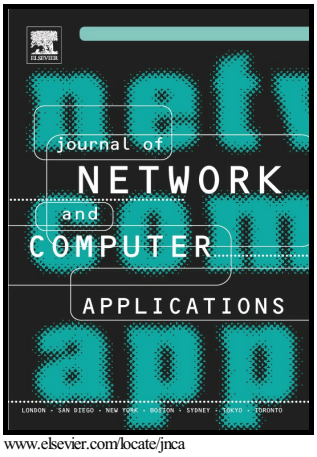

www.elsevier.com/locate/jnca

Optimization of Non-functional Properties in Internet of Things Applications

Xuan Thang Nguyen, Huu Tam Tran, Harun Baraki, Kurt Geihs

PII: S1084-8045(17)30128-5
DOI: http://dx.doi.org/10.1016/j.jnca.2017.03.019
Reference: YJNCA1891

To appear in: *Journal of Network and Computer Applications*

Received date: 15 September 2016
Revised date: 19 March 2017
Accepted date: 21 March 2017

Cite this article as: Xuan Thang Nguyen, Huu Tam Tran, Harun Baraki and Kur Geihs, Optimization of Non-functional Properties in Internet of Thing Applications, *Journal of Network and Computer Applications* http://dx.doi.org/10.1016/j.jnca.2017.03.019

Optimization of Non-functional Properties in Internet of Things Applications

Xuan Thang Nguyen[1], Huu Tam Tran[2], Harun Baraki[2], and Kurt Geihs[2]

[1]*Faculty of Information Technology, Hanoi University, Hanoi, Vietnam*

[2]*Distributed Systems Group, University of Kassel, Kassel, Germany*

Abstract

A major challenge in designing Internet of Things (IoT) systems is to meet various non-functional requirements such as lifetime, reliability, throughput, delay, and so forth. Furthermore, IoT systems tend to have competing requirements, which exacerbate these design challenges. We analyze this problem in detail and propose a model-driven approach to optimize an IoT application regarding to its non-functional requirements. Our approach defines optimizing as finding the best set of adjustable application parameters, which satisfies a given objective function. The relevant parameters are extracted during a simulation process. We apply a source code transformation that updates the source code with the generated adjustable parameter values and executes the compiler to create a new binary image of the application. Our experiment results demonstrate that non-functional requirements such as power consumption and reliability can be improved substantially during the optimization process.

Keywords: `Internet of Things, Sensor Networks, Optimization, Non-functional requirements, Simulation`

1. Introduction

In general, software requirements are partitioned into functional requirements and *non-functional requirements*. The functional requirements are associated with specific functions, tasks, features or behaviors that must be supported by the system, whereas the non-functional requirements are constraints

on various attributes of these functions or tasks. The non-functional requirements tend to be described in terms of *constraints* and qualities on the results of tasks that are given as functional requirements, for example, constraints on the speed or efficiency of a given task.

In the context of IoT software development, non-functional requirements such as the limited capacity of the terminal devices or the quality of service (QoS) play an important role and should be taken into account from the beginning of the development cycle [1, 2, 3]. Optimizing IoT software to meet non-functional requirements is an emerging trend and is receiving the attention of many researchers[1, 2, 4, 5]. However, the process of optimization is still time-consuming and increases development costs significantly. Developers have to take into account the resource limitations on their devices, the ad-hoc communication, the topology of the network, and the deployment environment's unpredictability [3, 6, 7, 8].

In those existing studies on IoT software optimization, we find that there are several issues that need further discussion. Firstly, the majority of studies aim at optimizing applications in the system deployment stage, i.e. after the design and programming phase [9]. However, considering non-functional requirements already at the design stage will simplify the optimization process and can relieve developers significantly. Secondly, parameters are usually selected relatively subjective according to the best knowledge of the authors. Most research is focusing on the optimization algorithms themselves [1]. There is a lack of research that determines quantitative parameters of the application and their impact on the non-functional requirements of IoT applications. Our work especially emphasizes this aspect. Thirdly, non-functional requirements mentioned in the aforementioned studies [1, 9] merely examine a few facets of an application. Only some non-functional requirements were put under consideration such as the network lifetime, transmission latency or packet loss ratio. While optimizing the application in terms of non-functional requirements, we need to pay attention to both qualitative requirements (security, availability, reliability, performance, etc.) and quantitative requirements (network lifetime, precision,

etc.). Moreover, non-functional requirements may contradict each other (e.g. battery usage vs. performance and reliability). Thus, the optimization is expected to provide information that supports programmers in choosing the most appropriate plan to meet several non-functional requirements at a time.

In this paper, we address the problem of IoT software optimization in order to meet non-functional requirements in the application design phase. First, we come up with the formulation that describes non-functional requirements using one or more *non-functional parameters*. These parameters can be quantified to allow a comparison of their satisfaction level towards the non-functional requirements in IoT applications. At the same time, we propose a set of adjustable parameters of IoT applications based on their level of satisfaction with regards to the non-functional requirements of the application. These parameters allow the application to be customized at different levels, for instance, at the hardware level (processor voltage and frequency), the software level (sensing frequency, duty cycle), or the network level (channel access schedule, message size, buffer size, and receiver power-off cycle). Then the optimization problem is narrowed down to a multi-purpose combination optimization problem with the search space being the values of the adjustable application parameters and the objective function is the minimum or maximum value of the primary non-functional parameters.

Because of the complex interrelationships between the parameters, the large number of their possible values and the measurable responses, and due to the existence of multiple objectives, we apply a Multi-Objective Evolutionary Algorithm (MOEA) [10] which is suited to a large number of input factors and is able to mutate all factors at each generation. MOEA can be considered a narrow, but deep heuristic search method which explores the search space in an uneven manner.

To evaluate the proposed solution, we investigate the design and implementation of an IoT application named *TempSense* where a wireless sensor network is used to monitor the temperature within an area. We identify a set of adjustable input parameters, non-functional constraints, and objectives of the

optimization problem. The experiment results demonstrate that non-functional constraints such as power consumption and reliability can be improved during the optimization process by selecting a proper set of adjustable parameters.

The main contribution of this paper is to present an approach to assist developers in optimizing IoT applications regarding non-functional requirements at different design levels, such as hardware, software, or communication protocol, by adjusting suitable parameters. The optimization component is part of our model-driven development (MDD) framework that enables programmers to implement IoT applications through a graphical user interface and a rule-based programming language [11]. The interaction between the optimization component and the MDD framework will be presented in this paper too. The overall objective is to support software developers during the *design phase* in implementing and optimizing their application. This paper will focus on the optimization component and extends our previous work presented in [11, 12, 13].

Another contribution of this work is an examination of the relationships that exist between the adjustable parameters and the non-functional requirements. To this end, we set up different experiment scenarios in which only a single adjustable parameter is changed and measure the impact on the non-functional requirements. The results clearly prove that non-functional requirements may oppose each other when trying to optimize them. This results will also help to quantify the satisfaction level of non-functional requirements with respect to specific software. The achievements of our research are not only significant in the development of IoT application software but can also be used in many different sectors of the software industry.

The rest of the paper is organized as follows. Section 2 analyzes the effects of some adjustable parameters, such as rebroadcast probability or sleep time on the applications' non-functional requirements. In Section 3 we present the formulation of the optimization problem and apply a MOEA to solve the optimization problem. The MDD approach applied in our framework is shown in Section 4. Section 5 describes the *TempSense* application and discusses the optimization results. Section 6 reviews the related work on optimizing non-

functional constraints in IoT systems. Finally, concluding remarks are given in Section 7.

