DingNet: A Self-Adaptive Internet-of-Things Exemplar

Conference Paper · May 2019

DOI: 10.1109/SEAMS.2019.00033

2 authors, including:

Danny Weyns
KU Leuven
286 PUBLICATIONS **7,198** CITATIONS

SEE PROFILE

DingNet: A Self-Adaptive Internet-of-Things Exemplar

Michiel Provoost
Department of Computer Science
Katholieke Universiteit Leuven, Kulak
8500 Kortrijk, Belgium
michiel.provoost1@student.kuleuven.be

Danny Weyns
Department of Computer Science
Katholieke Universiteit Leuven & Linnaeus University
3001 Leuven, Belgium & 351 95 Vaxjo, Sweden
danny.weyns@kuleuven.be

***Abstract*—Recent efforts have shown that research on self-adaptive systems can benefit from exemplar to evaluate and compare new methods, techniques and tools. One highly relevant application domain for self-adaptation is the Internet-of-Things (IoT). While some initial exemplars have been proposed for IoT, these applications are small in scale and hence limited to support research on larger IoT deployments, such as typically required in smart cities. To address this limitation, we introduce the DingNet exemplar, a reference implementation for research on self-adaptation in the domain of IoT. DingNet offers a simulator that maps directly to a physical IoT system that is deployed in the area of Leuven, Belgium. DingNet models a set of geographically distributed gateways that are connected to a user application deployed at a front-end server. The gateways can interact over a LoRaWAN network with local, possibly mobile motes that can be equipped with sensors and actuators. The exemplar comes with a set of scenarios for comparing the effectiveness of different self-adaptive solutions. We illustrate how the exemplar is used for a typical adaptation problem of IoT where mobile motes dynamically adapt their communication settings to ensure reliable and energy efficient communication.**

***Index Terms*—Self-adaptation, exemplar, Internet-of-Things**

I. Introduction

The complexity to manage modern software systems on the one hand, and the uncertainties these systems face during operation on the other hand, are primary drivers for self-adaptation [1]–[7]. However, the past two decades have shown that engineering self-adaptive systems is a challenging problem both to researchers and software engineers [2], [3], [8]. To streamline the research efforts, the community of adaptive and self-managing systems has created a repository of artifacts in the form of model problems, prototype applications, tools, and frameworks [9]. These artifacts can be used by researchers to evaluate new solutions and compare their results with other solutions. A pioneering artifact was *Znn.com*, a web-based client-server system, that provides a news service that serves multimedia news content to its customers.[1] Other examples are Tele Assistance System (TAS), a service-based system in the domain of e-health [10], Self-Adaptive Video Encoder focusing on adaptive data streaming [11], and Hogna that offers a platform for experimentation in Cloud environments [12]. These and other exemplars have been used in numerous research efforts, two examples are [13] that uses Znn.com to evaluate the resilience in self-adaptive systems using probabilistic model-checking and [14] that uses TAS to compare an architecture-based and control-based approach to adaptation.

[1]Znn.com is available via: https://github.com/cmu-able/znn

One of the emerging domains for research on self-adaptation is the Internet-of-Things (IoT). IoT is used in smart city applications, such as smart parking systems and environment monitoring. Engineering such IoT systems is challenging since these systems typically consist of complex infrastructure of gateways and devices that are spread over a city area. One particular challenge is handling uncertain operating conditions, such as interference in the wireless communication between devices and gateways, and dynamics in the environment that are difficult to predict. Without proper mitigation of such uncertainties, dependability goals of IoT systems, such as reliability and energy efficiency, may be jeopardized. Self-adaptation provides a key solution to tackle these challenges.

To support researchers in this domain, a few initial exemplars have been proposed. One example is Intelligent Ensembles [15] that offers a simulator of a cyber-physical system in the domain of traffic. Another example is DeltaIoT [16] that offers a simulator and a physical deployment of an IoT network for smart environment monitoring. While these exemplars have demonstrated their usefulness, see e.g., [17], the application scenarios supported by these artifacts are small in scale and hence limited to support research on larger IoT deployments, such as typically required in smart cities.

To address this limitation, we introduce the DingNet exemplar, a reference implementation for research on self-adaptation in the domain of IoT. DingNet offers a simulator that maps directly to a physical IoT setup that is deployed in Leuven, Belgium. DingNet allows modeling smart city applications that connect with a set of geographically distributed gateways that can interact over a wireless network with motes that can be equipped with sensors and actuators. Compared to existing IoT simulators that often only support stationary motes, e.g., [18], [19], a distinguish benefit of DingNet is support for mobile motes that move in the city area. We illustrate how the exemplar is used for a typical adaptation problem of IoT where mobile motes that move in a city area dynamically have to adapt their network settings to ensure reliable and energy efficient communication.

