9 \end{aligned} \quad (10)$$

VI. Evaluation

In this section we evaluate the performance of the proposed framework.

A. Suitability of the Framework

To evaluate the applicability of the framework, we employ it to generate a suitable configuration for the case-study introduced in Sect. II. The goal is to find an optimal CCA threshold value that conserves energy while still fulfilling the user's dependability requirements. To ease the analysis and presentation of the results, we first consider a single link between two nodes. In a second step, we demonstrate that the same principle correctly works on a network of nodes.

We use TempLab [16], a temperature-controlled testbed, to recreate a network scenario similar to the one found on real-world outdoor façades. In particular, we feed TempLab with temperature traces previously recorded in Madrid so that the on-board temperature of the sensor nodes in the testbed experiences the same profile as in the real-world. In our experiments we use a time-lapse factor of 12, so that a day is replayed within 2 hours in our testbed. We further set the width of the transitional region $\tau = 5\,\text{dB}$ and the packet size $b = 35$ bytes. The remaining model parameter were left at their default values of $\alpha = 0.078\text{dB/K}$, $\beta = 0.078\text{dB/K}$, and $\gamma = 0.037\text{dB/K}$. These values have been empirically determined by earlier experiments [4].

In the first set of experiments we employ the framework to configure the CCA threshold to obtain dependable communication on a *single link*. In particular, we aim to find a configuration that maximizes the CCA threshold while maintaining a PRR between the two nodes of 0.85 at all times. Based on temperature traces from the deployment in Madrid, the configuration framework determines an optimal CCA threshold of -83 dBm for this requirement specification. When trying different possible CCA threshold settings in the TempLab testbed, as shown in Fig. 3, we can see that with a threshold of -83 dBm, the PRR actually stays above 0.85 for the whole duration of the experiment, whilst with a higher CCA threshold this would not be the case.

In a second experiment, we employ the same PRR constraint of 0.85 but only with a probability of 0.6. For this scenario, the framework determines an optimal CCA threshold of -81 dBm. As shown in Fig. 3, the use of a threshold of -81 dBm actually ensures that the PRR stays above 0.85 in four out of the six intervals per day, which fulfills our requirement of sustaining a PRR of at least 0.85 in at least 60% of the cases.

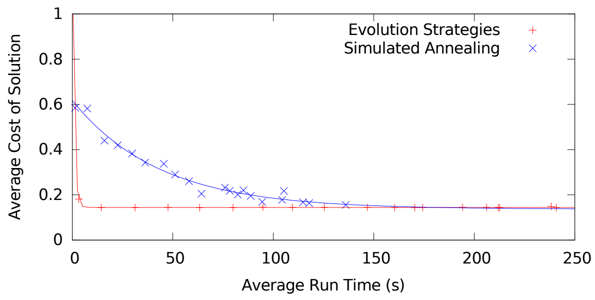

Fig. 5. Time/quality trade off for different optimization strategies.

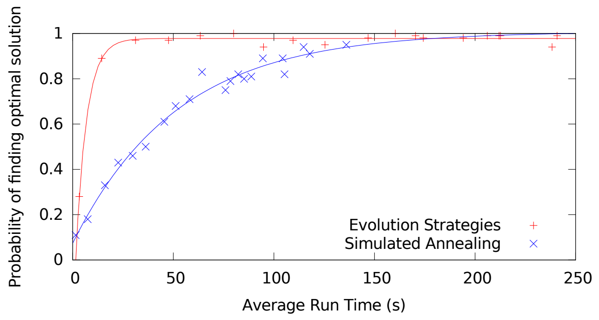

Fig. 6. Probability of finding the optimal solution within a given time.

We can also see that a CCA threshold of -80 dBm would violate the constraint by also dropping below 0.85 in the first interval, hence -81 dBm can be actually assumed to be the most energy conserving configuration that is able to fulfill the constraint.

We now use our framework to configure a network of seven sensor nodes connected to a single gateway node in a star topology. All nodes are exposed to different temperature profiles following the ones recorded on different building façades in Madrid. To configure the network, we employ the user-defined requirements introduced in Sect. V. For this scenario, the framework suggests the use of a CCA threshold of -83 dBm. Based on the results reported in Fig. 4, it can be seen that for the selected CCA threshold, the average PRR stays above 0.85 at all times. For CCA thresholds above -80 dBm, this constraint is clearly violated. Nevertheless, with a threshold of -82 dBm or higher, the PRR drops below 0.95 for at least three out of the 18 intervals, which indicates that the second constraint cannot be met. Consequently, the tool actually picked the best possible CCA value for the given scenario. This demonstrates that our tool is capable to generate useful configurations for realistic deployment scenarios within the selected application area and is able to handle the more complex requirements of typical users.