2. Optimization Problem Analysis

We consider to the large class of IoT applications which employed a wireless sensor network (WSN) to provide IoT services. In order to figure out an appropriate approach to solve the optimization problem, we analyze relationships that exist between the adjustable parameters and the non-functional requirements of the applications. We set up different scenarios where in each scenario only a single adjustable parameter was changed. The experiments are done by running the application in the Cooja simulator [14]. Cooja enables simultaneous simulations at different levels like the network level, operating system level and machine code instruction level. After each simulation we collected data on three non-functional property metrics: the *energy consumption per delivered packet*, the *latency per delivered packet* and the *packet loss ratio*. We used a simulated network with 200 randomly distributed nodes in an area of 500 x 500 meters to test the applications. Each simulation was configured to run for three rounds and the simulation length for each round was 300 seconds. After each simulation, the average values of all observed metrics were computed and stored. Because the metrics were measured in different units and scales, we normalized all values such that for each metric, the measured value range $[min, max]$ was mapped onto the range $[0, 1]$.

2.1. Impact of the Rebroadcast Probability

The first experiment conducted was the investigation of the gossip protocol [15], a popular data broadcasting protocol. While pure flooding produces many redundant packets when broadcasting, the gossip protocol makes use of a probabilistic rebroadcast scheme to address this problem. After receiving a packet, a node continues broadcasting this packet with a given probability p or drops the packet with a probability of $1 - p$.

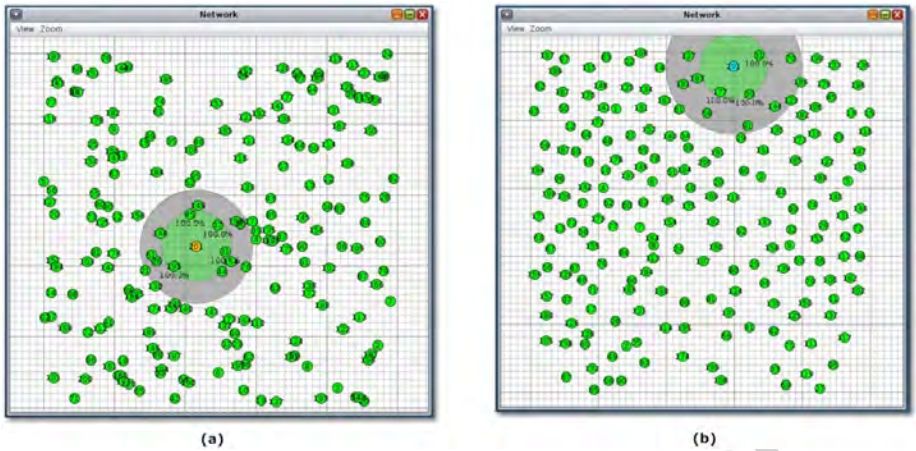

Figure 1: Two different experiment scenarios

Typically, choosing a correct value for p is an important step when designing a gossip-based routing protocol. To evaluate the impact of p on the non-functional parameters of the application, ten experiments are conducted with ten different p values, distributed evenly in the interval $[0, 1]$. Figure 1(a) shows the first experiment scenario, where each node has a transmission range and an interference range represented as a green circle and as a gray circle respectively. When a node transmits packets, only nodes in the green area are able to receive the packets. Nodes in the gray area can not receive the packets and they are also interfered which means that they are not able to receive the packets sent from other nodes when the selected node communicates simultaneously. The simulated network has a source node that frequently sends packets to its neighbors. Other nodes are implemented with the simple gossip routing protocol to continue retransmitting the packets. The radius of the transmission range and the interference range of each node in the simulation are 50 meters and 80 meters respectively.

Figure 2 shows the measurement values of the three non-functional parameters mentioned above corresponding to various values of the rebroadcast probability p. Figures 2(a) and (b) show that the packet loss rate generally decreases when p is increased. It means the network will become more reliable with bigger

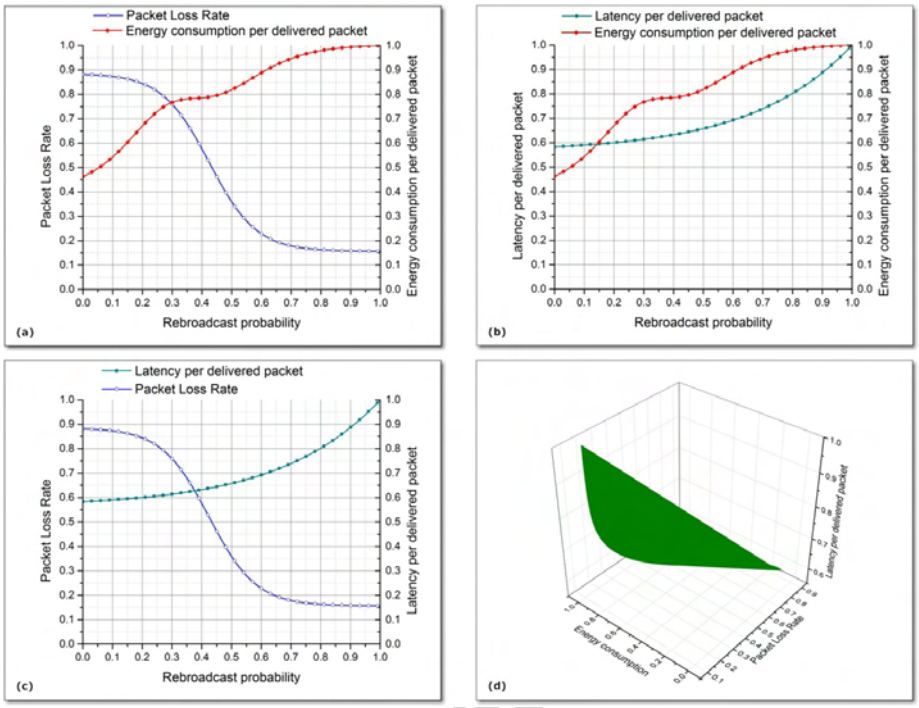

Figure 2: The impacts of rebroadcast probability on non-functional requirements

p values. Furthermore, we observe the bimodal behavior [15] of a gossip-based network where either most nodes receive a packet, or only a few do. We notice that the average packet loss rate does not reach 0, even when p is 1 because of interference. As the network activity increases with p, interference also occurs more frequently causing receiving nodes within the interference range to drop packets.

The changes of p also affect the average end-to-end latency of a gossip-based network, as shown in figures 2(b) and (c). As p increases from 0 to 1, the end-to-end delay also increases slowly due to the interferences between the nodes and their limited queue size. The analysis above shows that we can improve the network reliability by increasing the rebroadcast probability; however, it will raise the end-to-end delay and the power consumption. The increased power consumption can be explained as follows: when the value of p

is high, many redundant packets are produced in the network. After a packet has been received, the energy spent on retransmitting it again is wasted (fig. 2(a), (b)). In addition, the magnitude of change in the energy and the latency values is fairly small, around 40%, as illustrated in Figure 2(b). This means that the rebroadcast probability p may have a greater impact on network reliability, than on the delay or energy consumption.