The remainder of the paper is structured as follows. Section II presents an overview of the DingNet exemplar and introduces a set of adaption scenarios. We explain how we devised the DingNet simulator and shows how a concrete IoT application can be set up and self-adaptation is supported. In Section III we show how researchers can use the exemplar for research on self-adaptive IoT systems using an example.

Section IV discusses the scope of applicability of the exemplar. Finally, we draw conclusions in Section V.

II. Overview DingNet Exemplar

The DingNet exemplar supports the design and evaluation of smart city applications that comprise of potentially a large number of motes that collect and send data to gateways that are deployed in a city area. Example applications are garbage bins that send signals when they are full, devices that track the quality of the air in the city area, parking sensors that allow visualization of available places and locations, etc. We start this section with an overview of the DingNet network deployed in Leuven. Then we zoom in on the realization of the DingNet simulator; first the realization of the domain, second the support to study and realize self-adaptation.

A. DingNet IoT Network

Technically, DingNet[2] is a LoRaWAN-based IoT network that covers a large part of the city of Leuven. LoraWAN[3] is a communication protocol for wide area networks that is designed to allow low-powered motes to communicate with Internet-connected applications over long range wireless connections. Concretely, the DingNet network contains 14 gateways that are deployed on high buildings throughout the city to ensure good coverage. A mote in DingNet can send a few KB of data per 24 hour. The data can be encoded using keys to ensure secure transmission. Usually battery-powered (or energy-harvesting) IoT motes are used that can sense environment parameters and send the data to gateways. Motes communicate with the gateways within communication range. The gateways that receive data from the motes forward this data to an application server where the data is further processed by a user application. The communication between gateways and applications uses the MQTT protocol.[4] MQTT is lightweight publish/subscribe messaging transport for connections with remote locations where limited network bandwidth is available. Downstream traffic from the application to gateways and further to motes is also possible. This traffic enables the adaptation of the settings of devices. Figure 1 gives a high-level overview of the DingNet architecture.

Engineering smart cities applications with DingNet poses severe challenges to software engineers. First, these applications consist of a stationary and/or mobile motes equipped with sensors and actuators that interact with gateways in their neighborhood that are deployed over the city area. Testing such IoT applications directly in the field is cumbersome, error prone, and time-consuming. Simulation can support engineers with investigating and testing solutions before deploying them in the field. Second, smart city applications are subject to a variety of uncertainties, including network interference due to various factors in the environment, such as traffic and weather conditions, and dynamics in the data traffic produced in the network as motes may come and leave at will. In addition, the gateway infrastructure may be shared among multiple applications, affecting the availability of network resources. Self-adaptation offers a key solution to tackle these issues.

[2]https://admin.kuleuven.be/icts/english/services/dingnet

[3]https://www.thethingsnetwork.org/docs/lorawan/

[4]Message Queuing Telemetry Transport protocol; http://mqtt.org/

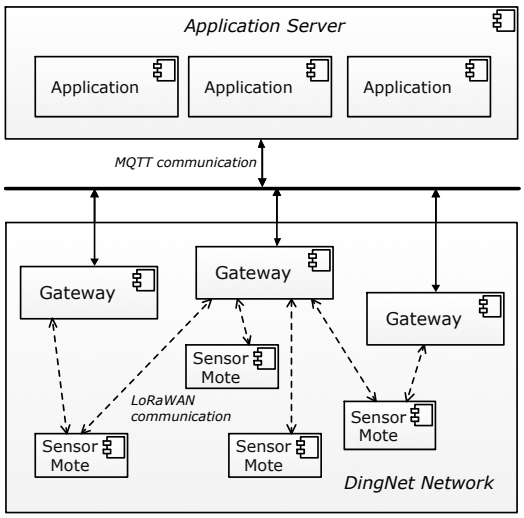

Fig. 1. DingNet Architecture

B. Exemplar Realization - Domain

The DingNet simulator provides an environment that directly maps to the DingNet deployment in Leuven. Figure 2 shows a high-level design of the core functionality provided by DingNet simulator.