B. Performance

We now evaluate the basic performance of the system and measure the relative performance of the different optimization algorithms. Employing the same model and similar settings as in the previous section, a globally optimal solution can be found with a CCA threshold of -84 dBm and a cost value $f = 0.14344$.

For the evaluation, we execute the parametrization process with both available optimization strategies. To evaluate different time/quality trade-offs, we artificially limit the maximal number of iterations. To reduce the effect of random events, each algorithm is executed 100 times for each setting.

Fig. 5 presents the trade-off between the runtime of the optimization algorithms and the average quality of the solutions found. To illustrate the observed trend, an exponential function ($x \mapsto a * e^{b*x} + c$) has been fitted to the raw data. It can be seen that with very short run times, the optimal solution is only found in a limited number of runs and the returned solutions are often far away from the optimal one. With a run-time of two minutes or more, both algorithms are very likely to identify the optimal configuration. Incorrect solutions tend to be close to the optimal solution, as shown in Fig. 6. In most applications, finding a good solution that satisfies all constraints is already sufficient. Due to a rather small number of nodes and a limited set of configuration options in this case-study, exhaustive search is also still a feasible option, but its performance degrades quickly if the number of possible configurations is increased.

Both optimization algorithms performed well in this scenario and a run-time in the order of minutes coupled with a high reliability enable an efficient use within a typical WSN development process. For the given model, evolutionary strategies exhibit a slightly superior performance, but both strategies are able to generate suitable configurations within reasonable time.

C. Extensibility

Even though the presented evaluation is limited to a single use case and protocol, the framework can be applied in several other scenarios. As part of the RELYonIT project [10], we have indeed implemented protocol models for a number of additional protocols:

1) A model for TempMAC, a temperature-aware extension of ContikiMAC [11]. This model is essentially an extension of the one introduced in Sect. V;
2) A model capturing the impact of duty cycle configuration on the energy expenditure of a MAC protocol. This model can be used to find the best duty cycle for a specific radio environment and therefore especially targets indoor deployments where radio interference tends to significantly affect network operation;
3) A model to aid the configuration of the jamming-based agreement (JAG) protocol [17]. This model can be used to determine the jamming period length that yields an optimal agreement probability in environments prone to external radio interference;
4) A model to aid the selection of an optimal packet length to minimize energy consumption and reduce latency.

Within RELYonIT the framework has also been applied to determine the configuration of WSN-based smart parking systems in densely populated urban areas where both temperature and radio interference influence the network operation.

VII. Related Work

Support systems for WSN configuration are a surprisingly rarely-considered aspect of deployment support. Few systems exist that support users with the complex task of finding optimal configuration parameters for a given environment.

Existing approaches often rely on simulation [18], [19] and systematically try out different possible configurations in an emulated environment. With existing simulation environments, this is usually a time-consuming task as the high-fidelity models require significant processing power and consequently only allow limited speed-up for larger networks. The long computation times significantly limit the number of configurations that can be evaluated and easily lead to sub-optimal configurations. While our approach shares the same basic strategy, we can significantly reduce the run-time of the model evaluation by using more abstract formal models. This allows to evaluate a larger number of possible configurations and thus increases the likelihood of finding an optimal configuration.

Only a very small number of works apply formal models and mathematical optimization to WSN protocol configuration. A well-known example is the ptunes system developed by Zimmerling et al. [20]. Ptunes employs a formal protocol model and constraint programming to find optimal MAC protocol configurations settings for a specific network topology and radio environment. Ptunes' goals are very similar to our approach, but at least in its current form, ptunes is limited to MAC protocol configuration, while our approach targets protocols at different levels of the network stack. More importantly, ptunes does not explicitly model any environmental effects and only considers internal interference. Instead, ptunes is intended to work online and constantly reconfigure the network, which allows to constantly adapt the configuration to a changing environment, but significantly limits the available run-time for optimization. Our approach of pre-deployment configuration can use more sophisticated models that require a higher run-time, but leads to more precise and dependable results.

Our work builds on simulated annealing and evolution strategies to implement the actual optimization. While our implementation of simulated annealing follows common design strategies found in relevant textbooks, our implementation of evolution strategies employs ideas from Reehuis and Bäck [14] to support integer and floating point parameters.