2.2. Impact of the Sleep Time

Sensor nodes often conserve their power by switching between sleeping and working states. When a node is in the sleeping state, its radio is turned off and the node is not able to receive or send packets. Otherwise, when a node is in the active state, it is fully functional. We conduct the second experiment scenario to evaluate the impact of the sleep time on the non-functional requirements of the application. The simulated sensor nodes in this experiment are similar to the nodes used in the first scenario. In this case, each node serves as a source node and as a relay node for sending packets to a sink node. Every node, except the sink will periodically send packets to the sink. A data collection tree routing protocol [16] is implemented in each node to support packet forwarding. The simulated nodes are configured to switch between sleep mode and active mode with a predefined sleep time. Twelve experiments are conducted with twelve different values for the sleep time (in seconds), distributed evenly in the interval $[0, 60]$. Figure 1(b) demonstrates the experiment configuration in the second scenario.

Figure 3 illustrates how different sleep time durations affect reliability, energy consumption, and latency. As shown in figure 3(a) and (b), a duty-cycling mechanism which puts sensor nodes into sleep mode during idle periods is an effective approach to conserving energy. As seen in the figure, the average energy spent on each delivered packet decreases dramatically when the sleep time is 15 seconds. However, further increments to the sleep period does not save more energy. This may be due to the increased overhead cost in packet transmission and routing, when nodes stay longer in the sleep mode. Additionally, increasing

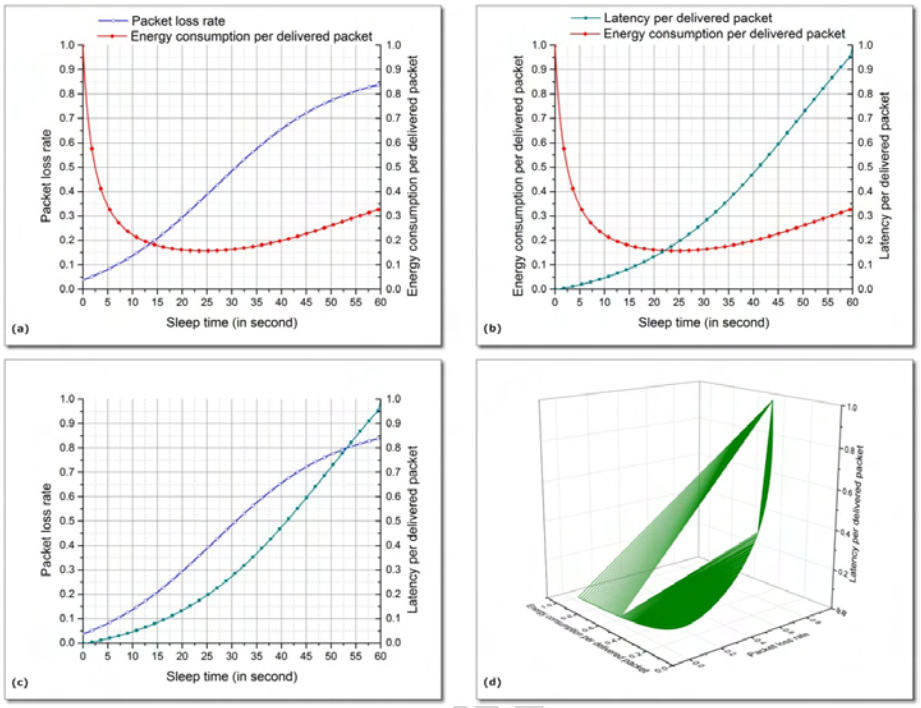

Figure 3: The impacts of sleep time on the non-functional application properties

the sleep time will introduce latency and packet loss rate into the network. In fact, as illustrated in Figure 3(c), the average latency per delivered packet and the packet loss rate increase almost proportionally to the sleep duration.

Based on the analysis presented in the last two sections, we can conclude that the adjustable parameters have complex influences on the non-functional parameters of WSN applications. Moreover, a single parameter may have contrary influences on different metrics, and the influences also strongly depend on the application context. Therefore, to obtain an acceptable compromise solution to the problem of minimizing all the desired non-functional parameters, a Pareto multi-objective optimization technique is required.

3. Optimization Method

3.1. Formulating the Optimization Problem

To optimize a WSN application regarding desired non-functional parameters such as *network lifetime*, *energy per delivered packet*, *latency per delivered packet*, or *packet loss ratio*, developers must first define a set of adjustable parameters whose values can be tuned to meet the application requirements. These parameters and their possible value ranges are specified when designing the application. In this step, the developers must also define an objective function such as the maximization or minimization of a set of non-functional parameters. In the optimization process, the application specified together with a concrete set of adjustable parameters is executed on a simulator. During the simulation process, the simulator will collect all experiment data and measure the metrics. The quality of the candidate application will be evaluated based on the metric values.

We assume that the target application has a set of N adjustable parameters $\{X_i\}_{i=1..N}$ and a set of M non-functional parameters $\{P_j\}_{j=1..M}$. Each X_i has a default value $X_{iDefaultvalue}$ and a value range from $X_{iMinvalue}$ to $X_{iMaxvalue}$ that are predefined by the developer. The possible parameter values form a solution space, in which each candidate solution is a concrete set of adjustable parameter values $S_C = \{X_{C1}, X_{C2}, ...X_{CN}\}$. The candidate solution is mapped to a point in the objective space by a set of evaluation metrics $P_C = \{P_{C1}, P_{C2}, ...P_{CM}\}$. The mapping from the solution space $\{S\}$ to the objective space $\{P\}$ is defined experimentally by the simulation. Then the multi-objective optimization can be formulated as follows:

$$\begin{aligned} &min\ P(X) = [P_1(X), P_2(X), ...P_M(X)] \\ &subject\ to\ X = [X_1, X_2, ...X_N] \\ &X_{iMinvalue} \leq X_i \leq X_{iMaxvalue} \end{aligned} \tag{1}$$

The multi-objective problem presented in equation 1 can be solved by using a common technique named *linear scalarization* that combines the multiple

objectives into one single-objective function. In more detail, this technique minimizes a weighted sum of the objectives as below:

$$\begin{gathered} min\ P(X) = \sum_{i=1}^{M} w_i P_i(X) \\ subject\ to\ X = [X_1, X_2, ...X_N] \\ X_{iMinvalue} \leq X_i \leq X_{iMaxvalue} \\ w_i \geq 0, i = 1...n \end{gathered} \tag{2}$$

The weighting coefficients w_i in equation 2 are chosen and maintained by the decision maker who is solving the optimization problem. In general, there is no a priori correspondence between the weighting coefficients and the solution vector. Therefore, it is difficult for the decision maker to be aware of which weights are the most appropriate to produce a good solution. In practice, the decision maker must try with different weight vectors in order to find out the ones that can produce a satisfactory solution. This is a technical shortcoming of the scalarization method because performing many optimizations with different weight vectors will significantly increase the computational burden.
Another approach to solving the multi-objective optimization problem is to use the concept of the Pareto optimality which is defined as follows:

- An objective vector $P_U = \{P_{U1}, P_{U2}, ...P_{UM}\}$ dominates an objective vector $P_V = \{P_{V1}, P_{V2}, ...P_{VM}\}$ if P_U is better than P_V with respect to at least one objective and not worse than P_V with respect to all other objectives, or represented in mathematic terms:

$$\begin{gathered} P_U\ dominates\ P_V \Leftrightarrow \exists i \in \{1...M\}\,| \\ P_{Ui} < P_{Vi} \cap \forall i \in \{1...M\} | P_{Ui} \leq P_{Vi} \end{gathered} \tag{3}$$

- A solution s' is said to be a Pareto optimum for the problem if and only if there is no $s \in S$ such that $P_i(s) \leq P_i(s')$ for all $i \in \{1...M\}$. Based on equation 3 we can say a solution s' is a Pareto optimum if there is

no solution s such that the objective vector $P(s)$ dominates the objective vector $P(s')$.