The *Environment* has a *map*, a set of *motes* and a set of *gateways*. The map allows defining regions and paths. The environment has also a *clock* with the logical time, where each time tick represents a time duration wall clock time. Furthermore, the environment has a number of *characteristics*. A *Characteristic* can represent any property of interest associated with the environment. Currently, the simulator supports

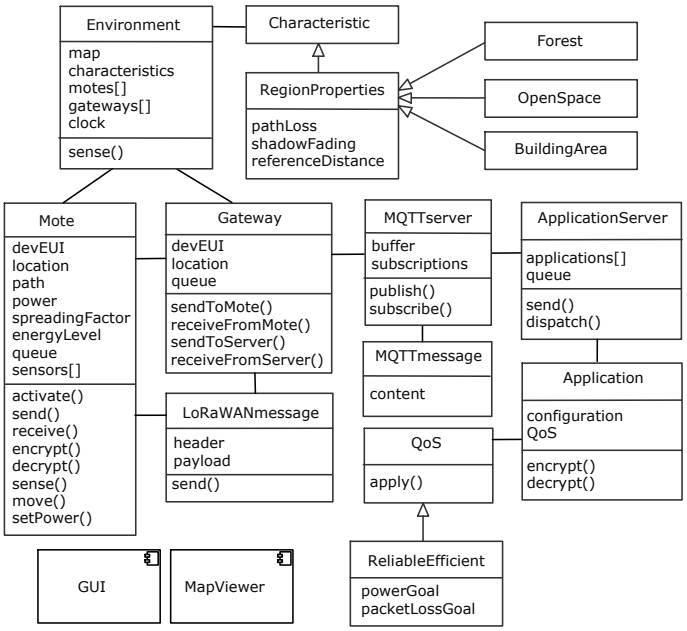

Fig. 2. Core functionality of the DingNet simulator

one type of characteristic, *RegionProperties*, which defines properties that determine the impact of the environmental conditions on the communication. Region properties are defined for three concrete types of regions: *Forrest*, *OpenSpace*, and *BuildingArea*. We derived the concrete values for *pathLoss*, *shadowFading*, and *referenceDistance* based on averages reported by Petajajarvi et al. [20]. The *pathLoss* is a parameter that specifies the signal decay over a certain terrain based on a *referenceDistance*. *ShadowFading* represents a parameter that specifies Gaussian noise in the environment, which is commonly used in IoT simulation. Other properties can be added as needed.

A *Mote* has an identifier, *devEUI*, a 64 bit unique identifier defined by LoRaWAN, a *location* in the environment that changes when the mote moves along a *path*, a *power* setting that determines the energy consumed by the mote for communication (between -1 min and 15 max power), the remaining *energyLevel* of the battery, a *queue* to buffer messages, and a set of *sensors[]* to interact with the environment. *activate()* starts the mote; it can then *send()* and *receive()* messages, *sense()* the environment and *move()* through it, *encrypt()* and *decrypt()* messages, and finally adapt its power setting using *setPower()*. For encryption and decryption, the mote uses a private key that it receives when registering at the system. At the other end, the application can decrypt and encrypt messages as well. Not shown in the design are the spreading factor that determines the duration of the communication (currently set to 12) and interactions with the environment via *actuators*, which are currently only abstractly defined.

The *Gateway* has also a *devEUI*, a *location*, and a *queue* to buffer messages. The gateway can exchange messages with motes and applications. Motes and gateways communicate using *LoRaWANmessages* via local antennas, while gateways and the *ApplicationServer* communicate via *MQTTmessages* that are relayed via an *MQTTserver*. The MQTT server uses a publish-subscribe communication mechanism. LoRaWAN messages have a *header* with the DevEUI van de sender among other info, and the *payload* that contains the content of the message (e.g., sample data or network settings). The application server contains a set of *applications*, each implementing the functionality of one user application. An application is defined by a quality of service (*QoS*); e.g., *ReliableEfficient* ensures that the transmission of data stays under a threshold, while the energy consumption of the motes is minimized.

Finally, *GUI* and *MapViewer* are two packages that support the visualization of the simulated IoT application. We show and explain the GUI below.

C. Exemplar Realization - Self-adaptation

We start with an overview of a number of adaptation scenarios for DingNet that enable researchers to compare adaptation solutions. We use one of the scenarios for illustration in the next section. Then we explain how the experimentation with self-adaptation can be done in the simulator.

1) Adaptation Scenarios: Table I summarizes the scenarios we devised for DingNet. The scenarios are organized by the type of uncertainty that requires adaptation, the type of adaptation that is required, and the types of requirements the system adaptations should met. For these scenarios, we propose metrics to evaluate and compare solutions as summarized in Table II. The definition of requirements and their evaluation is based on several sources, including evaluation frameworks for self-adaptive systems such as [21] and metrics for concrete IoT systems, such as specified in [22], [23] and [24].