VIII. Conclusion

In this paper, we introduced a novel framework for the automatic configuration of IoT communication protocols based on different models and user requirements. Our experimental evaluation demonstrates the feasibility of our approach for an exemplary application and an adequate performance of the current prototype. In the future we intend to implement additional protocol models and employ the framework in additional case studies. This will allow us to further assess the performance of the system and to identify means to increase the flexibility of the framework. Ultimately, we intend to enable the configuration of typical WSN and IoT software-stacks without requiring extensive expertise in the area.

Acknowledgment

The research leading to these results has received funding from the European Union 7^{th} Framework Programme (FP7-ICT-2011-8) under grant agreement n° 317826 (RELYonIT).

References

[1] F. J. Oppermann, C. A. Boano, and K. Römer, "A decade of wireless sensing applications: Survey and taxonomy," in *The Art of Wireless Sensor Networks*. Springer, 2014.

[2] C. A. Boano, J. Brown, N. Tsiftes, U. Roedig, and T. Voigt, "The impact of temperature on outdoor industrial sensornet applications," *IEEE Trans. Industr. Inform.*, vol. 6, no. 3, 2010.

[3] H. Wennerström, F. Hermans, O. Rensfelt, C. Rohner, and L.-Å. Nordén, "A long-term study of correlations between meteorological conditions and 802.15.4 link performance," in *Proc. of the* 10^{th} *IEEE SECON Conf.*, 2013.

[4] C. A. Boano et al., "Hot Packets: A systematic evaluation of the effect of temperature on low power wireless transceivers," in *Proc. of the* 5^{th} *ExtremeCom Conf.*, 2013.

[5] K. G. Langendoen, A. Baggio, and O. W. Visser, "Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture," in *Proc. of the* 14^{th} *WPDRTS Workshop*, 2006.

[6] N. Finne, J. Eriksson, A. Dunkels, and T. Voigt, "Experiences from two sensor network deployments – self-monitoring and self-configuration keys to success," in *Proc. of the* 6^{th} *WWIC Conf.*, 2008.

[7] D. M. Doolin and N. Sitar, "Wireless sensors for wildfire monitoring," in *Proc. of the SPIE Symposium*, 2005.

[8] S. Kim, S. Pakzad, D. E. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Health monitoring of civil infrastructures using wireless sensor networks," in *Proc. of the* 6^{th} *IPSN Conf.*, 2007.

[9] M. Ceriotti et al., "Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment," in *Proc. of the* 8^{th} *IPSN Conf.*, 2009.

[10] RELYonIT Consortium, "RELYonIT: Research by Experimentation for Dependability on the Internet of Things," 2015. [Online]. Available: http://www.relyonit.eu

[11] C. A. Boano, K. Römer, and N. Tsiftes, "Mitigating the adverse effects of temperature on low-power wireless protocols," in *Proc. of the* 11^{th} *MASS Conf.*, 2014.

[12] A. Dunkels, "The ContikiMAC radio duty cycling protocol," Swedish Institute of Computer Science, Tech. Rep. T2011:13, 2011.

[13] A. E. Smith and D. W. Coit, "Penalty functions," in *Handbook of Evolutionary Computation*. IOP Publishing Ltd., 1997.

[14] E. Reehuis and T. Bäck, "Mixed-integer evolution strategy using multi-objective selection applied to warehouse design optimization," in *Proc. of the 12th GECCO Conf.*, 2010.

[15] M. A. Zúñiga and B. Krishnamachari, "Analyzing the transitional region in low-power wireless links," in *Proc. of the* 1^{st} *SECON Conf.*, 2004.

[16] C. A. Boano, M. Zúñiga, J. Brown, U. Roedig, C. Keppitiyagama, and K. Römer, "TempLab: A testbed infrastructure to study the impact of temperature on wireless sensor networks," in *Proc. of the* 13^{th} *IPSN Conf.*, 2014.

[17] C. A. Boano, M. A. Zúñiga, K. Römer, and T. Voigt, "JAG: Reliable and predictable wireless agreement under external radio interference," in *Proc. of the* 33^{rd} *IEEE RTSS Symp.*, 2012.

[18] G. Simon, P. Volgyesi, M. Maróti, and A. Lédeczi, "Simulation-based optimization of communication protocols for large-scale wireless sensor networks," in *Proc. of the IEEE Aerospace Conf.*, 2003.

[19] M. Strübe, F. Lukas, B. Li, and R. Kapitza, "DrySim: Simulation-aided deployment-specific tailoring of mote-class WSN software," in *Proc. of the 17th ACM MSWiM Conf.*, 2014.

[20] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele, "pTunes: Runtime parameter adaptation for low-power MAC protocols," in *Proc. of the* 11^{th} *IPSN Conf.*, 2012.