A very common situation in the multi-objective optimization problem is that an improvement in one objective will lead to a degradation in one or more of the remaining objectives. In this case, the ideal optimal solution does not exist; instead, the solution is the set of all Pareto optimum solutions. This set of non-dominated solutions is named the Pareto frontier of the solution space. Thus, solving a multi-objective optimization problem is understood as computing or approximating the Pareto frontier and choosing the most appropriate Pareto optimum solution for the problem.

In the next subsection, a method for approximating the Pareto frontier is introduced.

3.2. SPEA2 Approach

To solve the optimization problem mentioned above, we apply SPEA2 (Strength Pareto Evolutionary Algorithm 2) which is a Multi-Objective Evolutionary Algorithm (MOEA) [10]. Compared to other heuristic algorithms, a MOEA is able to search for a set of solutions as a result. In the context of multiple objectives it means that a MOEA can search for a representative set of Pareto-optimal solutions, approximating the true Pareto frontier in a single run. After the optimization users may pick a solution of their interest from the Pareto set, taking into account the possible tradeoffs between competing objectives.

SPEA2 is an improved version of the Strength Pareto Evolutionary Algorithm (SPEA) which uses a ranking procedure to assign better fitness values to non-dominated solutions that encourage uniform distribution of the population near the Pareto frontier. We refer to [17] for detailed information on SPEA and SPEA2. SPEA uses a regular population $\boldsymbol{P}$ and an external set called an archive set $\boldsymbol{E}$. The archive set stores all non-dominated solutions of the search space that have been investigated so far during the search. In each iteration, fitness values are assigned to both archive and population members:

- Each individual $\boldsymbol{y}$ in the archive $\boldsymbol{E}$ is assigned a strength value defined as $s(y,t) = \frac{np(y,t)}{N_P+1}$, where $\boldsymbol{np(y,t)}$ is the number of individuals that $\boldsymbol{y}$ dominates in the population $\boldsymbol{P}$ and $\boldsymbol{N_P}$ is the size of the population $\boldsymbol{P}$.

- The fitness of the individual $\boldsymbol{y}$ is assigned as the strength value of $\boldsymbol{y}$: $\boldsymbol{f(y,t)=s(y,t)}$.

- The fitness of the individual $\boldsymbol{x}$ in the population $\boldsymbol{P}$ is calculated as:

$$f(x,t) = 1 + \sum_{\substack{\mathbf{y} \in \mathbf{E} \\ \mathbf{y\ dominates\ x}}} s(y,t)$$

In SPEA2, a fine-grained fitness assignment strategy which incorporates an estimation of the density of the Pareto frontier is employed. In detail, the strength value is assigned to individuals in both the archive set $\boldsymbol{E}$ and the population $\boldsymbol{P}$. Then the fitness of the individual $\boldsymbol{x}$ is calculated as:

$$f(x,t) = \sum_{\substack{\mathbf{y} \in \mathbf{E} \cup \mathbf{P} \\ \mathbf{y\ dominates\ x}}} s(y,t) + D(xt)$$

where $\boldsymbol{D(x,t)}$ is the density estimation of the individual $\boldsymbol{x}$. $\boldsymbol{D(x,t)}$ is evaluated using an adaptation of the k-th nearest neighbor method [17].

$$D(x) = \frac{1}{\sigma_x^k + 2}$$

where $\sigma_{\mathbf{x}}^{\mathbf{k}}$ is the Euclidean distance of the objective values between the solution $\boldsymbol{x}$ and its k-th nearest neighbor, and $\boldsymbol{k}$ is chosen as the square root of the size of the solution set $\boldsymbol{P}$ and archive set $\boldsymbol{E}$ combined.

The complete pseudo-code of the SPEA2 algorithm is presented in table 1 below.

In the listing, the function *EvaluateObjective()* accesses the simulator interface to estimate the objective values (non-functional parameters) for each solution. *CalRawFitness()* and *CalDensity()* functions are used to evaluate the

Table 1: Strength Pareto Evolution Algorithm 2

SPEA2 pseudocode
Input: P_N (population size)
E_N (archive size)
T (maximum number of generation)
Output: Set of non-dominated solutions
P ← InitializePopulation(P_N)
E ← $\emptyset$
while (!StopCondition(T)) **do**
for $S_i \in$ P **do**
$S_{iObjective}$ ← EvaluateObjective(S_i)
endfor
U ← P + E
for $S_i \in$ U **do**
$S_{iFitness}$ ← CalRawFitness(S_i,U) + CalDensity(S_i,U)
endfor
E ← GetNonDominated(U)
if *Size(E)* <E_N **then**
FillArchiveWithBestRemaining(E, E_N ,U)
elseif
RemoveMostSimillar(E, E_N)
endif
Parents ← SelectParents(E)
P ← CrossoverAndMutation(Parents)
endwhile
return GetNonDominated(E)

raw fitness of a given solution (the sum of strength of the solutions that dominate this solution) and the density of the related area of the Pareto frontier. The fitness value of a solution, is then assigned as the sum of the raw fitness and the density value. The *FillArchiveWithBestRemaining()* function is used to fill the archive set with remaining candidate solutions in order of fitness value. The function *RemoveMostSimilar()* truncates the archive set by removing those members with the smallest value of $\sigma_{\mathbf{x}}^{\mathbf{k}}$ as previously calculated. The binary tournament selection method is implemented in the function *SelectParents()* to choose parents from the archive set. The *CrossoverAndMutation()* function is used to perform the crossover and mutation operators on the selected parent individuals.

One of the first decisions that have to be taken when using a MOEA is to define how to encode or represent the solutions of the problem to solve. In our problem, a solution candidate is composed of a set of real numbers corresponding to the set of adjustable parameter values. Therefore, we used real-coded methods [18], where an individual (a solution in the solution space) is a vector of floating point numbers:

$$\begin{gathered} X = \{x_1, x_2, ..., x_N\} \\ for\ all\ i = 1...n,\ x_i^L \leq x_i \leq x_i^U \\ where:\ x_i^L\ -\ lower\ boundary\ of\ the\ i^{th}\ gene, \\ x_i^U\ -\ upper\ boundary\ of\ the\ i^{th}\ gene \end{gathered} \tag{4}$$

In order to generate new solutions from existing ones, EAs use selection, crossover and mutation operators:

- The selection operator chooses individuals from the population for later breeding. We use binary tournament selection with replacement in which two individuals are taken at random and the better individual is selected from the two. After selecting the better ones, the two individuals are immediately replaced into the population for the next selection operation.