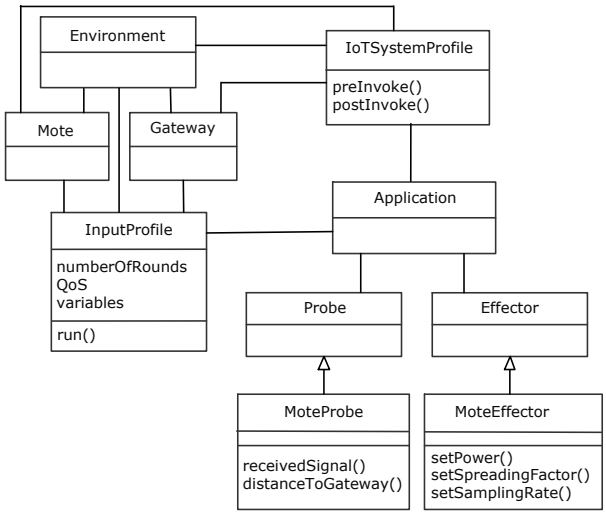

Fig. 3. Core experimentation features of DingNet simulator

2) Experimentation With DingNet: We start with the core experimentation features of the DingNet simulator. Then we explain how a concrete application can be set up with DingNet. We conclude with a snapshot of the DingNet GUI.

a) Core Experimentation Features: The features shown in Figure 3 supports the development of self-adaptive IoT applications with DingNet. The *IoTSystemProfile* allows adding behavior to the core components of the IoT system, before and after certain behavior. As an example, a profile may specify that the energy consumed by motes to communicate messages is higher to model lower quality batteries. The *InputProfile* on the other hand, allows defining application specific executions of the IoT network. This includes the number of runs of the system, the QoS that is activated, and specific variables. Two examples of variable definitions are: *the mote with number 2 will be activated with a probability of 0.6* and *the probability that additional noise is introduced due to bad weather conditions in region 1 is O.1*.

Finally, essential core features to realize a self-adaptation are the abstract classes *Probe* and *Effector*. The simulator offers a predefined instance *MoteProbe* that enables an adaption feedback loop to monitor the strength of signal received by a mote and the measured distance of a mote to the closest gateway. The predefined instance *MoteEffector* enables an adaptation feedback loop to set the power, the spreading factor, and the sampling rate of the sensors of a mote. These classes can be extended as needed, or other instantiations of the abstract probe and effector classes can be defined.

b) Setting up an IoT Application: We highlight the main classes and how they can be instantiated to set up a concrete IoT application with DingNet. The following excerpt

Scenario	Type of uncertainty	Type of adaptation	Typer of requirement
S1	Unpredictable environment: noise, mobility	Adapt power setting motes	Reliability, energy efficiency
S2	Unpredictable environment: noise, mobility, motes enter and leave the system	Adapt power motes, spreading factor motes, sampling rate motes	Reliability, energy efficiency
S3	Unpredictable environment: noise, mobility; multi-tenant setting	Adapt power and sampling rate motes; balance network resources at gateways, coordinate gateways	Cost, reliability, energy efficiency
S4	Unpredictable environment: noise, mobility, attacks; multi-tenant setting	Adapt power and sampling rate motes; balance resources at gateways, coordinate between gateways	Cost, security, reliability, energy efficiency

TABLE I
GENERIC ADAPTATION SCENARIOS FOR SMART CITY APPLICATIONS WITH DINGNET

Scenario	Type of uncertainty
Reliability	Percentage of messages lost per time unit
Energy efficiency	Power consumed by motes per number of transmissions
Cost	Price paid for resources per number of transmissions
Security	Percentage of messages compromised per time unit Percentage data of motes not delivered per time unit

TABLE II
GENERIC ADAPTATION SCENARIOS FOR IOT APPLICATIONS

illustrates how a mote can be defined.

```
public class Mote extends NetworkEntity {
  public class Mote {
  public Mote(Long DevEUI, Location location,
              Path path, Integer power,
              Integer spreadingFactor,
              Double energyLevel){
     super(DevEUI, location);
     this.path = path;
     ...
     environment.addMote(this);
     activate();
  }
  ...
}
Mote mote = new Mote(random.nextLong(),
                location, path, 5, 12, 30.0);
```

Mote extends *NetworkEntity* (not shown in Figure 2); this entity defines a unique identifier, a location, and a message queue for objects of classes derived from it. When the *activate()* method is called, the mote is fully operational to sense and communicate data and move through the environment along the path defined for it.

```
public class Gateway extends NetworkEntity {
  public Gateway(Long DevEUI, Location location){
  super(DevEUI, location);
  environment.addGateway(this);
}
Gateway gateway = new Gateway (location);
```

Gateway also extends *NetworkEntity*. The definition and instantiation of gateways is straightforward.

```
public class Environment {
    public Environment(Map map,
              Characteristic[] characteristics,
              Mote[] motes,
              Gateway[] gateways) {
    this.map = map;
    ...
}
Environment environment = new Environment(map,
        characteristics, motes, gateways);
```

The environment has a map on which motes and gateways are located. As explained above, a map allows defining regions and paths. Different characteristics can be associated with the environment, such as the transmission characteristics of particular regions in the environment.

c) DingNet GUI: Figure 4 shows a snapshot of the DingNet GUI. The top part shows a part of the area of Leuven where four gateways are located, with three motes in this case. A predefined environment can be loaded from a configuration file, or entities can be added manually. The part on the left gives an overview of the gateways and motes of the scenario that is studied. By clicking on an element, the characteristics of that element can dynamically be changed.