- The crossover operator combines the features of two parent individuals to form two offspring, with the possibility that good individuals may generate better ones. We implement Simulated Binary Crossover (SBX) operator .[46] that is applied to a pair of parent individuals: $X^1 = \{x_1^1, x_2^1, \ldots, x_N^1\}$ and $X^2 = \{x_1^2, x_2^2, \ldots, x_N^2\}$. In this operator, the following steps are involved to generate two offspring solutions: $C^1 = \{c_1^1, c_2^1, \ldots, c_N^1\}$ and $C^2 = \{c_1^2, c_2^2, \ldots, c_N^2\}$.

  Step 1: For each variable x_i a number u_i is created randomly between 0 and 1.

  Step 2: A probability distributed function is specified to create offspring solutions so that they have the same search power as that in a single-point crossover in binary-coded method. Then find a number β_i so that the area under the probability curve is equal to the number u_i. As figured out in .[46], β_i is calculated using the following formula, where η is a parameter called the distribution index for crossover. The larger value of η the higher probability for generating a near-parent offspring, and vice versa. It is common in the literature to set η to 20:

$$\beta_i = \begin{cases} (2u_i)^{\frac{1}{\eta+1}} & , if\ u_i \leq 0.5 \\ \left(\frac{1}{2(1-u_i)}\right)^{\frac{1}{\eta+1}} & , otherwise \end{cases}$$

  Step 3: Compute the offspring as: $c_i^1 = 0.5\left[(1+\beta_i)\, x_i^1 + (1-\beta_i) x_i^2\right]$ and $c_i^2 = 0.5\left[(1-\beta_i)\, x_i^1 + (1+\beta_i) x_i^2\right]$

- The mutation operator is used to maintain the diversity of the population in EA. It operates on a single selected individual at a time and modifies the individual independent of the rest of the population members. Among the many mutation operators for real-coded individual that are investigated in .[168], we implement the polynomial mutation operator in our approach. In this operation the following steps are used to generate the offspring $C = \{c_1, c_2, \ldots, c_N$ from the parent individual $X = \{x_1, x_2, \ldots, x_N$:

Step 1: For each variable x_i a number u_i is created randomly between 0 and 1.

Step 2: Using a polynomial probability distribution function:

$P(\delta) = 0.5\,(\eta+1)\,(1+|\delta|)^{\eta}$ to calculate the parameter δ_i as:

$$\delta_i = \begin{cases} (2u_i)^{\frac{1}{\eta+1}} - 1 & ,if\ u_i \leq 0.5 \\ 1 - (2(1-u_i))^{\frac{1}{\eta+1}} & ,otherwise \end{cases}$$

Where η is called as distribution index for mutation parameter. It is common in the literature to set η to 20.

Step 3: Calculate the offspring as:

$$c_i = x_i + \delta_i \left(x_i^U - x_i^L\right),$$

$$where\ x_i^U\ and\ x_i^L\ are\ the\ upper\ bound\ and\ lower\ bound\ of\ x_i$$

4. Model-driven Performance Engineering

4.1. Overview of the Application Development Process

We employ a MDD approach (Model-Driven Development) to describe and realize WSN applications [11, 12]. Figure 4 presents eight steps in the application development process using our MDD framework.

These steps can be summarized as follows:

- Step 1 - Modeling: Starting with an applications requirement specification, a PIM (Platform-Independent Model), which consists of the formal definition of the applications, is created using a DSL (Domain Specific Language). The proposed DSL offers a set of declarative sentences to express the behavior of sensor nodes such as sampling, aggregation and forwarding which is necessary for developing WSN applications. A PIM also contains other information that will be used in the later steps, such as the set of adjustable application parameters, the non-functional requirements, the objective function as well as the simulation configuration. An

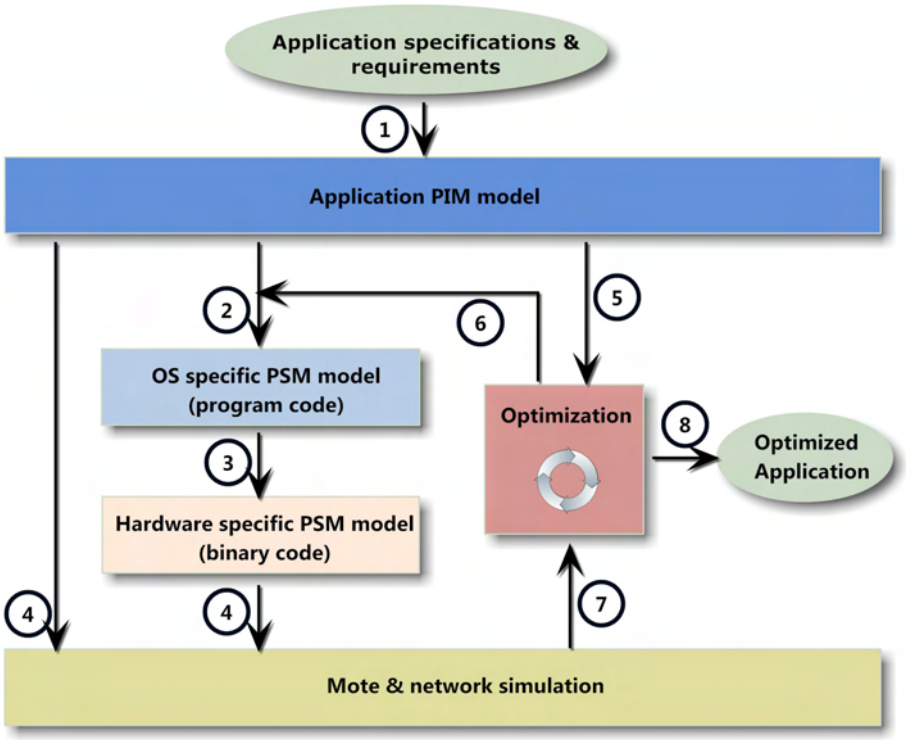

Figure 4: An overview of the application development process

example is given in figure 5.

- Step 2 - Transforming: The transformation process consists of the application of mapping rules and code template files to transform the PIM into the equivalent operating system-specific code (Platform-Specific Model for the given OS).

- Step 3 - Compiling: The binary code for the selected hardware platform is generated (hardware-specific PSM).

- Step 4 - Testing: The binary code is deployed and tested on the simulator

with the pre-defined simulation configuration in the PIM. The simulator has two interfaces. The first one is a graphical user interface, which is used to visualize the output of the simulation process. The second one is a text-based interface, which is used to provide normalized measured data to the optimization component. Depending on the testing result the developer will determine whether the application satisfies all the functional requirements.

- Step 5, 6, 7 - Optimizing: In these steps the successfully tested application is optimized. The set of adjustable parameters, which is defined in the PIM, is used by the optimization component to generate the search space (step 5). The evaluation of each candidate solution (step 6) is done by: i) repeating step 2 to step 4; ii) evaluating the normalized measured data from the simulator (step 7).

- Step 8 - Choosing: Based on the optimization results the developer will select the optimal application by evaluating the tradeoff between different constraint criteria and performance values.

4.2. Optimization Component Design

The design of the optimization component is presented in Figure 6. Application developers can interact with the component via a *Command-line User Interface*. To start the optimization process, the developer must specify some configuration files such as a simulation scenario, an application configuration and a corresponding PIM that describes the input parameters and the non-functional requirements that have to be optimized. The developer is suggested to verify the PIM before the optimization stage to ensure that the application meets all its functional requirements. This verification can be done by testing the application with the simulation using a proper scenario.

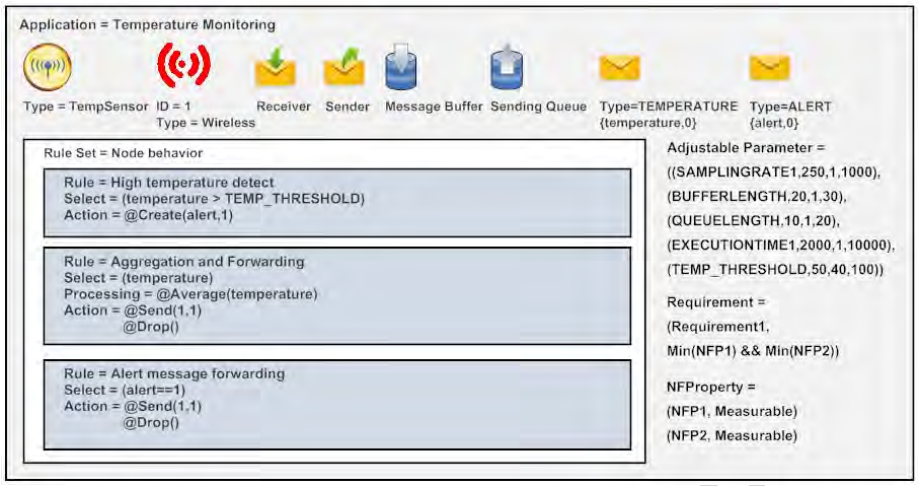

Figure 5: An example for a PIM in the MDD framework

During the optimization process various sets of adjustable parameter values are generated and stored into temporary files. The *Application Translation Interface* updates the PIM with the generated adjustable parameter values and executes the translation component and the compiler to create a new binary image of the application. The *Simulation Interface* is designed to interact with the Cooja simulator. It is a multi-threading application that is able to launch a maximum of 8 simulations in parallel. Each simulation is run with an instance of the Cooja simulator as Cooja is a single-thread program. The *Simulation Interface* takes as input the application's binary image and the simulation scenario. After that it generates an equivalent simulation file for Cooja and runs the simulation with a new instance of the Cooja simulator. When the simulation is finished, the *Simulation Interface* updates the optimization component with the simulation result.

The MOEA component is developed based on JMetal [19] - a popular object-oriented Java framework for multi-objective optimization with meta-heuristics. We extend JMetal with our solution representation model, a Simulated Binary Crossover [20] and a Polynomial Mutation Operator [21] and the SPEA2 fitness evaluation scheme. During the optimization, the simulation results are stored

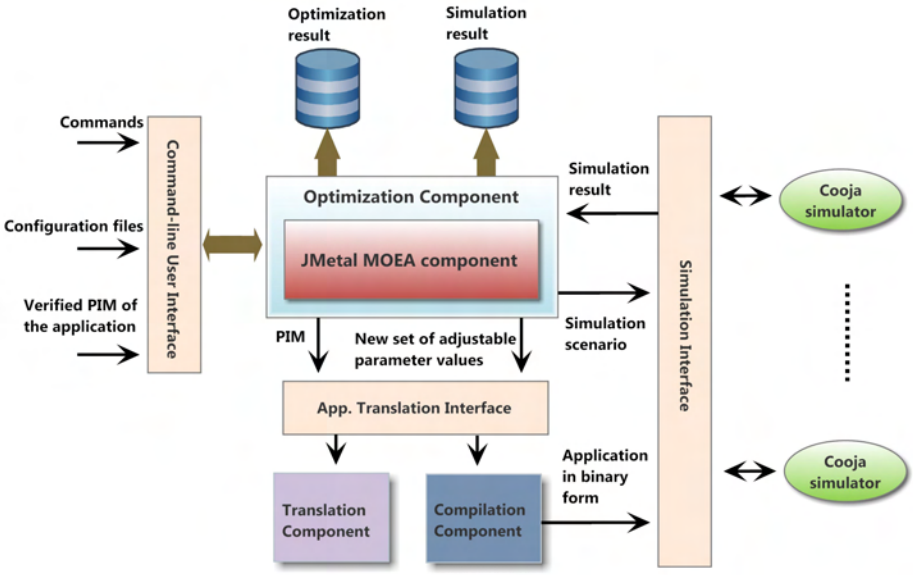

Figure 6: The optimization component design

separately from the optimization results. This information can be used by the developers for further analysis.

5. A Case Study and Evaluation

This section discusses the design, implementation and optimization of a real-world application named *TempSense*, where a sensor network is used to monitor the temperature in an area. In our experiment, we designed and implemented *TempSense* on the target platform Z1 using the Contiki OS. Then, we optimized the application in terms of power efficiency and reliability.

5.1. TempSense

The *TempSense* application falls into the category of monitoring applications where a sensor network is used to collect temperature data in an area and reports the data through Web services. Each node is equipped with a sensor to periodically sense the local temperature. The data is forwarded along the network to a base station. A Web service is installed on the base station to

support accessing the data via the Internet. An improved version of the gossip routing protocol is used for data transmission from nodes to the base station in the network:

- For the first K hops, messages are forwarded with probability 1.
- From hop $K+1$, messages are forwarded with probability P.
- Message lifetime is implemented to avoid infinite loops. A message will be dropped if the lifetime is exceeding a threshold L.

Developers can assign necessary values for the adjustable parameters like *BufferLength*, *MaxLifeTime* and *SleepTime* to define the maximum length of the message buffer, the maximum lifetime of messages and the sleep time. The default value and the value range of each parameter are also defined by setting the values of the attributes *Defaulvalue*, *Minvalue* and *Maxvalue*. To optimize the reliability and the energy efficiency of the system, non-functional parameters such as *energy per delivered packet* ($NFP1$) and *packet loss ratio* ($NFP2$), can be defined by the developer $Min(NFP1)$ && $Min(NFP2)$. Table 2 and table 3 present the lists of adjustable parameters and non-functional constraints for the *TempSense* application. Now we can formulate the optimization problem as follows:

$$\begin{gathered} min\ P(X) = [P_1(X), P_2(X)] \\ subject\ to\ X = [X_1, X_2, X_3, X_4, X_5, X_6]\ where \\ 1 \leq X_1 \leq 30; 1 \leq X_2 \leq 20; 1000 \leq X_3 \leq 60000 \\ 100 \leq X_4 \leq 5000; 1 \leq X_5 \leq 10; 0.1 \leq X_6 \leq 1 \end{gathered} \tag{5}$$

To test the *TempSense* application, we configured a simulated network of 200 nodes distributed randomly in an area of 500 x 500 meters. The simulated network has a sink node that receives temperature data from all other nodes that execute the *TempSense* application. The radius of the transmission range and the interference range of each node in the simulation are 50 meters and 80

Table 2: List of non-functional parameters and constraints

Name	Describe
NFP1	Energy per delivered packet
NFP2	Packet loss ratio
Non-functional constraints	Min(NFP1) && Min (NFP2)

Table 3: List of adjustable parameters

Parameter Name	Variable Name	Description	Default value	Min. value	Max. value
BUFFER LENGTH	X1	Size of the buffer storing incoming messages	20	1	30
QUEUE LENGTH	X2	Size of the buffer storing outgoing messages	10	1	20
SLEEP TIME	X3	During this time, the mote is in sleeping state	20000	1000	60000
MAX LIFE-TIME	X4	Maximum lifetime of a message in the network	2500	100	5000
GOSSIP_K	X5	Messages in the gossip protocol are forwarded in the first K hops	5	1	10
GOSSIP_P	X6	From hop K+1, messages are forwarded with probability P	0.6	0.1	1