The lower part on the left shows the input profiles that can be selected. The knobs in the middle allow selecting an adaptation approach to be applied. Two simple adaptation strategies are provided: *Signal-based adaptation* uses the strength of the signals received by the gateways to adapt and ensure reliable and energy efficient data communication; *Distance-based Adaptation* on the other hand uses the distance of the mote to the closest gateway to adapt. There are also knobs to run a simulation, to save it, or load a previous simulation. A progress bar shows the status of a running simulation.

The graphs on the left side show a number of technical properties of a mote of the IoT system (any mote can be selected). These properties include a plot of the strength of the signals received by the different gateways as shown in the snapshot (the colored dots at the top of this graph indicate lost packets), he power setting of the mote during the run, the distance to the gateways, the spreading factor, and the energy used. The data of these plots show the averages over the complete simulation run. Detailed data stored in a file can be obtained by saving the simulation run.

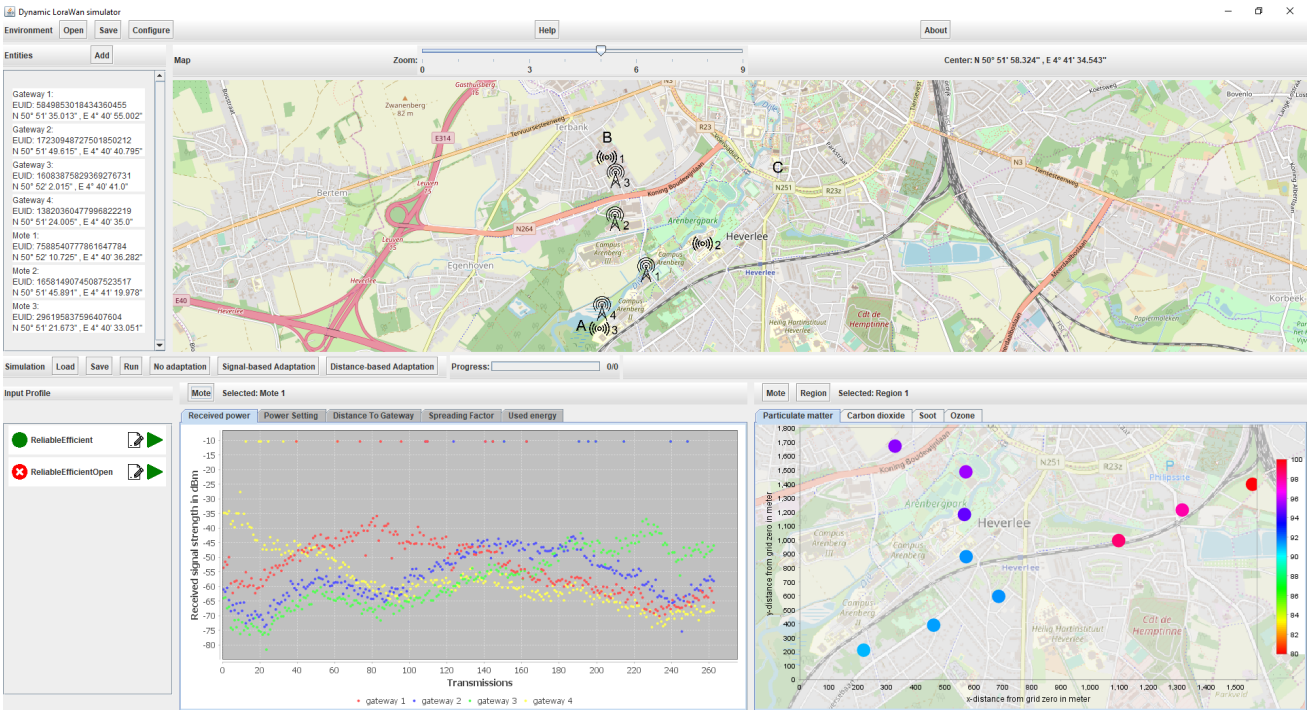

Fig. 4. General overview of the GUI of the DingNet simulator

The graphs on the left side show application-specific data. The simulator offers graphs to plot sensor data of particular matter, carbon dioxide, scoot, and ozone, and this per mote or per region in the environment (as shown in the snapshot). Again, detailed data can be saved in a file per simulation run.

III. Experimentation with DingNet

Experimentation with the DingNet artifact consists of several steps. We illustrate the steps using a concrete example.