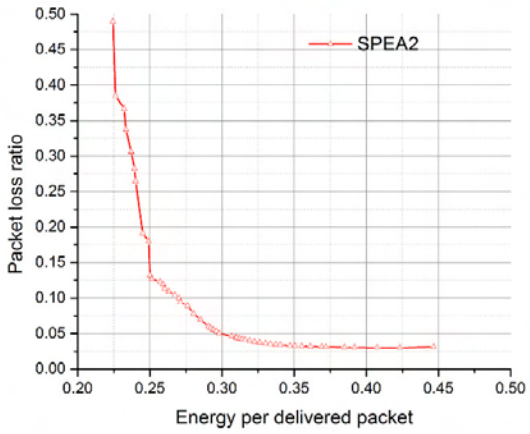

Figure 7: The Pareto frontier obtained when using SPEA2

meters respectively. In each experiment, we run the simulation three times, with each round being one minute long. The simulator collects all the non-functional parameters of the application after each simulation round but only the average metric values are reported to the optimization component.

5.2. Evaluation

The SPEA2 approach is used to solve the optimization problem. The size of the working population is $N = 50$. In our problem, a solution candidate is composed of set of real numbers or integers corresponding to the set of adjustable parameter values. We used real-coded methods where an individual (a solution in the solution space) is a vector of floating point numbers. The value of an parameter will be rounded up to the nearest integer when evaluating a solution candidate. In order to generate new solutions from existing ones, binary tournament selection with replacement, Simulated Binary Crossover (SBX) and polynomial mutation operators are applied. The fitness of each solution candidate is evaluated using the simulator. Our tests show that the convergence occurs quickly within 50 generations. Figure 7 shows the Pareto frontier obtained after the optimization process. Table 4 presents an example set of ten solutions selected from the Pareto frontier (with the round values of adjustable

Table 4: An example set of ten solutions for TempSense

	X1	X2	X3	X4	X5	X6	P1	P2
Default value	20	10	20000	2500	5	0.6	0.40007	0.41695
Solution 1	5	8	13500	1500	3	0.65	0.25131	0.12679
Solution 2	5	10	13500	2000	3	0.55	0.25019	0.13152
Solution 3	5	8	15500	3000	3	0.45	0.28512	0.06933
Solution 4	10	8	20000	3500	3	0.50	0.30672	0.04636
Solution 5	10	6	21000	3500	3	0.55	0.32251	0.03858
Solution 6	10	4	12000	4000	4	0.65	0.35054	0.03269
Solution 7	5	6	18500	1000	3	0.55	0.24006	0.26466
Solution 8	5	6	21500	4000	3	0.55	0.36122	0.03171
Solution 9	15	8	25500	3500	4	0.65	0.39176	0.03042
Solution 10	15	8	30500	1000	3	0.6	0.22616	0.37103

parameters).

Our experiment results demonstrate that the application's satisfaction of non-functional requirements can be improved during the optimization process by selecting a proper set of adjustable parameters. We observed that our approach is able to achieve a good approximation of the Pareto set for the multi-objective optimization problem.

6. Related Work

Various optimization techniques have been examined in the scope of IoT, proposing several methods to optimize different WSN properties such as: node deployment, node localization, routing, data processing, cost and lifetime. In [4] the authors formulated the sensor node deployment task as a multi-objective optimization problem to maximize the coverage and lifetime, minimize the number of deployed sensor nodes while maintaining connectivity between each sensor node and the sink. In [22] Vecchio et al. introduced an approach for the local-

ization problem in WSNs where a MOEA is used to improve the effectiveness and the stability of a localization protocol.

To investigate the trade-off between QoS performances in a WSN routing protocol, in [23] a heuristic algorithm is used to design a multi-objective QoS routing to support diverse requirements by different WSN applications. In the context of data gathering and fusion in WSN, a multi-objective optimization technique has been shown to be very effective to prolonging the network's lifetime while maintaining the QoS metrics of the application [24, 25].

The authors of [26, 27] studied a modular hardware architecture for sensor nodes which provides energy optimization opportunities at the hardware level via adjustable parameters (i.e., processor voltage and frequency, sensing frequency, duty cycle, etc.). When designing a communication protocol for sensor networks, developers can tune different parameters such as channel access schedule, message size, and duty cycle to meet application requirements [28].

In [29] the authors mentioned several dynamic optimization approaches that enable sensor nodes to dynamically customize their parameter values in situ according to both operating environment and application requirements. In [30], the authors presented an method for software reconfiguration in WSNs based on model-integrated computing. The authors modeled the WSN operation space (defined by the WSN software components' models and application requirements) and defined reconfiguration as the process of switching from one point in the operation space to another. In that way, the key question of reconfiguring WSNs is considered as a search problem in the operation space. Another dynamic optimization method for WSNs is presented in [31]. The authors implemented the Markov Decision Process to dynamically optimize mote processor voltage, frequency, and sensing frequency in accordance with application requirements over the lifetime of the mote. The main drawback of the dynamic optimization method is that the optimization module has to be implemented on each sensor node despite of resource constraints.

An important issue that cannot be neglected, but that was not discussed in this work, is the preservation of information security on shared data. This can

be considered as an orthogonal dimension to our problem area. However, we would like to mention relevant works that are suitable to extend our framework. Munir et al. [32] presented an optimized light weight authentication protocol for IoT devices. The study shows that the protocol outperforms other works in terms of computation, storage and communication cost. To deal with the location privacy problem in IoT, the authors of [33, 34] proposed a user-defined privacy location-sharing framework with an optimized query algorithm to protect a user's location privacy. The experiment results show that the framework incurs a lower time complexity than existing frameworks. To offer real-time data security when combining IoT and cloud computing, different types of security frameworks have been introduced. Chang et al. [35, 36] applied a multi-layered security framework which could be used as an integrated data security solution for business clouds.

Our solution presented in this paper can assist designers in optimizing applications regarding non-functional requirements at different design levels, such as hardware, software, or communication protocol, by adjusting suitable parameters. Beside that, our MDD framework is not only simplifying the application development, but it enables developers to execute the optimization process more easily and comprehensible, and allows them to select a solution of the Pareto set.

7. Conclusions and Future Work

We have developed a new solution for IoT systems to deal with non-functional constraint problems in an early phase of development. In the first stage, the interdependencies between adjustable platform and application parameters and non-functional requirements are examined. Furthermore, major requirements are identified and represented as non-functional constraints. Then a MOEA method is used to solve the optimization problem. To evaluate the proposed method, a case study was implemented. The experiment demonstrates that non-functional constraints, such as power consumption and reliability, can be

improved significantly during the optimization process by selecting a proper set of adjustable parameters. In our future work, we try to reduce the efforts for the experiments needed for the optimization process. In order to reduce the total runtime of the optimization task, the simulations could be executed in parallel if a sufficient number of processors is available, or they can be offloaded to cloud servers and surrogates nearby.

Acknowledgment

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.01-2016.03. The authors would like to thank the German Research Foundation (DFG) for supporting their participation in worldwide research networks.

8. References

[1] Z. Fei, B. Li, S. Yang, C. Xing, H. Chen, L. Hanzo, A survey of multi-objective optimization in wireless sensor networks: Metrics, algorithms and open problems, IEEE Communications Surveys & Tutorials 99 (2016) 1–38.