Selection of a scenario. First a scenario is selected from Table I and concrete requirements are defined. For the example, we selected scenario $S1$ and defined the following requirements:

R1: The percentage of packets loss should be lower as 10%.

R2: Subject to R1, the total energy consumed by the motes should be minimized.

Specification of configuration. For the example case, we use a region of Leuven in the neighborhood of the Department of Computer Science (Heverlee). We consider four gateways in this region and three motes, two mobile and one stationary as shown in Figure 4. Mobile mote 1 moves from the locations A to B marked on the map (2636 m), while mobile mote moves from C to A (2972 m). Both motes move at a constant speed of 0.5 m/s. Mote 3 is stationed at location C. The mobile motes are equipped with a sensor that take every 10 meters a sample of particular matter in the air. We use the default environment settings (and associated noise in the environment) for the area where the motes move, which is defined as follows:

From A to B: 2636 m forest (pathLoss 3, ref dist 1000 m, shadowFading 1.5).

From C to B: 1600 m open space (pathLoss 2, ref dist 1000 m, shadowFading 1.5); then 1372 m forest.

Specification of the profiles. For the IoT system profile, we do not add any particular behavior to the IoT system. The input profile we use is defined as follows:

```
<inputProfile>
 <numberOfRounds>100</numberOfRounds>
 <QoS>ReliableEfficient</QoS>
 <variables>
  <variable>
   <name>mote</name>
   <values>
    <number>1</number>
      <activityProb>1.0<\activityProb>
    <number>m2</number>
      <activityProb>0.6<\activityProb>
   </values>
 </variable>
 <variable>
  <name>region</name>
  <values>
    <number>1</number>
      <weatherNoiceProb>0.1</weatherNoiceProb>
...
</variable>
...
</inputProfile>
```

This profile specifies that the application will perform 100 runs with the *ReliableEfficient* QoS requirement activated. Next, the profile specifies that the *mote* with *number* equal to 1 will be activated with a probability of 1.0 in each run; on the other hand, the *mote* with *number* equal to $m2$ will be activated with a probability of 0.6. The last part that is shown specifies that for *region* with *number* equal 1 the probability that additional noise is introduced due to bad weather conditions (*weatherNoiceProb*) is 0.1.

Definition of Metrics. We defined the following concrete metrics to evaluate self-adaption solutions:

M1. The average percentage of messages lost.
M2. The average of total energy consumed by the motes.

Implementation of self-adaptation approach. We defined two simple adaption strategies to illustrate the exemplar:

Signal-based adaptation. If the strongest signal strength of a mote received at any of the gateways is below a minimum threshold (concretely $-48\,dbm$) incrementally increase the power setting of the mote with steps of 1. If the strongest received signal strength is above a maximum threshold (concretely $-42\,dbm$) decrease the power setting with steps of 1.

Distance-based adaptation. The power setting of a mote is adapted per interval of $25\,m$ to the nearest gateway. If the distance to the nearest gateway increases to a further interval, increase the power setting with 1. If the distance to the nearest gateway decreases to a nearer interval, decrease the power setting with 1. Every $10\,m$, additional messages need to be sent to track the distance between a mote and the gateways.

We implemented a simple feedback loop for these adaptation strategies, and connected them with the standard mote probe and mote effector provided by the simulator.

Execution. Finally, we run the simulator via the input profile, and we collect and analyze the data. Figure 5 shows the adaptation of the power using signal-based adaptation. The mote dynamically adapts its power setting with changing strength of the received signals at the gateways.

Figure 6 shows the adaptation of the power using distance-based adaptation. The mote dynamically adapts its power setting when it moves closer to/further away from the gateways.

Adaptation approach	Packet Loss	Total used energy
No adaptation, power 14	9.50%	12555.5 mJ
Distance-based	5.70%	6180.6 mJ
Signal-based	5.32%	3968.8 mJ

TABLE III
GENERIC ADAPTATION SCENARIOS FOR IOT APPLICATIONS

The three approaches realize the packet loss requirement. Note that the main source of packet loss is collision between messages simultaneously sent by different motes. Signal-based adaptation clearly outperforms distance-based adaptation for