[2] Y. Zhang, S. He, J. Chen, Data gathering optimization by dynamic sensing and routing in rechargeable sensor networks, IEEE/ACM Transactions on Networking 24 (3) (2016) 1632–1646.

[3] Y. Qin, Q. Z. Sheng, N. J. Falkner, S. Dustdar, H. Wang, A. V. Vasilakos, When things matter: A survey on data-centric Internet of Things, Journal of Network and Computer Applications 64 (2016) 137–153.

[4] S. Sengupta, S. Das, M. Nasir, B. K. Panigrahi, Multi-objective node deployment in WSNs: In search of an optimal trade-off among coverage, lifetime, energy consumption, and connectivity, Engineering Applications of Artificial Intelligence 26 (1) (2013) 405–416.

[5] H. A. Hashim, B. O. Ayinde, M. A. Abido, Optimal placement of relay nodes in wireless sensor network using artificial bee colony algorithm, Journal of Network and Computer Applications 64 (2016) 239–248.

[6] L. Da Xu, W. He, S. Li, Internet of Things in industries: A survey, Industrial Informatics, IEEE Transactions on 10 (4) (2014) 2233–2243.

[7] M. Díaz, C. Martín, B. Rubio, State-of-the-art, challenges, and open issues in the integration of Internet of Things and cloud computing, Journal of Network and Computer Applications 67 (2016) 99–117.

[8] M. Iqbal, M. Naeem, A. Anpalagan, N. N. Qadri, M. Imran, Multi-objective optimization in sensor networks: Optimization classification, applications and solution approaches, Computer Networks 99 (2016) 134–161.

[9] R. Asorey-Cacheda, A.-J. Garcia-Sanchez, F. Garcia-Sanchez, J. Garcia-Haro, A survey on non-linear optimization problems in wireless sensor networks, Journal of Network and Computer Applications 28 (2017) 1–20.

[10] E. Zitzler, Evolutionary algorithms for multiobjective optimization: Methods and applications, Vol. 63, Citeseer, 1999.

[11] X. T. Nguyen, H. T. Tran, H. Baraki, K. Geihs, Frasad: A framework for model-driven IoT application development, in: Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on, IEEE, 2015, pp. 387–392.

[12] N. X. Thang, M. Zapf, K. Geihs, Model driven development for data-centric sensor network applications, in: Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia, ACM, 2011, pp. 194–197.

[13] N. X. Thang, K. Geihs, Model-driven development with optimization of non-functional constraints in sensor network, in: Proceedings of the 2010 ICSE Workshop on Software Engineering for Sensor Network Applications, ACM, 2010, pp. 61–65.

[14] F. Österlind, J. Eriksson, A. Dunkels, Cooja timeline: a power visualizer for sensor network simulation, in: Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, ACM, 2010, pp. 385–386.

[15] Z. J. Haas, J. Y. Halpern, L. Li, Gossip-based ad hoc routing, IEEE/ACM Transactions on Networking (ToN) 14 (3) (2006) 479–491.

[16] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, P. Levis, Collection tree protocol, in: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, ACM, 2009, pp. 1–14.

[17] E. Zitzler, M. Laumanns, L. Thiele, et al., SPEA2: improving the strength pareto evolutionary algorithm, in: Eurogen, no. 103 in 3242, 2001, pp. 95–100.

[18] F. Herrera, M. Lozano, J. L. Verdegay, Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis, Artificial intelligence review 12 (4) (1998) 265–319.

[19] J. J. Durillo, A. J. Nebro, jmetal: A java framework for multi-objective optimization, Advances in Engineering Software 42 (10) (2011) 760–771.

[20] K. Deb, R. B. Agrawal, Simulated binary crossover for continuous search space, Complex Systems 9 (3) (1994) 1–15.

[21] M. Raghuwanshi, O. Kakde, Survey on multiobjective evolutionary and real coded genetic algorithms, in: Proceedings of the 8th Asia Pacific symposium on intelligent and evolutionary systems, 2004, pp. 150–161.

[22] M. Vecchio, R. López-Valcarce, F. Marcelloni, A two-objective evolutionary approach based on topological constraints for node localization in wireless sensor networks, Applied Soft Computing 12 (7) (2012) 1891–1901.

[23] G. H. Ekbatanifard, R. Monsefi, M.-R. Akbarzadeh-T, M. Yaghmaee, et al., A multi-objective genetic algorithm based approach for energy efficient QoS-routing in two-tiered wireless sensor networks, in: Wireless Pervasive

Computing (ISWPC), 2010 5th IEEE International Symposium on, IEEE, 2010, pp. 80–85.

[24] H. Lin, H. Uster, Exact and heuristic algorithms for data-gathering cluster-based wireless sensor network design problem, Networking, IEEE/ACM Transactions on 22 (3) (2014) 903–916.

[25] A. Thakkar, K. Kotecha, Cluster head election for energy and delay constraint applications of wireless sensor network, Sensors Journal, IEEE 14 (8) (2014) 2658–2664.

[26] J. Byun, B. Jeon, J. Noh, Y. Kim, S. Park, An intelligent self-adjusting sensor for smart home services based on zigbee communications, Consumer Electronics, IEEE Transactions on 58 (3) (2012) 794–802.

[27] P. Dutta, J. Taneja, J. Jeong, X. Jiang, D. Culler, A building block approach to sensornet systems, in: Proceedings of the 6th ACM conference on Embedded network sensor systems, ACM, 2008, pp. 267–280.

[28] D. H. Phan, J. Suzuki, S. Omura, K. Oba, A. Vasilakos, Multiobjective communication optimization for cloud-integrated body sensor networks, in: Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on, IEEE, 2014, pp. 685–693.

[29] P. K. Dutta, D. E. Culler, System software techniques for low-power operation in wireless sensor networks, in: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design, IEEE Computer Society, 2005, pp. 925–932.

[30] A. Taherkordi, F. Loiret, R. Rouvoy, F. Eliassen, Optimizing sensor network reprogramming via in situ reconfigurable components, ACM Transactions on Sensor Networks (TOSN) 9 (2) (2013) 14.

[31] A. Munir, A. Gordon-Ross, An MDP-based dynamic optimization methodology for wireless sensor networks, Parallel and Distributed Systems, IEEE Transactions on 23 (4) (2012) 616–625.

[32] R. Amin, N. Kumar, G. Biswas, R. Iqbal, V. Chang, A light weight authentication protocol for IoT-enabled devices in distributed cloud computing environment, Future Generation Computer Systems (2016) 1–27.

[33] G. Sun, Y. Xie, D. Liao, H. Yu, V. Chang, User-defined privacy location-sharing system in mobile online social networks, Journal of Network and Computer Applications (2016) 1–12.

[34] G. Sun, D. Liao, H. Li, H. Yu, V. Chang, L2p2: A location-label based approach for privacy preserving in LBS, Future Generation Computer Systems (2016) 1–10.

[35] V. Chang, M. Ramachandran, Towards achieving data security with the cloud computing adoption framework, IEEE Transactions on Services Computing 9 (1) (2016) 138–151.

[36] V. Chang, Y.-H. Kuo, M. Ramachandran, Cloud computing adoption framework: A security framework for business clouds, Future Generation Computer Systems 57 (2016) 24–41.