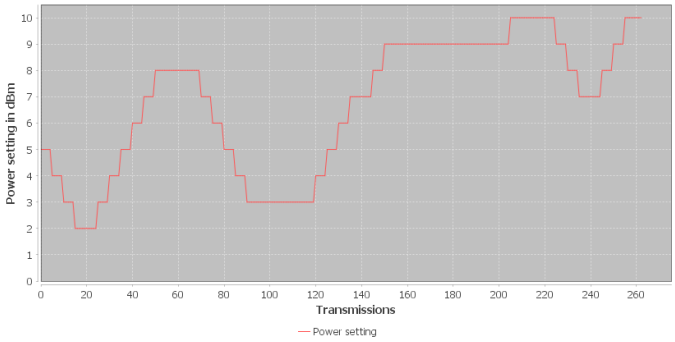

Fig. 5. Signal-based adaptation of the power setting of mote 1

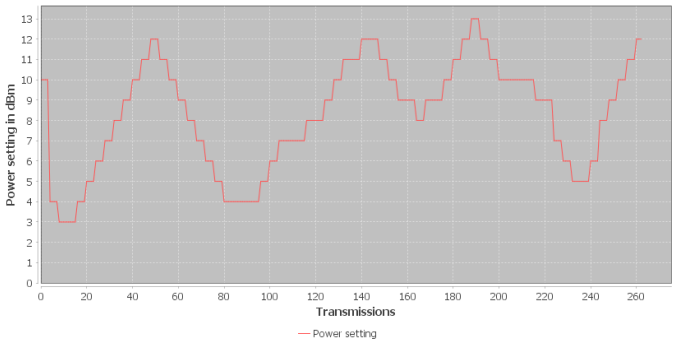

Fig. 6. Distance-based adaptation of the power setting of mote 1

energy consumption, and distance-based adaptation is much better compared to no adaptation with maximum power.

IV. Scope of Applicability

We conclude with a few words on the scope of applicability of the DingNet exemplar. DingNet targets smart city applications that build upon IoT. The primary focus is on low bandwidth applications as the amount of data that can be communicated is restricted. Uncertainties include dynamics in environment and human behavior that may be difficult to predict. Furthermore, DingNet targets applications with typical goals of IoT systems, including reliability, energy efficiency, latency, and cost. In addition to these system goals, DingNet has also the potential to support user goals, an example could be suggest alternative routes to users that aim to cycle via areas in the city with good air quality. For systems with critical goals DingNet is less applicable. A last note on the scalability of the simulator. In the experiments that we have performed so far, the simulator was very efficient. However, we tested the simulator only for smaller scale problems.

V. Conclusions and Outlook

The Internet-of-Things provides an enabling layer for smart cities applications. However, engineering such applications is challenging as these systems typically consist of a complex distributed infrastructure that is subject to various uncertainties. Self-adaptation can then offer a solution. In this paper, we proposed the DingNet exemplar to support research on self-adaptation in the IoT domain. The DingNet simulator supports researchers and engineers to design and analyze self-adaptive smart city applications that build upon the IoT. We illustrated how we used the simulator to study a typical self-adaptation problem where mobile motes dynamically adapt their power settings to ensure reliable and energy efficient communication of sensor data that is sent from the motes to the application. In the future we plan to enhance the simulator by collecting data from field experiments and use this data to bring the simulator as close as possible to the real setting. We also plan to study adaptation scenarios that are situated at the application level of smart cities, as the example about cycling via trajectories with good air quality. We hope that the research community will use the DingNet exemplar to evaluate and compare research advances in adaptive and self-managing systems, and drive their further efforts. The exemplar that is implemented in Java is freely available for experimentation [25].

REFERENCES

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, pp. 41–50, Jan. 2003.

[2] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. D. M. Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle, "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar]*, pp. 1–26, 2009.

[3] R. de Lemos, D. Garlan, C. Ghezzi, H. Giese, J. Andersson, M. Litoiu, B. R. Schmerl, D. Weyns, L. Baresi, N. Bencomo, Y. Brun, J. Cámara, R. Calinescu, M. B. Cohen, A. Gorla, V. Grassi, L. Grunske, P. Inverardi, J. Jézéquel, S. Malek, R. Mirandola, M. Mori, H. A. Müller, R. Rouvoy, C. M. F. Rubira, É. Rutten, M. Shaw, G. Tamburrelli, G. Tamura, N. M. Villegas, T. Vogel, and F. Zambonelli, "Software engineering for self-adaptive systems: Research challenges in the provision of assurances," in *Software Engineering for Self-Adaptive Systems III. Assurances - International Seminar, Dagstuhl Castle, Germany, December 15-19, 2013, Revised Selected and Invited Papers*, pp. 3–30, 2013.

[4] D. Weyns, "Software engineering for self-adaptive systems," in *Handbook of Software Engineering*, Springer, 2019.

[5] N. Esfahani, E. Kouroshfar, and S. Malek, "Taming uncertainty in self-adaptive software," in *19th Symposium and the 13th European Conference on Foundations of Software Engineering*, pp. 234–244, ACM, 2011.

[6] D. Perez-Palacin and R. Mirandola, "Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation," in *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ICPE '14, (New York, NY, USA), pp. 3–14, ACM, 2014.

[7] S. Mahdavi-Hezavehi, P. Avgeriou, and D. Weyns, "A classification framework of uncertainty in architecture-based self-adaptive systems with multiple quality requirements," in *Managing Trade-Offs in Adaptable Software Architectures*, pp. 45–77, 01 2017.

[8] R. de Lemos, D. Garlan, C. Ghezzi, H. Giese, J. Andersson, M. Litoiu, B. Schmerl, D. Weyns, L. Baresi, N. Bencomo, Y. Brun, J. Camara, R. Calinescu, M. B. Cohen, A. Gorla, V. Grassi, L. Grunske, P. Inverardi, J.-M. Jezequel, S. Malek, R. Mirandola, M. Mori, H. A. Müller, R. Rouvoy, C. M. F. Rubira, E. Rutten, M. Shaw, G. Tamburrelli, G. Tamura, N. M. Villegas, T. Vogel, and F. Zambonelli, "Software engineering for self-adaptive systems: Research challenges in the provision of assurances," in *Software Engineering for Self-Adaptive Systems III. Assurances* (R. de Lemos, D. Garlan, C. Ghezzi, and H. Giese, eds.), (Cham), pp. 3–30, Springer International Publishing, 2017.

[9] https://www.hpi.uni potsdam.de/giese/public/selfadapt/exemplars/

[10] D. Weyns and R. Calinescu, "Tele assistance: A self-adaptive service-based system exemplar," in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2015.

[11] M. Maggio, A. V. Papadopoulos, A. Filieri, and H. Hoffmann, "Self-adaptive video encoder: Comparison of multiple adaptation strategies made simple," in *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 123–128, May 2017.

[12] C. Barna, H. Ghanbari, M. Litoiu, and M. Shtern, "Hogna: A platform for self-adaptive applications in cloud environments," in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 83–87, May 2015.

[13] J. Cámara and R. de Lemos, "Evaluation of resilience in self-adaptive systems using probabilistic model-checking," in *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '12, (Piscataway, NJ, USA), pp. 53–62, IEEE Press, 2012.

[14] S. Shevtsov, M. U. Iftikhar, and D. Weyns, "Simca vs activforms: Comparing control- and architecture-based adaptation on the tas exemplar," in *Proceedings of the 1st International Workshop on Control Theory for Software Engineering*, CTSE 2015, (New York, NY, USA), pp. 1–8, ACM, 2015.

[15] F. Krijt, Z. Jiracek, T. Bures, P. Hnetynka, and I. Gerostathopoulos, "Intelligent ensembles - a declarative group description language and java framework," in *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 116–122, May 2017.

[16] M. U. Iftikhar, G. S. Ramachandran, P. Bollanse, D. Weyns, and D. Hughes, "Deltaiot: A self-adaptive internet of things exemplar," in *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 76–82, May 2017.

[17] D. Weyns, M. U. Iftikhar, D. Hughes, and N. Matthys, "Applying architecture-based adaptation to automate the management of internet-of-things," in *Software Architecture - 12th European Conference on Software Architecture, ECSA 2018, Madrid, Spain, September 24-28, 2018, Proceedings*, pp. 49–67, 2018.

[18] M. C. Bor, U. Roedig, T. Voigt, and J. M. Alonso, "Do lora low-power wide-area networks scale?," in *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '16, (New York, NY, USA), pp. 59–67, ACM, 2016.

[19] B. Reynders, Q. Wang, and S. Pollin, "A lorawan module for ns-3: Implementation and evaluation," in *Proceedings of the 10th Workshop on Ns-3*, WNS3 '18, (New York, NY, USA), pp. 61–68, ACM, 2018.

[20] J. Petajajarvi, K. Mikhaylov, A. Roivainen, T. Hanninen, and M. Pettissalo, "On the coverage of lpwans: range evaluation and channel attenuation model for lora technology," in *ITS Telecommunications (ITST), 2015 14th International Conference on*, pp. 55–59, IEEE, 2015.

[21] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas, "A framework for evaluating quality-driven self-adaptive software systems," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, (New York, NY, USA), pp. 80–89, ACM, 2011.

[22] M.-A. Nef, L. Perlepes, S. Karagiorgou, G. I. Stamoulis, and P. K. Kikiras, "Enabling qos in the internet of things," in *Proc. of the 5th Int. Conf. on Commun., Theory, Reliability, and Quality of Service (CTRQ 2012)*, pp. 33–38, 2012.

[23] A. Dvornikov, P. Abramov, S. Efremov, and L. Voskov, "Qos metrics measurement in long range iot networks," in *Business Informatics (CBI), 2017 IEEE 19th Conference on*, vol. 2, pp. 15–20, IEEE, 2017.

[24] R. Berrevoets and D. Weyns, "A qos-aware adaptive mobility handling approach for lora-based iot systems," in *2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pp. 130–139, Sep. 2018.

[25] https://people.cs.kuleuven.be/danny.weyns/software/DingNet/index.htm